Control of manufacturing systems with downstream batch processors

Cheng, Tajan John Benedict

2007

Cheng, T.J. B. (2007). Control of manufacturing systems with downstream batch processors. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/13611

https://doi.org/10.32657/10356/13611

Control of Manufacturing Systems with Downstream Batch Processors

By

Tajan John Benedict Cheng

B.S. in Manufacturing Engineering and Management
De La Salle University, 2002
S.M. in Innovation in Manufacturing Systems and Technology
Nanyang Technological University, 2004

SUBMITTED TO THE SMA OFFICE IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN INNOVATION IN MANUFACTURING SYSTEMS AND TECHNOLOGY (IMST)
AT THE
SINGAPORE-MIT ALLIANCE
2007

Signature of author: . . . . . .
IMST Programme
February 28, 2008

Certified by: . . . . . .

Assoc. Prof. Appa Iyer Sivakumar	Dr. Stanley Gershwin
SMA Fellow, NTU	SMA Fellow, MIT
Thesis Advisor	Thesis Advisor

Accepted by: . . . . . .

Prof. Yue Chee Yoon	Prof. David Hardt

Programme Co-Chair
IMST Programme

Programme Co-Chair
IMST Programme

Control of Batch Processors in Manufacturing Systems

By

Tajan John Benedict Cheng

Submitted to the SMA office
in Partial Fulfillment of the Requirements for the
degree of Doctor of Philosophy
in Innovation in Manufacturing Systems and Technology (IMST)

ABSTRACT

Batch processors can concurrently process more than one batch and are regularly used in the semiconductor manufacturing industry, particularly in the wafer fabrication stage. The reentrant nature of the wafer fab and the incompatibility of jobs from different job families, combined with the typically long processing times of the batch processor, make the control of batch processors an interesting problem with important implications.

Current literature has predominantly assumed that the batch processor exists in isolation, and treats the job arrival pattern it receives as a problem constraint. We propose to control the job arrival pattern of the batch processors by controlling the processor immediately upstream of the batch processor, such that the performance of the over-all system is improved.

We initially show that increasing the correlation of the job families of incoming jobs reduces the mean queue time at the batch processor buffers, regardless of batch processor policy. This initial finding validates our hypothesis. We also examine the problem of controlling the upstream processor when jobs have a queue time limit at the batch processor buffer. Although this problem is prevalent in wafer fabs, to the author's knowledge it has yet to be analyzed. Finally we consider two-and three-stage systems with a downstream batch processor under simple control policies. After characterizing the long-term performance of the system, we also determine that constraining the upstream processor to process jobs belonging to the same job family the batch processor will process next can reduce the over-all mean cycle time of jobs passing through the system, compared to an upstream processor policy that is strictly myopic to the anticipated needs of the batch processor. This observation is true whether the upstream processor is a serial processor or another batch processor with smaller size.

The work presented not only analyses several small-scale issues regarding the control of batch processors inside the wafer fab, but also serves as the foundation for future research on multi-stage policies for more complex systems with batch processors.

Keywords: production control, batch processor, multiple-stage system

Thesis Advisors:
1. Assoc. Prof. Appa Iyer Sivakumar, SMA Fellow, NTU
2. Dr. Stanley Gershwin, SMA Fellow, MIT

Acknowledgments

The author is grateful for the guidance provided by his thesis advisers, Associate Professor Appa Iyer Sivakumar and Doctor Stanley Gershwin, throughout the duration of the research. Without their help, the author would have dropped out with probability 99.9%, and won the lottery, and consequently dropped out, with probability 0.1%.

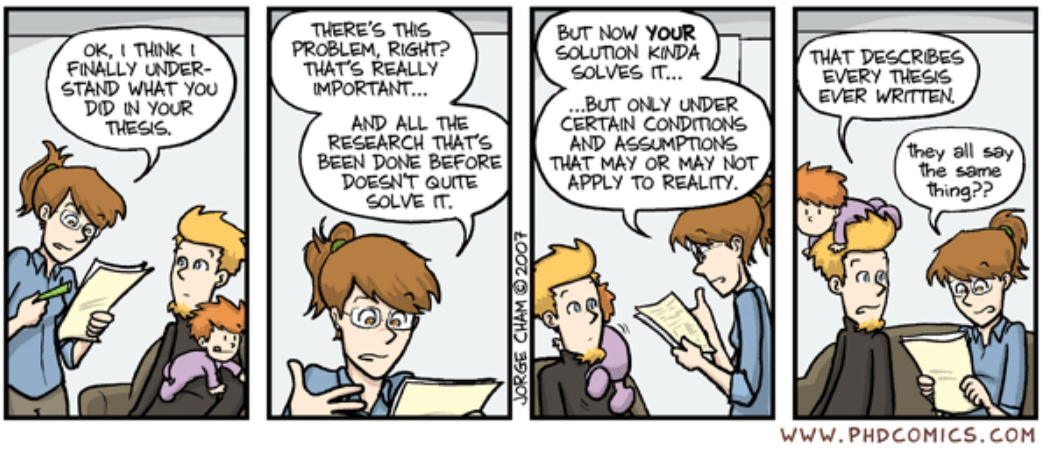
Dr. Mathi Mathirajan has also provided assistance, particularly in the organization of the dissertation. The author appreciates the interaction with the industrial engineering department of Chartered Semiconductor Manufacturing, for providing industry perspective on the control problems.

Understanding the work of others is difficult; the author thanks the people who took time off their busy schedules to answer his queries: Prof. Richard Serfozo, Prof. Reha Uzsoy, Prof. Villiam Makis, Prof. Jan Karel Lenstra, Prof. S.H. Chung, Prof. John Fowler, Asst. Prof. Seok-Ho Chang, Prof. Toshi Ibaraki, Dr. John Neale, Dr. Jennifer Robinson, Dr. Qi Chao and Dr. Amit Gupta.

The people comprising the NTU and MIT SMA offices deserve praise for the amount of work they do behind the scenes. The life of a research student is monotonous in isolation. The author appreciates the hospitality extended to him by fellow SMA students. Perhaps misery just loves company?

The author is grateful for the funding provided by the Singapore-MIT Alliance for this research study, although the author would be more thankful if the stipend was adjusted to account for inflation.

The author thanks his family for their support throughout the candidature.



"Piled Higher and Deeper" by Jorge Cham www.phdcomics.com

Table of Contents

Acknowledgments 5

Table of Contents 7

List of Figures Used 15

List of Tables Used 19

List of Publications 23

List of Symbols Used 25

List of Definitions 29

Chapter 1 Introduction 33

1.1 Introduction 33

1.2 Background of Research Problem 35

1.2.1 Primer on semiconductor manufacturing process 35

1.2.1.1 Wafer Fabrication 35

1.2.1.2 Wafer Test 37

1.2.1.3 Integrated Chip (IC) Assembly 37

1.2.1.4 IC Test 37

1.2.2 Importance of batch processors 38

1.3 Research Motivation 39

1.4 Research Objective 40

1.5 Organization of Chapters 42

Chapter 2 Literature Review 43

2.1 Introduction to Batch Processors in Semiconductor Manufacturing 43

2.2 Batch Processors with Incompatible Job Families (Wafer Fab Model) 47

2.2.1 Batch processor as a single stage 47
2.2.1.1 Optimal control of a single stage of batch processors 48
2.2.1.1.1 Finite Horizon Problems 48
2.2.1.1.2 Infinite Horizon Problems 51
2.2.1.2 Estimation of performance measures 57
2.2.2 Multi-stage models involving a batch processor 58
2.2.2.1 Explicit enumeration of system structure 59
2.2.2.1.1 Prediction of Performance Measures 59
2.2.2.1.2 Finite Horizon Optimization Problems 60
2.2.2.1.3 Infinite horizon optimization problems 61
2.2.2.2 Generalized system structure 63
2.2.2.2.1 Prediction of performance measures 63
2.2.2.2.2 Optimal Control 64
2.3 Batch Processors as Burn-in Ovens 66
2.3.1 Burn-in ovens as a single entity 66
2.3.1.1 Finite Horizon Problems 66
2.3.1.1.1 Deterministic Problems 67
2.3.1.1.2 Stochastic problems 69
2.3.1.2 Infinite Horizon Problems 69
2.3.1.2.1 Decomposition into static problem 69
2.3.1.2.2 Threshold policies 69
2.3.1.2.3 Look-ahead policies 70
2.3.2 Multi-stage models involving a burn-in oven 70
2.3.2.1 Explicit enumeration of system structure 70
2.3.2.2 Generalized system structure 71
2.4 Area of Contribution of Research 71
2.5 Chapter Summary 73
Chapter 3 Control of a Batch Processor in Isolation 75
3.1 Introduction 75
3.2 Use of Offline Deterministic Methods for Online Batch Processor Control in a Stochastic Environment 76
3.3 Complexity of the Deterministic Finite-Horizon Optimal Batch Processor Scheduling Problem 77
3.3.1 Problem Statement 77
3.3.2 Problem Complexity 77
3.3.3 Dynamic Programming Algorithm 79
3.3.3.1 Theoretical Complexity of Algorithm 81
3.3.3.2 Empirical Analysis of Algorithm Complexity 85
3.3.3.3 Discussion of Results 87

3.4 Approximation Method for On-line Batch Processor Scheduling Problem in a Stochastic Environment: Model Predictive Control 89
3.4.1 Description of MPC Algorithm 89
3.4.2 Performance of MPC-based Online Algorithm 90
3.4.2.1 Experimental Design 91
3.4.2.2 Performance Evaluation of Approximate DP Algorithm 94
3.4.2.2.1 Deterministic Arrival Times with Uncorrelated Job Families 95
3.4.2.2.2 Deterministic Arrival Times with Correlated Job Families 100
3.4.2.2.3 Stochastic Arrival Times with Uncorrelated Job Families 105
3.5 Chapter Summary 110

Chapter 4 Control of Job Arrivals into Batch Processor Buffer 113
4.1 Introduction 113
4.2 Problem Statement 115
4.3 Objective Function Discussion 116
4.4 Problem Methodology 118
4.4.1 Batch Processor Control Policies Evaluated 118
4.5 Notation used 119
4.6 Model One Assumptions and Development 120
4.6.1 Model One Assumptions 120
4.6.2 Model One Development 121
4.6.2.1 When the Batch Processor is under No-Idling Policy 121
4.6.2.1.1 Case One: $X_t = (0, 0)$ 121
4.6.2.1.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$ 121
4.6.2.1.3 Case Three: $X_t = (> Q\text{-}1, 1)$ 122
4.6.2.2 When the Batch Processor is under Full-Batch Policy 123
4.6.2.2.1 Case One: $X_t = (0 \text{ to } Q - 1, 0)$ 123
4.6.2.2.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$ 123
4.6.2.2.3 Case Three: $X_t = (> Q\text{-}1, 1)$ 123
4.6.3 Model One Discussion 124
4.7 Model Two Assumptions and Development 127
4.7.1 Model Two Assumptions 127
4.7.2 Model Two Development 127
4.7.2.1 When the Batch Processor is under No-Idling Policy 128
4.7.2.1.1 Case One: $X_t = (0, 0)$ 128
4.7.2.1.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$ 128
4.7.2.1.3 Case Three: $X_t = (> Q - 1, 1)$ 130
4.7.2.2 When the Batch Processor is under Full-Batch Policy 135
4.7.2.2.1 Case One: $X_t = (0 \text{ to } Q - 1)$ 135
4.7.2.2.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$ 137
4.7.2.2.3 Case Three: $X_t = (> Q\text{-}1, 1)$ 138

4.7.3 Model Two Discussion 138
4.7.3.1 When the batch processor is under no-idling policy 138
4.7.3.2 When the batch processor is under full-batch policy 143
4.8 Model Three Assumptions and Development 146
4.8.1 Model Three Assumptions 146
4.8.2 Model Three Development 147
4.8.2.1 When the Batch Processor is under No-Idling Policy 148
4.8.2.1.1 When the Batch Processor is Initially Up 148
4.8.2.1.2 When the Batch Processor is Initially Down 163
4.8.2.2 When the Batch Processor is under Full-Batch Policy 169
4.8.2.2.1 When the Batch Processor is Initially Up 170
4.8.2.2.2 When the Batch Processor is Initially Down 171
4.8.3 Model Three Discussion 172
4.8.3.1 Comparison of Optimal Policy for Different Initial Batch Processor States 172
4.8.3.1.1 Batch processor is under no-idling policy 172
4.8.3.1.2 Batch processor is under full-batch policy 174
4.8.3.2 Comparison of Optimal Policy for Different Batch Processor Policies 176
4.9 Model Four Assumptions and Development 177
4.9.1 Problem formulation for Model Four 177
4.9.2 Model Four Assumptions 178
4.9.3 Model Four Development 178
4.9.4 Model Four Discussion 182
4.9.4.1 Numerical Experimentation 183
4.10 Characterization of Optimal Feeder Processor Policy when Batch Processor is Reliable and has Geometric Processing Time 185
4.11 Extension of Results to Different Systems 186
4.12 Chapter Summary 188
Chapter 5 Control of Upstream Serial Processor with a Downstream Batch Processor 191
5.1 Introduction 191
5.2 Problem Statement for Two-stage Model 192
5.3 Model Development 193
5.3.1 Development of Two-Stage Markovian Model 194
5.3.1.1 Model Assumptions 194
5.3.1.2 State Representation 195
5.3.1.3 Serial Processor Control Policies 196
5.3.1.3.1 Myopic control policy 196
5.3.1.3.2 Greedy-look-ahead policy 197
5.3.1.3.3 Balanced-look-ahead policy 197

5.3.1.4 Batch Processor Control Policies 197
5.3.1.5 State Space Discussion under Differing Batch Processor Policies 199
5.4 Development of Three-Stage Markovian Model 200
5.4.1 Problem formulation for Three-stage Model 200
5.4.2 Model Assumptions 201
5.4.3 Processor Control Policies 202
5.4.4 State Representation 202
5.5 Verification of Markovian Models-Comparison with Simulation 203
5.5.1 Comparison Methodology 203
5.5.2 Comparison Results 205
5.6 Behavior of Two-Stage System 207
5.6.1 Experimental Design 207
5.6.2 Sensitivity of Model to Increasing Buffer Size 210
5.6.3 Sensitivity of Model to Increasing Batch Processor Size 214
5.6.4 Sensitivity of Model to Different Batch Processor Policies 218
5.6.5 Sensitivity of Model to Different Serial Processor Policies 220
5.7 Behavior of Three-Stage System 228
5.7.1 Experimental Design 229
5.7.2 Sensitivity of Model to Different Serial Processor Policies 229
5.8 Chapter Summary 232

Chapter 6 Control of Upstream Batch Processor with a Downstream Batch Processor 235
6.1 Introduction 235
6.2 Problem Statement 235
6.3 Two-stage Model Development 236
6.3.1 Development of Two-Stage Markovian Model 237
6.3.1.1 Model Assumptions 237
6.3.1.2 Feeder Processor Control Policies 237
6.3.1.2.1 Myopic control policy variant 238
6.3.1.2.2 Greedy look-ahead policy 238
6.3.1.2.3 Balanced look-ahead policy 238
6.3.1.3 Batch Processor Control Policies 239
6.3.1.4 State Representation 239
6.3.2 Development of Three-Stage Markovian Model 240
6.3.2.1 Problem formulation for three-stage model 240
6.3.2.2 Model Assumptions 241
6.3.2.3 Processor Control Policies 242
6.3.2.4 State Representation 242

6.3.3 Verification of Markovian Models-Comparison with Simulation 243
6.3.3.1 Comparison Methodology 244
6.3.3.2 Comparison Results 245
6.4 Behavior of Two-Stage System 247
6.4.1 Experimental Design 247
6.4.2 Behavior of System when Batch Processor is under No-Idling Policy 249
6.4.2.1 Feeder Processor is under No-Idling Policy 249
6.4.2.2 Feeder Processor is under Full-Batch Policy 250
6.4.3 Behavior of System when Batch Processor is under Full-Batch Policy 256
6.4.3.1 Feeder Processor is under No-Idling Policy 256
6.4.3.2 Feeder Processor is under Full-Batch Policy 257
6.4.4 Sensitivity of Model to Different Feeder Processor Policies 259
6.5 Behavior of Three-Stage System 263
6.5.1 Experimental Design 264
6.5.2 Sensitivity of Model to Different Serial Processor Policies 264
6.6 Chapter Summary 269

Chapter 7 Discussion of Conclusions and Recommendations 271
7.1 Summary of Conclusions 272
7.1.1 Control of the batch processor in isolation 272
7.1.2 Control of job arrivals into the batch processor buffer 273
7.1.3 Performance of a serial - batch processor system under simple control policies 276
7.1.4 Performance of a batch - batch processor system under simple control policies 279
7.2 Contributions 280
7.3 Limitations 283
7.4 Future Research Directions 284

List of References 285

Appendix A Control of multiple batch processors in a single stage 295
A.1 Problem Statement 295
A.2 Integer Linear Programming Formulation 296
A.3 DP Algorithm 298
A.3.1 State representation 298
A.3.2 Set of feasible controls 299
A.3.3 State transition 299
A.4 Complexity Analysis 301
A.5 Suggested Heuristic Development 302
A.5.1 Adaptation of the MPC-based heuristic to multiple processors 303

A.5.2 Experimental Setup for Performance Evaluation of proposed policies 306
A.5.3 Discussion of Simulation Results 307
A.5.4 Conclusion 317

Appendix B Simulation Results for Evaluating MPC-Based Heuristic Performance 319
B.1 Experiment 1: System has deterministic arrival 319
B.2 Experiment 2: System has positive correlation between job families of consecutive job arrivals 320
B.3 Experiment 3: System has imperfect prediction of arrival times 321

Appendix C Transition probability equations for two-stage models, with the serial processor under a threshold policy 323
C.1 Model One: Serial processor processing time is deterministic, batch processor processing time is geometric 324
C.2 Model Two: Processing time of both processors are geometric 324
C.3 Model Three: Processing time of both processors are geometric, and the batch processor is unreliable 326

Appendix D Probability Transition Equation Flowcharts for Two-Stage Markov Models 329
D.1 Serial Processor feeding Batch Processor 330
D.2 Two Batch Processors in series 334

Appendix E Probability Transition Equation Flowcharts for Three-Stage Markov Models 339
E.1 Serial-Serial-Batch Processor System 340
E.2 Serial processor feeds (feeder) batch processor, which feeds batch processor 345

List of Figures Used

Figure 1.1: Estimated 2005 revenues and forecasted 2008 revenues (B refers to billions) for various stages of semiconductor manufacturing process. 33

Figure 1.2: Estimated time line for the semiconductor manufacturing process. 35

Figure 2.1: Proposed division of current literature on batch processor behavior and control. 44

Figure 2.2: High-level division of the discussion of previous work on batch processors in semiconductor manufacturing. 46

Figure 2.3: Classification scheme for literature on batch processors in isolation. 47

Figure 2.4: Classification scheme for multi-stage systems with a wafer fab type batch processor. This figure is a subsection of Figure 2.1. 59

Figure 2.5: Classification scheme for literature on burn-in ovens in isolation..... 66

Figure 2.6: Visual summary of contributions of this dissertation. 73

Figure 3.1: A sample 3-ary tree that illustrates node trimming in relation to Conditions A, B and C. 84

Figure 3.2: Growth of computational time as job family quantity is increased. 88

Figure 3.3: Growth of computational time as the number of jobs are increased. 89

Figure 3.4: Difference in horizon selection between the two policies. 99

Figure 3.5: Difference between MPC- based heuristic and NACHM. 99

Figure 3.6: Evolution of mean cycle time for the three heuristics upon increase in correlation coefficient between job families of successive job arrivals. 101

Figure 3.7: Evolution of mean cycle time as the error coefficient is increased. 106

Figure 4.1: A subsection of a process flow inside the wafer fab. 114

Figure 4.2: Simple 2-stage manufacturing model. 116

Figure 4.3: Summarizing the differences between the four models investigated. 120

Figure 4.4: Probability of the prospective job incurring a penalty as a function of Z_t when the batch processor is busy and under no-idling policy..... 125

Figure 4.5: Markov Chain of Model One system under threshold policy, with the batch processor under no-idling policy. 126

Figure 4.6: Markov Chain of Model One system under threshold policy, with the batch processor under full-batch policy. 126

Figure 4.7: Relationship between Scenarios One and Two under Case 3.2. 131

Figure 4.8: Probability of the prospective job incurring a penalty when the batch processor is idle and under full-batch policy. 136

Figure 4.9: Why the probability of incurring a penalty is non-decreasing when the batch processor is under no-idling policy. 139

Figure 4.10: Probability of the prospective job incurring a penalty when the batch processor is busy and under no-idling policy. 142

Figure 4.11: Total probability of the prospective job incurring a penalty when the batch processor is busy and under full-batch policy. 143

Figure 4.12: Total probability, and approximate probability plot, of the prospective job incurring a penalty when the batch processor is busy and under full-batch policy. 144

Figure 4.13: Comparing probabilities of the prospective job incurring a penalty when the batch processor is busy and under differing policies. 145

Figure 4.14: Simple 2-stage manufacturing model that allows for the possibility of the serial processor being starved of WIP. 178

Figure 4.15: Probability of incurring penalty may not be strictly decreasing with increased serial processor processing rate. 184

Figure 4.16: The probability of incurring a penalty, according to the queue length of the serial processor buffer, when the batch processor is busy. 185

Figure 5.1: A two-stage manufacturing system with a serial-batch configuration. The system is processing two job families, and has two buffers, one for each job family, in front of every processor. 193

Figure 5.2: A three-stage manufacturing system with a serial-serial-batch configuration. 201

Figure 5.3: Comparing the cycle time and throughput from Markov and simulation models, when the policy is myopic-myopic-no-idling or myopic-myopic-full-batch. 207

Figure 5.4: Throughput as a function of the batch processor traffic intensity. 210

Figure 5.5: Throughput as a function of the buffer size. 212

Figure 5.6: Cycle time as a function of the batch processor traffic intensity. 212

Figure 5.7: Throughput as a function of the buffer size. 213

Figure 5.8: Throughput as a function of the batch processor traffic intensity. 215

Figure 5.9: Mean cycle time of the system as a function of the batch processor traffic intensity. 216

Figure 5.10: Mean cycle time of the system as a function of the traffic intensity for a balanced system under myopic-full-batch policy. 217

Figure 5.11: Mean cycle time of the system as a function of the traffic intensity for a balanced system under myopic-no-idling policy. 217

Figure 5.12: Throughput versus cycle time for various two-stage systems with the serial processor under myopic policy. 220

Figure 5.13: Throughput versus cycle time performance of systems with different serial processor policies when the batch processor is under no-idling. 221

Figure 5.14: Throughput versus cycle time performance of systems with different serial processor policies when the batch processor is under full-batch. 222

Figure 5.15: Performance comparison between the combined and the greedy-look-ahead policy. 225

Figure 5.16: Sensitivity of the relative performance of two serial processor control policies to the mean processing rate of the serial processor, when the batch processor is under no-idling policy. 226

Figure 5.17: Sensitivity of the relative performance of two serial processor control policies to the mean processing rate of the serial processor, when the batch processor is under full-batch policy. 227

Figure 5.18: Performance of a three-stage system with two upstream processors and a downstream batch processor, with the batch processor is under no-idling policy. 230

Figure 5.19: Performance of a three-stage system with two upstream processors and a downstream batch processor. 232

Figure 6.1: A two-stage manufacturing system with a batch-batch configuration. The system is processing two job families, and has two buffers, one for each job family, in front of every processor. 236

Figure 6.2: A three-stage manufacturing system with a serial-batch-batch configuration. 241

Figure 6.3: Comparing the cycle time and throughput from Markov and simulation models, when the policy is myopic-(myopic-no-idling)-no-idling or myopic-(myopic-no-idling)-full-batch. 247

Figure 6.4 : The plot of cycle time versus throughput for a two-stage system with two batch processors. 250

Figure 6.5: The plot of cycle time versus throughput for a two-stage batch-batch system. 253

Figure 6.6: Performance plot for four different system configurations. 254

Figure 6.7: Detailed plot of systems A1 and A2 in Figure 6.6, obtained through simulation. 255

Figure 6.8: Cycle time versus throughput for a two-stage batch-batch system. 257

Figure 6.9: Cycle time versus throughput for a two-stage batch-batch system. 258

Figure 6.10: Performance plot for four different system configurations. 258

Figure 6.11: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three, and buffer capacities are seven. 260

Figure 6.12: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three, and buffer capacities are seven. 261

Figure 6.13: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three and buffer capacities are seven. 262

Figure 6.14: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three and buffer capacities = seven. 263

Figure 6.15: Performance of several policy combinations for a three stage system when both the feeder and batch processors are under full-batch policy. 265

Figure 6.16: Performance of several policy combinations for a three stage system when only the feeder processor is under no-idling policy. 266

Figure 6.17: Performance of several policy combinations for a three stage system when only the feeder processor is under full-batch policy. 267

Figure 6.18: Performance of several policy combinations for a three stage system, when both the feeder and batch processor are under no-idling policy. 267

Figure A- 1: The difference between mean values under Policy One and NACH-MM. 308

Figure A- 2: Mean cycle time values for NACH-MM. 310

Figure A- 3: Comparing the mean cycle time for the three policies when the job families of successive job arrivals are uncorrelated. 315

List of Tables Used

Table 2.1: Summary of key Look-ahead policy Attributes 57
Table 2.2: Deterministic finite horizon scheduling problems with a single Burn-in. 67
Table 3.1: Time required to perform Naive and Improved DP algorithms 86
Table 3.2: Experimental Setup to Determine Algorithm Complexity 87
Table 3.3: Mean Computational Time for Naïve DP Algorithm in Seconds..... 87
Table 3.4: Experimental set-up summary for evaluation of MPC-based heuristic..... 94
Table 3.5: Paired T-Test results for Experiment One: Uncorrelated Job Arrivals..... 95
Table 3.6: Comparing (4, 8) with NACHM when there are eight families 98
Table 3.7: Paired T-Test results when CORR(X,Y) = 0.25..... 101
Table 3.8: Paired T-Test results when CORR(X,Y) = 0.7..... 102
Table 3.9: Estimated percentage reduction in mean cycle time if correlation of job families of successive job arrivals are increased from 0.0 to 0.25..... 103
Table 3.10: Estimated percentage reduction in mean cycle time if correlation of job families of successive job arrivals are increased from 0.25 to 0.7..... 104
Table 3.11 Paired T-Test results when Error coefficient Ω = 0.1. 106
Table 3.12: Paired T-Test Results when Error Coefficient Ω = 0.3. 107
Table 3.13: Paired T-Test results when Error Coefficient Ω = 0.5..... 108
Table 3.14: Settings for which a significant difference exists in the differences between mean cycle times of two policies..... 109
Table 4.1: Division of Analysis for Model One..... 121
Table 4.2: Division of Analysis for Model Two..... 128
Table 4.3: Experimental setup to support hypothesis 142
Table 4.4: Division of Analysis for Model Three 147
Table 4.5: Experimental setup to compare probabilities of incurring a penalty according to initial batch processor state 173
Table 4.6: Transition equations for modified m/m/1 queue 181
Table 4.7: Main numerical experimental setup parameters 183
Table 5.1: Two-stage simulation models compared with Markov models 204
Table 5.2: Three-stage simulation models compared with Markov models..... 205

Table 5.3: Comparison between two-stage Markov model and simulation models.....206
Table 5.4: Comparison between three-stage Markov model and simulation models.....206
Table 5.5: Experimental setup summary for Two-Stage System.....208
Table 5.6: Supplementary Experimental Setup.....225
Table 5.7: performance of the greedy look-ahead policy, relative to myopic policy.....226
Table 5.8: Experimental setup for Three-Stage System.....229
Table 5.9:Relative performance of policy combinations versus myopic-myopic for three stage system with batch processor under no-idling policy.....230
Table 5.10:Relative performance of policy combinations versus myopic-myopic for three stage system with batch processor under full-batch policy.....232
Table 6.1: Two-stage simulation models compared with Markov models.....244
Table 6.2: Three-stage simulation models compared with Markov models.....245
Table 6.3: Comparison between Markov model and simulation models.....245
Table 6.4: Comparison between Results of Markov Models and simulation models.....246
Table 6.5: Experimental setup summary for Two-Stage System.....248
Table 6.6: Relative performance of greedy look-ahead policy over myopic policy for balanced two-stage system with two batch processors.....261
Table 6.7: Experimental setup for Three-Stage System.....264
Table 6.8: Performance of three policy combinations relative to that of myopic-myopic policy combination.....268
Table A- 1: Values given to various system parameters for comparing policies.....306
Table A- 2: Summary of Data comparing Policy one and NACH-MM.....307
Table A- 3: Effect of increased correlation on performance of NACH-MM.....309
Table A- 4: Effect of increased correlation on performance of Policy One.....311
Table A- 5: Effect of increased correlation on performance of Policy Two.....312
Table A- 6: Effect of increased correlation on performance of Policy Three.....313
Table A- 7: Summary of Data comparing Policy two and NACH-MM.....314
Table A- 8: Summary of Data comparing Policy three and NACH-MM.....314
Table A- 9: Settings where Policies One, Two and Three have significantly different performance.....316
Table B- 1: Mean cycle time values of heuristics under deterministic arrival times.....319
Table B- 2: Mean cycle time values of heuristics when CORR(X,Y) = 0.25.....320
Table B- 3: Mean cycle time values of heuristics when CORR(X,Y) = 0.7.....320
Table B- 4: Mean cycle time values of the heuristics when Ω = 0.1.....321
Table B- 5: Mean cycle time values of the heuristics Ω = 0.3.....321

Table B- 6: Mean cycle time values of the heuristics when Ω = 0.5. 322

Table C- 1: Model One transition probabilities with Batch Processor under No-Idling Policy 324

Table C- 2: Model One transition probabilities with Batch Processor under Full-Batch Policy 324

Table C- 3: Model Two transition probabilities with Batch Processor Under No-Idling Policy 325

Table C- 4: Model Two transition probabilities with Batch Processor Under Full-Batch Policy 325

Table C- 5: Model Three transition probabilities with Batch Processor Under No-Idling Policy 326

Table C- 6: Model Three transition probabilities with Batch Processor Under Full-Batch Policy 327

List of Publications

Portion of the work in this dissertation has been discussed in the following conferences.

- "Performance of a serial-batch processor system with incompatible job families under simple control policies", Singapore-MIT Alliance Annual Symposium, January, 2006.
- "Control of Job Arrivals with Processing Time Windows into Batch Processor Buffer", Singapore-MIT Alliance Annual Symposium, January, 2007.

List of Symbols Used

Q_i size of the i^{th} batch processor. Subscript is dropped when sizes are identical or when only one batch processor is present in the system.

λ average arrival rate of jobs.

For the reader's ease, we segregate symbols according to the chapter it was used, if the use of the symbol is exclusive to a certain chapter.

Chapter 3 (and Appendix A) Symbols

n number of jobs to be scheduled

m number of job families

f_j set containing the jobs that belong to family j. If $f_1 = \{1, 2, 3, 4\}$, Only Jobs One, Two, Three and Four belong to Job Family One.

r_i time instance Job i arrives in front of the batch processor.

p_j processing time for Job Family j

C_k time instance Batch k finishes

S_k time instance Batch k starts

x_{ik} 1 if Job i is assigned to Batch k and 0 otherwise

y_{jk} 1 if Job Family j is assigned to Batch k and 0 otherwise

t time instance decision has to be made

$q_j(t)$ queue length for jobs belonging to Family j at time instance t.

a_i job family to which Job i belongs to. $a_i = \{j \mid i \in f_j\}$.

r_i time Job i arrives in front of the batch processor.

n_j total number of jobs to be scheduled that belong to Job Family j. $\sum_{j=1}^{m} n_j = n$

$g_j(t)$ number of jobs belonging to Job Family j still in front of the batch processor right after a batch of Job Family j is loaded.

$h_j(t)$ number of jobs belonging to Job Family j loaded into the batch processor.

$s(t)$ time instance of the next job arrival.

$V_j(x, y)$ number of jobs belonging to Job Family j that arrives in front of the batch processor in the time interval $(x, y]$.

F number of job families to consider for the MPC-based heuristic

H maximum number of future job arrivals to consider for the MPC-based heuristic

φ probability that the second job Y will belong to the same job family as the first job X.

K number of identical batch processors in parallel at the batch processing stage.

z_{lk} 1 if l^{th} batch is assigned to batch processor k and 0 otherwise

$D_k(t)$ time instance Processor k will be available. If, at Time Instance t, Processor k is idle, then $D_k(t) = 0$.

$d_k(t)$ amount of processing time assigned to Batch Processor k at Time Instance t. If Batch Processor k is to be left idle, or if Batch Processor k is busy and unable to process a new batch, then $d_k(t) = 0$.

$y_j(t)$ number of jobs belonging to Job Family j that are loaded into processors at Time Instance t.

$\varphi(t)$ earliest time a batch processor that is busy at Time Instance t will become available. If $\{D_k(t) | D_k(t) > 0\}$ is NULL, set $\varphi(t) = infinity$.

$\alpha(t)$ earliest time instance a processor that starts processing at Time Instance t becomes idle. If $\{d_k(t) | d_k(t) > 0\}$ is NULL, set $\alpha(t) = infinity$.

$c_k(t)$ flag variable to determine whether Processor k becomes idle at Time Instance t.

Chapter 4 (and Appendix C) Symbols

R	Reward for serial processor processing a job
C	Penalty incurred by a job reaching its processing time window
T	processing time window length
P_S	Probability of a serial processor finishing a job within a time instance, if processor is busy.
P_B	Probability of a batch processor finishing a job within a time instance, if processor is busy and does not fail.
Z_t	Number of jobs inside the batch processor buffer at time instance t
S_t	Status of the batch processor at time instance t.
F_t	Status of the serial processor at time instance t.
X_t	System state at time instance t
P_A	Probability of a job arrival into the serial processor buffer within a time instance.
$P(i)$	Probability of i arrivals into the serial processor buffer while the serial processor is processing prospective job
$P_{(i)}$	Probability of prospective job incurring a penalty if there were i arrivals into the serial processor buffer while the prospective job is being processed at the serial processor.
Y_t	Number of jobs inside the serial processor buffer at time instance t
V_t	number of jobs already at the batch processor buffer waiting for additional job arrivals to form a full batch.
B_t	Number of additional job arrivals the prospective job needs to wait for in order to form a full batch
$\lfloor x \rfloor$	largest integer less than or equal to x.
$\lceil x \rceil$	smallest integer greater than or equal to x.
L	Threshold for the batch processor buffer size. May be supplemented by subscripts when the policy has more than one threshold.

T_A Event where the prospective job incurs a penalty due to the need to wait for additional job arrivals, to form a full batch.

T_B Event where the prospective job incurs a penalty due to the need to wait for the batch processor to become available.

$P(\circ)$ Probability that more than $B_t - Y_t$ jobs arrive at the serial processor buffer while the serial processor is processing the prospective job.

Chapter 5 and Chapter 6 Symbols

μ_{si} Mean processing rate of the serial processor M_i. Subscript i is ignored when only one serial processor is present in the system.

μ_B Mean processing rate (in batches) of the downstream batch processor

$b_{ji}(t)$ amount of jobs belonging to Job Family i (i = 1,2) in front of the processor M_j (j < 2) at time instance t, *including* the job currently being processed.

$b_{2i}(t)$ amount of jobs belonging to Job Family i (i = 1,2) stored in front of the batch processor at time instance t, *excluding* the job/s currently being processed.

$S_{sj}(t)$ status of the serial processor M_j at Time Instance t. The subscript j is removed if only one serial processor is present at the system

$S_b(t)$ status of the batch processor at Time Instance t.

μ_f Mean processing rate of the feeder processor, in terms of batches.

$S_f(t)$ status of the feeder processor at Time Instance t.

$N(t)$ number of jobs currently being processed by the feeder processor at Time Instance t.

List of Definitions

Batch processor - processor that can process more than one job at a time.

Serial processor – processor that can process only one job at a time

Batch processor size – maximum job quantity that can be concurrently processed by the batch processor

Wafer fab batch processor model – jobs belonging to different job families cannot be processed together at the batch processor. The processing time is only dependent on the job family being processed, and not on the batch composition or quantity.

Burn-in model – jobs belonging to different job families can be processed together at the batch processor. The batch processing time is the maximum of all the individual processing times of jobs inside the batch.

Static scheduling problem – scheduling problem where all jobs become available for processing at the same time.

Dynamic scheduling problem – scheduling problem where not all jobs become available for processing at the same time.

Makespan (CMAX) – time instance the last job belonging to a predetermined set of jobs finishes processing.

Lateness – difference between the time instance a job finishes processing and its due time.

Tardiness – maximum of the lateness of a job and 0.

Agreeable – two attributes r and d of a set C of jobs are agreeable if and only if: for any two jobs i and j belonging to set C, $r_i \geq r_j$ implies $d_i \geq d_j$

Finite horizon problem – the number of jobs to pass through the system is finite.

Infinite horizon problem – the number of jobs to pass through the system is infinite.

Threshold policy – control policy of the form: do X if and only if a certain attribute is less than the threshold. Otherwise, Do Y. A common threshold policy for batch processors is the **Minimum Batch Size (MBS)** policy: Wait for additional job arrivals when the queue length is less than the minimum batch size. Otherwise, process a batch.

Look – ahead heuristic – Generic term for batch processor heuristics that assume knowledge of a limited number of job arrivals. Look-ahead methods are discussed in Section 2.2.1.1.2.

Polynomial Time Approximation Scheme (PTAS) - given a problem and a value $\epsilon > 0$, a solution that has objective values at most $1 + \epsilon$ times of the optimal solution's objective value can be generated in polynomial time. The polynomial is dependent on ϵ.

Offline algorithm – Algorithms that can be executed once. Output dictates at which time instances each job is to be processed, with certainty.

Online algorithm – Algorithms that are executed regularly during production, and only a limited number of decisions are made at each instance.

Model Predictive Control – Type of online heuristic that typically reduces the problem size for computational tractability. MPC is discussed in greater detail in Section 3.2.

NP-Hard – Practically speaking, a problem is NP-Hard if the amount of time needed to solve the problem increases at an exponential rate. For the formal definition of NP-hardness, refer to [Papadimitriou and Steiglitz 1998].

Coefficient of Variation (CV) – ratio between the standard deviation and the mean of a random variable.

Cycle time – amount of time spent by a job inside a system.

Processing time window – The maximum time allowed for jobs to spend at the batch processor buffer.

Traffic Intensity (*TI)* - the mean job arrival rate (in jobs per unit time) divided by the product of the processor's average processing rate (in service completions per unit time) and the size of the processor.

Chapter 1 Introduction

1.1 Introduction

The semiconductor manufacturing industry has quickly become one of the main drivers of the global economy, with companies also projecting significant growth in the future. Worldwide semiconductor revenues were estimated at US$228 billion for 2005, and is forecasted to rise to US$309 billion for 2008, according to [Chamness 2006]. Figure 1.1 shows the estimated and projected annual revenues generated by semiconductor manufacturing.

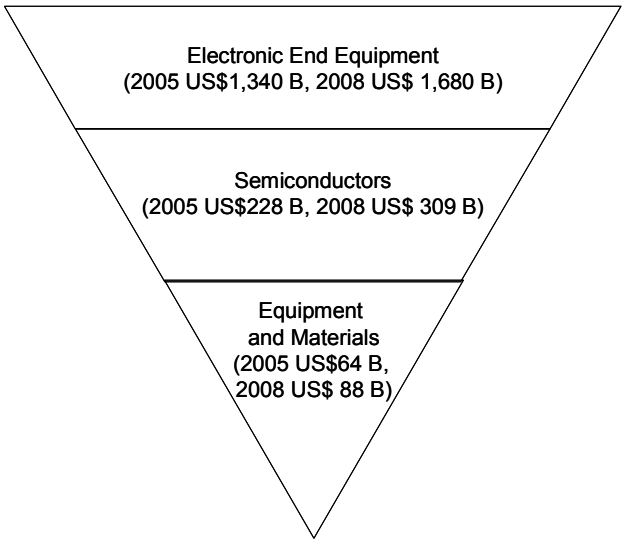


Figure 1.1: Estimated 2005 revenues and forecasted 2008 revenues (B refers to billions) for various stages of semiconductor manufacturing process.

We are primarily interested in the production of semiconductors, which added an estimated US$ 164 B in value for 2005.

Capacity expansion inside a semiconductor manufacturing plant is expensive and time-consuming. Consequently, it is typical for companies to operate at high levels of capacity utilization. VLSI research reported that the capacity utilization for semiconductor wafer fabrication facilities (also known as wafer fabs) is 92.4% for the first quarter and 93.3% for the second quarter of 2006, and is expected to rise to 95% for the rest of 2006 ([LaPedus 2006]). At high capacity utilization, proper production control is needed to meet production targets.

According to [Mason *et al.* 2002], competitiveness in the semiconductor manufacturing industry hinges on the following factors:

- The ability to rapidly incorporate advanced technologies in electronic products
- The ability to improve manufacturing processes (in terms of yield / speed / reliability)
- The ability to meet customer due dates

Although the inverse correlation between yield and cycle time ([Wein 1992]) means that production control also plays a part in improving yield, it is the third factor that controlling the production has the largest influence on. Consequently, considerable effort has been exerted in developing production control methods, as the ability to meet customer demands become a semiconductor company's most important asset when the technology, and hence the price, have inevitably settled down to a stable level.

Batch processors are extensively used in certain portions of the semiconductor manufacturing process. **Batch processors** are processors that can process more than one job at a time. We define the **size** of the batch processor to be the maximum number of jobs that can be processed concurrently. Batch processor processing times are frequently independent of the number of jobs processed. The large batch processor size is oftentimes required to offset the long processing times at the batch processor. Thus, the problem of controlling systems involving batch processors is an interesting and potentially rewarding research problem.

1.2 Background of Research Problem

The entire semiconductor manufacturing process can be divided into four distinct stages: wafer fabrication, wafer probe or test, Integrated Circuit (IC) assembly and IC test (see, for example: [Sivakumar 1999]). The first two stages are commonly referred to as the semiconductor front end, with the last two referred as the semiconductor back end. Figure 1.2 shows the estimated cycle time per stage, according to [Sivakumar].

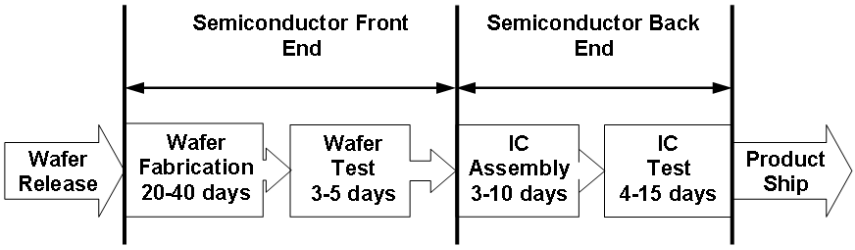


Figure 1.2: Estimated time line for the semiconductor manufacturing process.

The semiconductor front end converts raw silicon wafers into dies, and can be divided into wafer fabrication and wafer test. The semiconductor back end transforms the dies into IC chips, ready to be connected to electronic devices. Jobs typically spend the longest time in wafer fabrication and IC test, two stages which feature batch processors.

1.2.1 Primer on semiconductor manufacturing process

The four stages of semiconductor manufacturing are briefly described, with representative processes involving batch processors highlighted.

1.2.1.1 Wafer Fabrication

Wafer fabrication is deemed the most complicated of the four stages of semiconductor manufacturing, in terms of production control. Wafer fabrication converts a thin wafer of raw silicon into numerous dies, each composed of millions of tiny transistors. The interested reader is referred to semiconductor manufacturing textbooks ([Madou 2002], for example) for further information on wafer fabrication processes.

Under the wafer fabrication stage, several operations are performed on batch processors. The most-often mentioned operations are the oxidation and diffusion operations, although cleaning,

etching, ion implantation and electrical beam writing may also be performed via batch processors [Mathirajan and Sivakumar 2006].

Oxidation – Oxidation refers to the process of growing a layer of silicon dioxide over the silicon wafer. The two most important properties for the layer are its thickness and its dielectric properties. Thus, the operation is normally preceded by a cleaning sequence designed to remove all contamination. The silicon dioxide layer is then grown by placing the wafer inside a furnace with an atmosphere of either high purity oxygen or water vapor at temperatures in the range of 600 to 1250 degrees Celsius ([Madou 2002]). The thickness grows at a rate between 3.3 – 4.5 nm/min ([Huff]) .

Diffusion - Diffusion is the process of ensuring a uniform distribution of dopants into the silicon. Doping is performed to change the electrical characteristics of silicon. Diffusion actually consists of two major steps, pre-deposition and the actual diffusion.

Pre-deposition is the process of introducing a small amount of dopant into the surface of the silicon. Before this can be done, the surface of the silicon must be thoroughly cleaned to prevent any unwanted material from being added with the dopant. After cleaning, the silicon wafers are placed into a pre-deposition furnace, where the dopant (in the form of powder, liquid or gas) is flown through the wafers via nitrogen gas. The higher the temperature, the faster the dopant penetrates the silicon wafer surface. The typical temperature used is between 950 to 1280 degrees Celsius ([Madou 2002]).

The diffusion process consists of growing a thin layer of silicon dioxide over the silicon. This process helps the dopant to diffuse into the silicon. To grow the layer of silicon dioxide, the wafer is placed in a furnace, and high purity oxygen is blown in the furnace at high temperatures. The combined pre-diffusion and diffusion operation can take ten hours or more, as compared to one or two hours for other operations, according to [Uzsoy 1995].

Even though oxidation and diffusion induce two different transformations on the material, the essential problem for both operations are similar: jobs arrive in front of a processing stage at future time instances. Each job belongs to a job family, and only jobs from the same job family can be batched together. The stage contains at least one batch processor.

1.2.1.2 Wafer Test

Wafer test is mostly concerned with determining the functionality of each die, by using electrical probes to run a series of tests. Although the control of wafer test operations is also a difficult problem that has attracted the attention of several researchers, this stage does not prominently involve batch processors, and is not further discussed.

1.2.1.3 Integrated Chip (IC) Assembly

IC assembly extracts the good dies from the wafers, and bonds these dies to a suitable substrate, along with optional components such as heat sinks. An epoxy cure process performed on a batch processor may be required. The die and the corresponding substrate are held together with epoxy, which requires curing at elevated temperatures for several hours to hold the die and the substrate together with sufficient strength.

1.2.1.4 IC Test

IC Test has two primary operations of interest: Burn-in and functional test. Burn-in ovens are batch processors which expose ICs to elevated thermal and electrical stress; this operation simulates the expected cumulative stress an IC will experience over a prescribed period of usage. ICs that suffer from "infant mortality" are then readily identified during the succeeding functional test operation. The duration of the burn-in operation is dependent on the job family, and can range from a few hours to two whole days. Because there are no chemical recipes involved, it is possible to mix several job families in one oven; for quality considerations the longest burn-in time among all jobs inside the oven is the burn-in time for the entire batch.

Due to the prohibitive cost of the burn-in boards used by the ICs while undergoing burn-in, the queue time at the burn-in operation is strictly monitored, and any reduction can lead to significant cost savings in the form of reduced burn-in board requirements. The burn-in operation also directly feeds the functional test operation, the bottleneck of the semiconductor back end. For these reasons, the control of the burn-in ovens is of utmost importance.

1.2.2 Importance of batch processors

The main performance measures of a manufacturing system are typically throughput (the output rate of the system) and cycle time (the amount of time elapsed from the job entering the system until the job exits the system). If yield is perfect, these two performance measures are related by Little's Law. If L is the amount of work in progress (WIP) inside the system, W is the cycle time and λ is the throughput, then Little's Law states that $L = \lambda W$ ([Hopp and Spearman 2000], for example).

Batch processors can significantly impact both performance measures. Assume we are controlling a diffusion oven with size of Q jobs. If we never wait for additional job arrivals before processing a batch, then it is possible that the machine will be processing very few jobs at a time. Consistently processing batches below the oven size can cause the effective long-term oven capacity to be significantly lower than forecasted. On the other hand, if we insist on processing only full batches, then jobs typically will be forced to wait for additional job arrivals. When job arrivals are rare then jobs may have to wait for long periods of time at the oven queue.

Effective production control of the batch processor seeks to answer the conundrum facing the production controller each instance a partial batch is available: Process a batch now, or process a batch later. This is true for both the wafer fab and burn-in batch processor models. Unfortunately, the inherent difficulty in controlling the diffusion furnace (in wafer fab) and the burn-in oven (in IC test) stem from different characteristics of the problems. In wafer fabrication, only jobs from the same job family can be processed together, and the processing time of the batch is dependent only on the job family being processed. In IC test, there is typically no limitation in the composition of the batch, but the batch processing time is dependent on the composition of the batch. Although some researchers have modeled batch processors that exhibit both characteristics (for example, [Boudhar 2003, Boudhar 2003] and [Yuan *et al.* 2004]), a large majority of the research conducted regarding batch processor control can be divided into these two categories. This fundamental difference in the

characteristics of the problems makes it attractive to segregate the problem of controlling batch processors inside the wafer fab from the problem of controlling burn-in ovens.

In this dissertation, we concentrate on batch processor with incompatible job families, for the following reasons:

- The semiconductor front-end manufacturing system is more complex than the back-end. This complexity generally results in longer idle time for jobs, relative to the processing times. This makes it relatively easier for an improved control policy to generate significant savings.
- Batch processors are more frequently used in the front-end, compared to the back-end. In wafer fabrication, the reentrant nature of the process flow forces jobs to enter the ovens several times. In contrast, chips undergo burn-in only once.

1.3 Research Motivation

The problem of controlling the production of batch processors has significant implications to the performance of wafer fabs, for several reasons. These include:

- Operations performed on batch processors typically have long processing times, compared to other operations.
- The effective long-term capacity of batch processors is dependent on the control policies used.
- The work-in-progress (WIP) and output of batch processors are difficult to approximate with smooth production rates. batch processors induce sudden WIP decreases and output increases, since they are bulk servers. This can cause system-wide performance degradation if not properly controlled.

The control of batch processors has often been considered an isolated problem from the problem of controlling the production of wafer fabs. Recently, look-ahead methods that use

information from a limited number of upstream and downstream operations in controlling the batch processor have become popular ([Robinson *et al.* 1995]). Unfortunately, incremental gains have not kept up with the increasing complexity of control policies based on constant refining of the early look-ahead methods. This has led to the hypothesis that a fundamental change in approaching the problem of controlling systems involving batch processors is needed, to obtain significant improvements in the performance of the wafer fab. Although work has been done in controlling multi-stage systems involving batch processors, majority of the models do not consider the presence of incompatible job families, a characteristic that is very relevant to complex systems such as semiconductor wafer fabs. It is the bridging of this gap in scheduling and control literature that we propose to initiate, with this dissertation.

While look-ahead methods are generally an improvement over policies that assume no future information, look-ahead methods have a significant drawback: the batch processor cannot influence the arrival patterns of its WIP. We believe that a hypothetical control policy that proactively drives WIP into the batch processor such that a good batch is always loaded into the batch processor can result in better performance than a look-ahead policy. This hypothetical policy is based on the observation of [Mason and Fowler 2000] that good batch processor schedules tend to produce good overall schedules (in minimizing total weighted tardiness), and is similar that of [Bhatnagar *et al.* 1999] and [Chandra and Gupta 1997], where lot release is in accordance to an idealized batch processor schedule. The work presented differs from what was previously proposed in that we focus in the local control of the batch processor and its nearby operations.

1.4 Research Objective

The main objective of this research is to investigate how the concept of controlling a batch processor's upstream processors in accordance to the anticipated needs of the batch processor can be of benefit to the system, particularly in terms of reducing the mean amount of time jobs spend inside the system.

To achieve this objective, we also classify the subordinate objectives under three sections:

- Analysis of the problem of optimally controlling the batch processor in isolation.
    - We analyze the complexity of optimally solving the deterministic, finite-horizon version of the problem to determine which facets of the problem cause great difficulty in minimizing the amount of time jobs spend in the queue.
    - We determine the consequences of constraining the upstream processor to the anticipated needs of the batch processor on the mean time spent by jobs at the batch processor buffer by changing the arrival pattern observed by the batch processor and observing its effect on different control policies.
- Analysis of the problem of controlling upstream processors, given queuing time limits for jobs in front of the batch processor.
    - We determine how the upstream processor can be controlled when jobs cannot stay too long at the batch processor buffer without incurring penalties.
    - We look at how the optimal upstream processor policy changes as the batch processor control policy changes.
    - We evaluate the effect of the reliability of the batch processor on the optimal upstream processor policy.
- Analysis of multi-stage models with a downstream batch processor and incompatible job families
    - We characterize the performance of a two-stage system with an upstream serial processor and a downstream batch processor where jobs belong to different job families and both processors are operated under different simple control policies.
    - We determine the effect of constraining the selection of the job family the upstream serial processor processes according to the anticipated needs of the downstream batch processor, as compared against using a locally optimal serial processor policy.

  - We determine whether the conclusions derived when the upstream processor is a serial processor also hold true when the upstream processor is a batch processor with smaller size.
- We extend analysis to three-stage systems to determine how the system reacts to constraining additional processors further upstream of the batch processor to the anticipated needs of the batch processor.

1.5 Organization of Chapters

In Chapter 2, we classify the existing literature on the control of batch processors in semiconductor manufacturing. In doing so, we identify gaps in current literature, and determine where the work documented in this dissertation falls under. In Chapter 3, the control of a single batch processor with incompatible job families and future job arrivals is analyzed, and a heuristic with reasonable running time has its performance evaluated against a previously proposed look-ahead heuristic. In Chapter 4, the problem of controlling the processors upstream of the batch processor is analyzed, given the possibility that the upstream processors also feed other processors. Chapters 5 and 6 look into the behavior of simple manufacturing systems with a downstream batch processor and incompatible job families, under simple control policies. Chapter 5 assumes that the upstream processors are serial processors, while Chapter 6 looks into the possibility that the upstream processor is another batch processor with smaller size. Chapter 7 discusses the insights obtained from this study, and summarizes the results obtained from this research, before pointing out several avenues for future research.

Chapter 2 Literature Review

A semiconductor wafer fab can be characterized by the presence of batch processors, reentrant flows, high product variety, sequence-dependent set-ups and both random and scheduled equipment down times. Due to its degree of difficulty, the problem of controlling semiconductor wafer fabs has received much attention from researchers. For a review of control methods for more general problems, the reader is referred to [Gupta and Sivakumar 2006], [Jones and Rabelo 1998], [Lee *et al.* 1997], and [Vaessens *et al.* 1996], among others.

In this section, we discuss methods for controlling batch processors, particularly batch processors found in wafer fabrication. In doing so, we also touch upon examples of how current literature considers the presence of batch processors in a larger system. When appropriate, we highlight the gap that exists between general semiconductor manufacturing control methods and batch processor control methods.

2.1 Introduction to Batch Processors in Semiconductor Manufacturing

Batch processors are used in several industries, including shoe manufacturing, furniture manufacturing, metal works industry (examples include [Ram and Patel 1998] and [Schwindt and Trautmann 2003]) and printed circuit board assembly ([Bhatnagar *et al.* 1999]). The practice of simultaneously processing several jobs has also been observed in the service industry ([Simons and Russell 2002]) and in multimedia traffic management ([Fonseca and

Facanha 2002]). We focus on batch processors found in semiconductor manufacturing, for which [Mathirajan and Sivakumar 2006] provide a review of.

There are generally two distinct batch processor models found in semiconductor manufacturing. Diffusion and oxidation ovens in the wafer fab belong to the wafer fab batch processor model. The **wafer fab batch processor model** assumes that jobs belonging to different job families cannot be processed together. The processing time is only dependent on the job family being processed. Burn-in ovens found in assembly/test, on the other hand, operate in a different manner. The **burn-in model** assumes that jobs belonging to different job families can be batched together. However, the processing time of the batch is the maximum of the minimum processing times of the jobs that comprise the batch.

Another batch processor model found in semiconductor manufacturing is exemplified by the E-beam writer, where the processing time of a batch is the sum of the setup time and the individual processing times of each job ([Hung 1998]). Literature covering this model of the batch processor is relatively rare, and is not further discussed.

We discuss the literature considering batch processors with incompatible job families in greater detail. There will be instances when a model assumes only a single job family, and these are considered under the wafer fab model. There also exist instances where the batch processor both has compatible and incompatible families: we do not discuss these papers. Figure 2.1 shows a simplified representation of the models used in discussing batch processor control in semiconductor manufacturing.

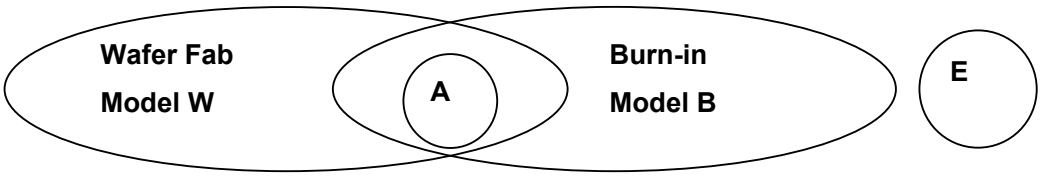


Figure 2.1: Proposed division of current literature on batch processor behavior and control.

The two main divisions are: wafer fab model (Set W) and burn-in model (Set B). Set E considers another batch processor model found in semiconductor manufacturing, typified by the E-beam writer. There are overlaps between the wafer fab model and the burn-in model. Set A is for the case where all jobs belong to one job family. We discuss set A under the wafer fab model.

Figure 2.2 shows how the discussion on the literature on batch processors in semiconductor manufacturing is divided.

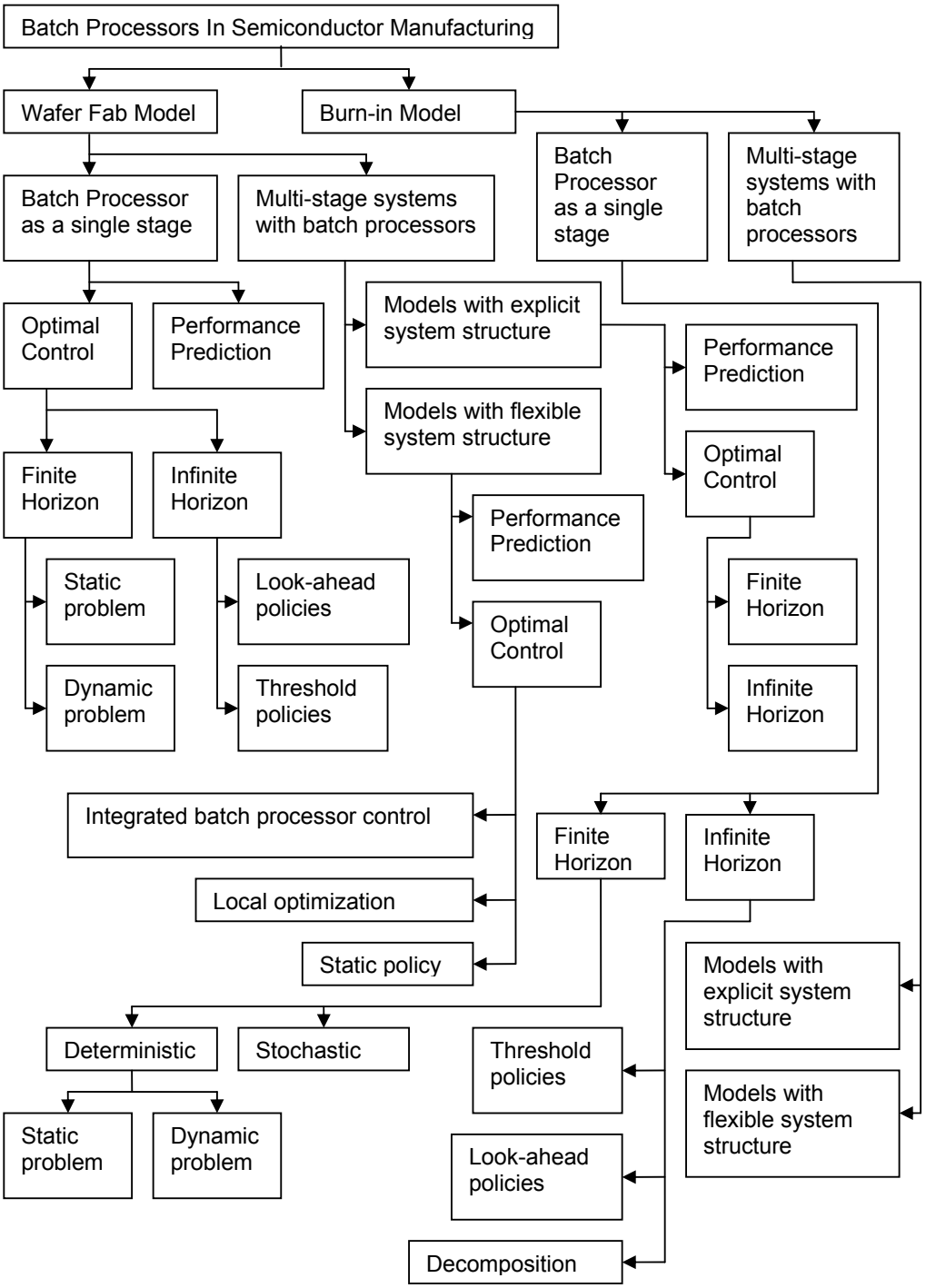


Figure 2.2: High-level division of the discussion of previous work on batch processors in semiconductor manufacturing.

An explicit system structure is present when a specific system configuration (number of stages, as well as the number and type of processors per stage) is assumed. A flexible system structure, on the other hand, allows for a variable number of stages. A static problem assumes that all jobs are queued at time instance zero, while dynamic problems assume at least some jobs will become ready only at a future time instance.

2.2 Batch Processors with Incompatible Job Families (Wafer Fab Model)

We divide the discussion of the behavior and control of batch processors inside the wafer fab into research that considers the batch processing stage in isolation and batch processors that are parts of a larger manufacturing system.

2.2.1 Batch processor as a single stage

The behavior and control of a stage containing one or more batch processors in the wafer fab is a family of problems that has been significantly discussed in past literature. Figure 2.3 illustrates the classification scheme adopted in this survey for batch processors in a single stage. **Optimal control** refers to the general goal of controlling a stage such that a performance measure is optimized. **Estimation of performance parameters** or **performance prediction** is mainly used to determine gross behavior of the system, and is typically used for design and production planning.

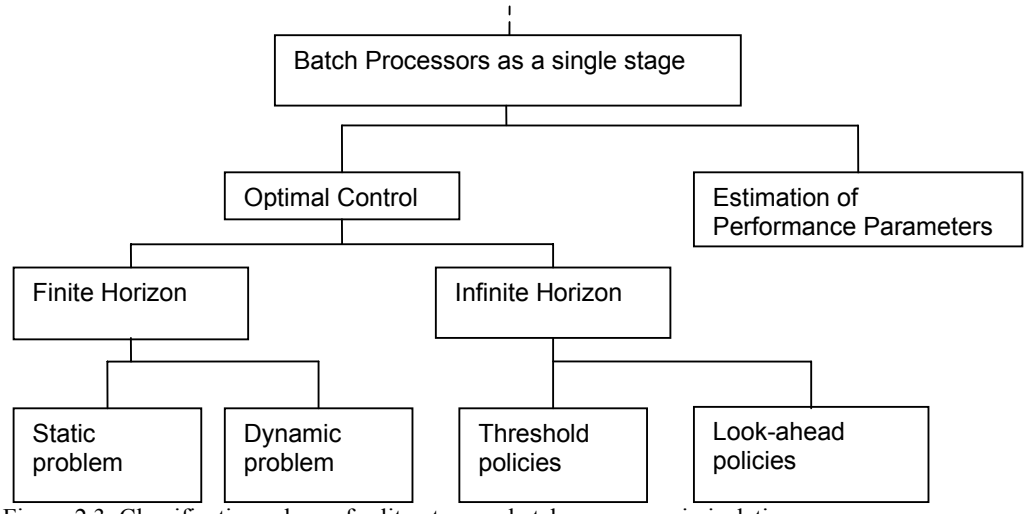


Figure 2.3: Classification scheme for literature on batch processors in isolation.

This figure is a subsection of Figure 2.1 enlarged for emphasis. A static problem is a scheduling problem where all jobs to be scheduled are initially waiting in front of the stage. A dynamic problem is a scheduling problem where at least some jobs will not be available in front of the stage at the start of the system.

2.2.1.1 Optimal control of a single stage of batch processors

We divide the discussion into finite horizon problems and infinite horizon problems. Finite horizon problems assume that the number of jobs to be processed is finite, while infinite horizon problems assume that there is an unending stream of jobs to be processed. This implies some degree of uncertainty for infinite horizon problems. In contrast, finite horizon problems typically assume data is deterministic. This differentiation does not imply that solutions to finite horizon problems are irrelevant to infinite horizon problems: finite horizon algorithms can be used for infinite horizon problems by repeatedly running the algorithm at specific intervals.

2.2.1.1.1 Finite Horizon Problems

Finite horizon problems are typically scheduling problems. Scheduling problems can be broadly classified into two categories: static and dynamic. A **static** scheduling problem assumes all jobs are already in front of the batch processor at the start of the schedule. A **dynamic** scheduling problem occurs when at least some jobs will become available only after a period of time has elapsed since the start of the schedule.

- ***Static scheduling problem***

[Uzsoy 1995] considers the problem of scheduling *n* jobs, belonging to *m* job families on a stage containing one or more batch processors. When there is only one batch processor, optimal algorithms are provided to minimize **makespan** (the maximum completion time), the sum of the weighted completion times, and the maximum lateness. Of particular interest is the optimal algorithm for minimizing the sum of weighted completion times: jobs for each family are indexed according to their weights, and assigned to batches such that only the last batch may be partially full. The weight of the batch is the sum of the individual job weights, and batches are processed in decreasing order of the ratio of weight to processing time. When there are multiple batch processors in parallel, the author suggests heuristics based on serial processor scheduling literature; these heuristics are shown to have a guaranteed performance bound on minimizing the same three objectives. [Uzsoy 1995] also considers the dynamic

scheduling problem of optimizing makespan and maximum lateness. An efficient optimal algorithm is proposed to optimize makespan, and three heuristics are evaluated against each other in optimizing the maximum lateness.

[Dobson and Nambimadom 2001] look into the problem of scheduling a batch processor with incompatible job families. Unlike most problems involving wafer-fab models, jobs are assumed to have different job sizes and job weights, and the objective is to minimize the weighted completion time. This model for the batch processor is also discussed by [Azizoglu and Webster 2001], [Koh *et al.* 2004] and [Koh *et al.* 2005].

[Mehta and Uzsoy 1998] propose a dynamic programming (DP) formulation (running in polynomial time if the batch processor size and number of job families is fixed) to minimize the total tardiness incurred in scheduling n jobs belonging to m job families on a batch processor. The authors also propose a decomposition-based heuristic for larger problem instances. [Jolai 2005] show similar results to that of [Mehta and Uzsoy 1998], when the performance measure is changed to the number of tardy jobs.

[Perez *et al.* 2005] discuss the problem of scheduling a single batch processor to minimize the total weighted tardiness. The NP-hard problem is split in two decisions: segregating jobs into batches, and sequencing batches. The efficacy of Earliest Due Date (EDD) and Apparent Tardiness Cost (ATC) heuristic in segregating jobs into batches are evaluated. ATC considers the processing times, weights, slack time and involves a decay parameter used to vary the relative importance of the slack time. For sequencing batches, an exact DP algorithm and three different heuristics are suggested: EDD, ATC and a modification of the decomposition heuristic (DH) suggested by [Mehta and Uzsoy 1998]. Simulation experiments suggest that ATC outperforms EDD when it comes to distributing jobs to batches. Furthermore, although ATC-DP outperforms all the other hybrid heuristics evaluated, the performance of ATC-DH does not vary from ATC-DP by much, with significantly shorter computational time. [Balasubramanian *et al.* 2004] extend the problem discussed by [Perez *et al.* 2005] by assuming the stage contains several batch processors in parallel, and propose using GA for either assigning jobs to batches or sequencing batches.

One possible interpretation of batch processors is as vehicles, as most methods of transportation can concurrently carry several items. Given this interpretation, earliness/tardiness measures may be of interest. [Xiangtong and Fengsheng 1999] present two different DP algorithms for the problem of scheduling n jobs (belonging to a single family) on a batch processor to minimize the earliness/tardiness criterion.

- ***Dynamic scheduling problem***

One of the earliest control problems involving batch processors was [Ikura and Gimple 1986], which consider the problem of scheduling n jobs (belonging to one family) on a single batch processor to reduce the makespan. Assigning jobs to batches such that only the first batch may be partially full will minimize makespan for the static problem. When each job i has a release time r_i and a due time d_i, and the release and due times are agreeable (**agreeable** release times and due times mean that, for any two jobs i and j, $r_i \geq r_j$ implies $d_i \geq d_j$), the authors propose an algorithm that finds the feasible schedule with the minimum makespan, if it exists.

[Webster and Baker 1995] consider the problem of minimizing the maximum lateness of n jobs (belonging to only one job family) on a batch processor. When the due times and release times are agreeable, the authors provide an optimal DP algorithm that runs in $O(n^3)$ time. [Bar-Noy *et al.* 2002] provide an approximation algorithm with a known worst-case performance bound for the NP-hard problem of scheduling n jobs (belonging to m job families) to minimize the total weighted number of tardy jobs on a stage containing c batch processors. The special cases of (1) a fixed number of job families and (2) fixed number of distinct due dates and processing times are solvable in polynomial time

[Zhang *et al.* 2003] considers the problem of controlling a stage of m parallel batch processors to minimize the makespan of n jobs. Unlike other problem formulations, the arrival times are unknown. If a partial batch is available, the authors suggest idling for an amount of time dependent on the latest job arrival time before processing a batch. The competitive ratio of this online algorithm (compared against the optimal value of an algorithm that has access to the arrival times) is also obtained.

[Monch *et al.* 2006] elaborate on the problem initially proposed by [Balasubramanian *et al.* 2004] by assuming that the jobs arrive at future time instances. The authors note that variants of the ATC heuristic generate good performance within a short computational time interval. Unfortunately, the performance of ATC is sensitive to the value of the decay parameter. To determine the optimal parameter value, the authors evaluate two different machine learning techniques: neural network and inductive decision trees, and determine that both methods generate parameters that exhibit good performance, although training of the inductive decision tree takes shorter time than that of the neural network.

The problem of minimizing the total completion time to process n jobs belonging to m job families at a batch processor, when the jobs arrive at future time instances has been assumed to be NP-Hard, but we have not come across any literature that proves this assumption. We show that this problem is NP-Hard in Chapter 3, and develop a dynamic program to solve this problem. The empirical complexity of the DP algorithm is also investigated.

2.2.1.1.2 Infinite Horizon Problems

Infinite horizon problems are typically suitable for modeling systems which operate on a continuous basis, where jobs arrive in front of the batch processor at certain time instances. Thus, we can only characterize the arrival distribution.

There are typically two types of solutions proposed for infinite horizon problems: threshold policies and look-ahead policies. **Threshold policies** are typically of the form: process a batch only if there are at least X number of jobs waiting in front of the batch processor. This is also called a minimum batch size (MBS) policy, with MBS = X. Unfortunately, threshold policies rarely consider the possibility of multiple job families. **Look-ahead policies** typically assume that the future arrival time of a certain number of jobs can be predicted.

- ***Threshold policy***

[Deb and Serfozo 1973] is possibly the first research performed on the problem of optimally controlling a batch processor. The authors develop dynamic programming formulations for minimizing the average cost per unit time and the expected discounted cost of a batch

processor with stochastic processing times and Poisson job arrivals. A cost that is linear with the number of jobs is incurred when a batch is processed and the holding cost per unit time per job is a non-negative function. The optimal policy is proven to be a minimum batch size policy. When the holding cost is linear and the processing time distribution is exponential, closed-form equations are developed to determine the optimal MBS in minimizing the expected discount cost, as well as in minimizing the expected average cost per unit time. [Aalto 1998] further develop the model of [Deb and Serfozo 1973] by assuming that job arrivals come from a compound Poisson process. Follow-up work ([Aalto 2000]) consider the special case where the holding costs are linear.

[Avramidis *et al.* 1998] consider a more abstract problem than that of [Deb and Serfozo 1973], and assume general processing time and arrival rate distributions. The authors determine the optimal threshold, among all threshold policies. (In this case, there exists no guarantee that there is an optimal policy in the form of a threshold. The optimal threshold policy is simply the best policy among all threshold policies.)

Majority of batch processor literature ignore the possibility of a limit in the maximum queue time of jobs at the batch processor buffer. This situation exists in wafer fabs, where jobs that are queued in front of a batch processor beyond a certain amount of time are deemed contaminated. [Makis 1985] considers a batch processor with Poisson arrival rate and general service time distribution. The batch processor has infinite size and the service cost function is affine with respect to the number of jobs processed, while the holding cost function is nonnegative with respect to the number of jobs. Jobs have a maximum queue time in front of the batch processor. [Makis 1985] shows that the optimal policy in minimizing average cost is to process all jobs once a threshold queue has been reached, and to process all jobs even if the threshold is unmet if a job is about to reach its queue time limit.

One disadvantage of using a threshold policy is that the arrival rates have to be approximated; large deviations between the approximated and the actual arrival rate can cause large-scale degradation in the system performance. [Sung and Choung 1999] looks at a single-family, single batch processor model, and uses a neural network to determine the batching decisions at

the batch processor in order to minimize mean queue time. A direct comparison between various MBS policies and the proposed neural network-based approach when the arrival rate and distribution varies illustrates the robustness of the neural-network approach in the face of varying arrival rates and distributions.

All threshold policies mentioned have only dealt with jobs belonging to a single job family. No guidelines exist as to how batches between job families are to be selected. [Xia *et al.* 2002] discuss the performance of a single batch processor with exponential service time distributions and different job families, each of which has an identical Poisson arrival process. If the mean service time is identical for all job families and the buffer capacities are either infinite or equal in size for all job families, then the throughput rate of the system is stochastically maximized if we choose to process the job family with the longest queue.

- ***Look-ahead policies***

Look-ahead policies assume that a certain amount of future job arrivals can be predicted, and use this information to determine the appropriate action at the batch processor. We discuss look-ahead methods in greater detail, as we will later use a look-ahead method as a baseline policy to compare against in Chapter 3.

[Glassey and Weng 1991] propose what is possibly the first look-ahead policy, called Dynamic Batching Heuristic (DBH), for controlling batch processors in the semiconductor wafer fab. DBH assumes that all jobs belong to one job family, with a processing time P. At a time instance t that the batch processor is available and only a partial batch is available, DBH is activated. DBH assumes a planning horizon from t to $t + P$. The batch processor can be processed within this time horizon, and DBH computes the optimal time to process a batch by:

- predicting the next X job arrivals. X is a parameter of DBH, but cannot be more than the number of jobs needed to form a full batch. Any jobs arriving outside of the planning horizon are also ignored. Thus, the total number of future job arrivals considered Y may be less than the parameter X.

- calculating the total waiting time incurred by all jobs (including the *Y* future job arrivals) if a batch processor is started at time instance *i,* for each of the *Y* + *1* possible time instances *i* (the current time instance + the *Y* time instances a job arrival occurs).

The time instance *i* with the minimum waiting time is selected as the time instance the batch processor is processed.

Simulation experiments show that DBH outperforms the best MBS policies, even with moderate errors in the predicted arrival time, as long as the traffic intensity is not low.

[Fowler *et al.* 1992] suggests a next arrival control heuristic (NACH) that only considers the first future job arrival. Furthermore, NACH does not specify a future time instance when the batch processor is to be loaded. Rather, NACH decides to either process a batch immediately, or wait for the next job arrival. When there are moderate prediction errors, NACH tends to slightly outperform DBH.

[Fowler *et al.* 1992] also extend NACH to the case where jobs have different job families. (We refer to this extension as NACHM.) Future knowledge of the first job arrival for each job family is assumed to be known. When a job arrival corresponding to family *j* occurs, NACH is used to evaluate whether a batch of job family *j* is to be produced ("push logic"). When the batch processor finishes processing, a different algorithm ("pull logic") is used.

- If there exists at least one full batch, then the full batch with the Weighted Shortest Processing Time (WSPT) is chosen for processing. The weight of a particular job family *i* is the total number of jobs not belonging to job family *i* currently in front of the batch processor.
- If a full batch does not exist, then for all job families *i*, NACH is performed. The job families for which NACH decides to process is noted.
- If NACH suggests processing for all job families, then WSPT is used to select which job family to process. Conversely, if, for all job families, NACH suggests waiting for the next job arrival, then the decision is to wait for the next job arrival.

- If the NACH decisions are not unanimous for all job families, then NACHM follows the decision that minimizes the total waiting time incurred if the recommended decision of NACH is followed for all job families.

Comparison of NACHM with MBS shows that NACHM outperforms the best set of MBS policies, even under moderate prediction errors.

Other look-ahead policy variants have considered minimizing the cost per unit time (MCR heuristic of [Weng and Leachman 1993] and RHCR heuristic of [Robinson *et al.* 1995]), minimizing the cost per job processed (DJAH of [Van Der Zee *et al.* 2001]), considering the downstream processor (RHCR-S heuristic of [Robinson *et al.* 1995], NACHM-Setup heuristic of [Solomon *et al.* 2002], and DJAH-F of [Van Der Zee *et al.* 2001]) , and considering the possibility of having multiple batch processors in parallel (DSH of [Van Der Zee 2001] and NACH-MM of [Fowler *et al.* 2000]). We discuss NACH-MM in greater detail as we benchmark the proposed heuristic in Appendix A against NACH-MM.

NACH-MM, proposed in [Fowler *et al.* 2000], is based on NACHM (proposed in [Fowler *et al.* 1992]). Aside from the difference in computing the time instance the next batch processor will become idle, the only modifications with the original NACHM push logic (proposed in [Fowler *et al.* 1992]) are that a batch of the particular job family is immediately processed under one of the following conditions:

- If the number of idle batch processors are greater than one, or
- If the next job arrival for that particular job family occurs only after a batch processor finishes processing, or
- If a batch for that particular job family can be finished before any batch processor finishes processing.

The pull logic of NACH-MM differs in that a batch is guaranteed to be processed under one of the following conditions:

- There are multiple batch processors idle

- The next job arrival (belonging to any job family) will occur only after a batch processor finishes processing.

Simulation experiments suggest that, although the performance of NACH-MM generally improves upon the performance of MBS policies, the performance of NACH-MM deteriorates compared to the best MBS policy when the number of batch processors are large relative to the number of job families. This effect is magnified when the arrival rate is high, as immediately processing a batch when more than one batch processor is idle causes a significant number of batches with small number of jobs to be processed.

[Duenyas and Neale 1997] consider a single batch processor with incompatible job families; the processing time distribution of each job family is exponential. The batch processor size and the holding cost per unit time per job in queue may be different for each job family. The objective is to minimize the average holding cost per unit time. When there are only two job families, each of which has a Poisson arrival process and idling of the processor is not allowed, processing the job with the longest queue is optimal if the sizes, arrival rates and unit holding costs are identical for both job families. If idling is allowed, the optimal policy can have a complicated form and a heuristic policy is proposed. If a full batch is available, all the other job families with lower mean processing times are evaluated to determine the minimum batch size to preserve stability. The family with the weighted shortest processing time is loaded (weights are equal to the number of jobs inside the batch). If no full batches are available, then the family with the weighted shortest processing time is selected. The estimated benefit of waiting for an additional job from the particular job family is determined, and a batch is loaded. This heuristic can be easily adapted to reflect knowledge of future arrivals (NADH).

Previous look-ahead methods consider all jobs to have equal importance. [Gupta *et al.* 2004] propose a look-ahead batching policy (LAB) that considers the possibility of hot lots in the system. Any hot lot in front of the batch processor will have to be processed as soon as a processor becomes idle, regardless of batch processor policy. At each decision point at the batch processor, LAB looks ahead up to one processing time interval to determine whether any hot lots will arrive. If a hot lot will arrive, the processor is reserved for the hot lot. If not, then

another heuristic is used to determine whether a batch is to be processed or not. LAB has also been adapted for minimizing Earliness/Tardiness measures ([Gupta *et al.* 2004]), where two different E/T measures ($E^2 + T^2$, and $|E| + |T|$) are used. [Gupta and Sivakumar 2006] modify LAB to consider optimizing several objectives concurrently. The aim is to simultaneously minimize the maximum tardiness, the number of late jobs and the average tardiness. LAB uses simulation to evaluate which batching scenario among all possible batching scenarios within one processing time interval is optimal. All three LAB policies assume that jobs belong to a single job family.

Table 2.1 provides a concise summary of the characteristics of the various proposed look-ahead policies, listed in chronological order, that can handle multiple job families.

TABLE 2.1: SUMMARY OF KEY LOOK-AHEAD POLICY ATTRIBUTES

Heuristic Acronym	Postpones decisions to future time interval?	No. of future job arrivals to be considered	Ability to handle parallel processors?	Incorporates downstream processor information?
NACH	Yes	1 per job family	No	No
MCR	No	No. of jobs needed to form a full batch	No	No
RCHR	Yes	No. of jobs needed to form a full batch	No	No
RCHR-S	Yes	No. of jobs needed to form a full batch	No	Yes
NADH	Yes	1 per job family	No	No
NACH-MM	Yes	1 per job family	Yes	No
DJAH	Yes	1 per job family	Yes	No
DSH	Yes	1 per job family	Yes	No
DJAH-F	Yes	1 per job family	Yes	Yes
NACH-Setup	Yes	1 per job family	Yes	Yes

We propose a model predictive control (MPC) -based heuristic for the infinite horizon problem, based on the optimal DP algorithm generated for the finite horizon problem in Chapter 3. This MPC-based heuristic is benchmarked against NACHM. Furthermore, we also evaluate a simple extension of the MPC-based heuristic against NACH-MM when the batch processing stage contains more than one processor. The results are documented in Appendix A.

2.2.1.2 Estimation of performance measures

[Ming-Guang *et al.* 2001] develop an analytic approximation for a station involving c batch processors, each with size x and capable of processing any one of y job families. The batch

processors are assumed to process the job family with the largest queue size, and are under no-idling policies. This model is shown to be a better approximation than the $M/M^x/c$ (only one job family) model proposed by [Ghare 1968], particularly when the number of job families is large, and the arrival rates are moderate. When the arrival rates are moderate, the total queue length is large, but the queue length for each job family is low. Thus, neglecting the job family would overestimate the number of jobs being processed by the batch processor.

[Fowler *et al.* 2002] use a $G/G(^b_p)/c$ model to determine steady state performance of a stage of batch processors with incompatible job families, assuming constant batch size per job family. ($G(^b_p)$ refers to a general distribution that can process up to b_p jobs at a time, where p is the job family.) When the number of job families is large, a complete enumeration of the possible batch sizes to obtain the optimal batch size is not feasible, and a genetic algorithm is used to approximate the optimal batch sizes.

[Jacobs *et al.* 2006] use arrival and departure times of lots at the batch processor to estimate the amount of variability observed at the batch processor. The authors observe that bad batching policies can degrade cycle time measures even if other sources of variability are low. This observation further supports the contention that special attention should be given to considering batch processor policies inside a wafer fab.

2.2.2 Multi-stage models involving a batch processor

Recent years have seen an increase in interest on systems involving a batch processor. While majority of the research involving multi-stage systems are discussed in this section, extensions of look-ahead strategies that consider additional stages were discussed as part of look-ahead strategies (under a subsection of Section 2.2.1.1) for greater continuity. Figure 2.4 shows the classification structure adopted for this section.

Research that deals with the prediction of performance measures is typically used for production design and planning. A large system that uses a fixed batch processor policy essentially assumes a simplistic batch processor policy that is never changed. Using local

optimization of the batch processor means that the control of the batch processor is adapted from one of the approaches suggested in Section 2.2.1, while using an integrated control of the batch processor means that the suggested system-wide control policy explicitly analyzes the presence of batch processors in the system in greater detail.

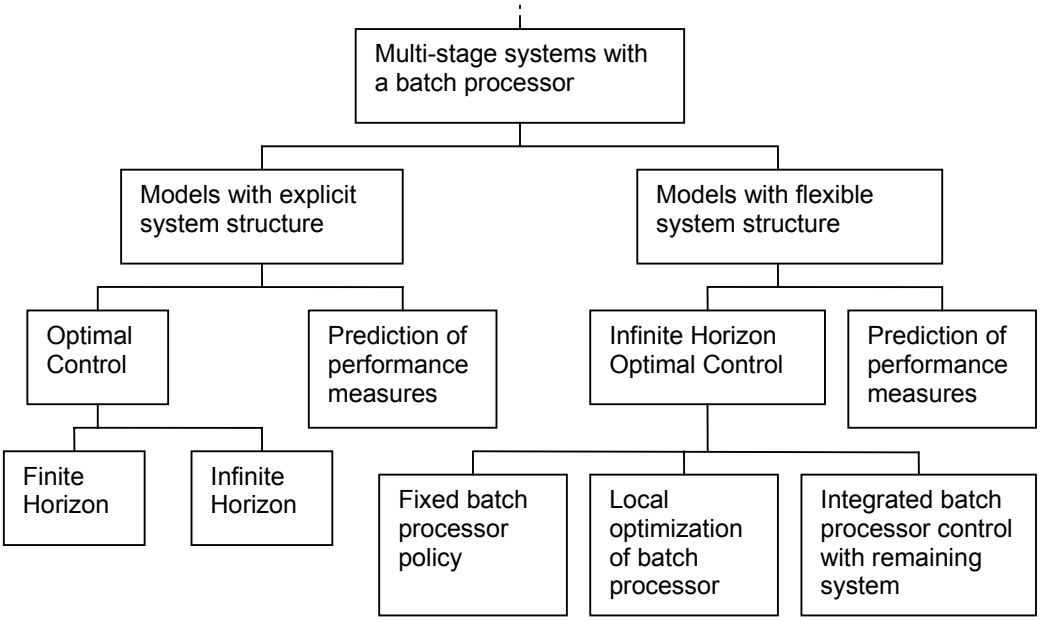


Figure 2.4: Classification scheme for multi-stage systems with a wafer fab type batch processor. This figure is a subsection of Figure 2.1.

2.2.2.1 Explicit enumeration of system structure

For brevity, the system is represented in the form: X→Y, where X and Y represent stages that contain serial processors (represented as δ) or batch processors (represented as β). Unless specified otherwise, the stage is assumed to contain only one processor.

2.2.2.1.1 Prediction of Performance Measures

[Chang and Gershwin 2005] consider a two-stage system composed of two unreliable batch processors with a finite buffer in between ($\beta_1 \rightarrow \beta_2$). Both processors operate under full-batch policy. The processing, failure and repair times are all exponential, but the processor can fail only if it is busy. The two-stage system is modeled as a continuous time discrete state Markov chain, and the steady-state probabilities are obtained. The authors prove that deadlock could occur if the buffer capacity is low. Furthermore, it is also proven that when the sizes of the

batch processors are not relatively prime with each other, the system is not ergodic. The two-stage system in [Chang and Gershwin 2005] also demonstrated behavior that is fairly non-intuitive. For example, when the batch processor size of the first processor is increased, there is initially an increase in throughput, until a certain level where the throughput starts dropping. This is attributed to the increased blocking probability that occurs when the batch processor size is large.

2.2.2.1.2 Finite Horizon Optimization Problems

[Ahmadi *et al.* 1992] consider the static problem of scheduling n jobs on several multi-stage models that contain at least one batch processor. All jobs belong to a single job family. Jobs may have different processing times at the serial processor. The authors establish the optimal policies for minimizing makespan for a $\delta \rightarrow \beta$ system, a $\beta \rightarrow \delta$ system, and a $\beta_1 \rightarrow \beta_2$ system. A DP algorithm that runs in $O(n^3)$ time is proposed to minimize the total completion time on a $\delta \rightarrow \beta$ system; this algorithm is also applicable for a $\beta_1 \rightarrow \beta_2$ system. Unfortunately, minimizing the total completion time on a $\beta \rightarrow \delta$ system is NP-hard, although two proposed heuristics are shown to have reasonably good performance. The minimization of makespan on a three-stage $\delta_1 \rightarrow \beta \rightarrow \delta_2$ system is proven to be NP-hard, and a heuristic with a proven worst-case performance of two is provided.

[Ahmadi *et al.* 1992] also obtain the policy that minimizes the makespan for a $\beta \rightarrow \delta$ system when n jobs to be scheduled belong to one of m job families, and only jobs belonging to the same job family can be batched together. To the author's knowledge, this is the only work on an explicitly enumerated system involving a batch processor that considers the possibility of multiple job families. The rest of the papers discussed in this section (Section 2.2.2.1.2) assume jobs belong to a single job family.

[Kim and Kim 2002] consider the static total completion time minimization on a $\beta \rightarrow \delta$ system problem initially analyzed by [Ahmadi *et al.* 1992] and suggest the use of genetic algorithms (GA) to distribute jobs into batches. Computational experiments show that seeding GA with

some solutions from the heuristic proposed by [Ahmadi *et al.* 1992] will outperform a GA whose seeds are entirely randomly generated.

[Sung and Min 2001] discuss the static problem of scheduling n jobs for a $\delta\rightarrow\beta$, a $\beta_1 \rightarrow \beta_2$, and a $\beta\rightarrow\delta$ system. The objective is to minimize the E/T measure, and all jobs have a common due date. Jobs have individual processing times at a serial processor.

[Sung and Kim 2003] consider the static problem of scheduling n jobs in a $\beta_1 \rightarrow \beta_2$ system. By identifying certain properties of optimal solutions, three optimal efficient dynamic programming algorithms are provided to minimize the maximum tardiness, the number of tardy jobs and the total tardiness.

[Sung *et al.* 2000] look into the static problem of scheduling n jobs in a flowshop composed of m batch processors. Each processor's processing time and size may be distinct. After showing that a batch processor with greater batch processor size and shorter processing time than an adjacent batch processor can be ignored in the optimization of a regular performance measure, two heuristics are proposed, one for minimizing the makespan and another for minimizing the sum of completion times.

2.2.2.1.3 Infinite horizon optimization problems

[Gurnani *et al.* 1992] consider a $\delta\rightarrow\beta$ system where the upstream stage is composed of m parallel serial processors. All processors have deterministic processing times, but the serial processor is assumed to be unreliable. The authors determine the optimal threshold policy when the batch processor has unlimited size. Numerical experiments also suggest that this optimal policy can also be used when the batch processor size is sufficiently large.

[Neale and Duenyas 2000] analyze the problem of controlling a single-product, multi-stage system involving one batch processor, with future job arrivals. For the deterministic, finite horizon problem, an optimal algorithm that runs in $O(n^3)$ is developed to minimize the sum of completion times for a $\delta \rightarrow \beta$ network. Because all jobs are identical to each other, the problem of sequencing jobs at serial processors upstream of the batch processor is trivial. A pseudo-polynomial optimal algorithm is also developed for a $\beta \rightarrow \delta$ network.

When only the arrival time and processing time distributions are known, the authors develop a DP formulation that minimizes the average number of jobs in a $\delta \rightarrow \beta$ system, and found that the optimal policy at the batch processor is frequently a threshold policy whose threshold is strictly non-decreasing with respect to the serial processor queue length. Their results also show that considering the status of the serial processor can result in a significant reduction in the number of jobs inside the system over the optimal policy when the batch processor is considered in isolation. However, the difficulty in obtaining the optimal policy necessitated the development of a two control limit heuristic (TCLH), where the minimum batch size is smaller when there are no jobs in front of the upstream serial processor, as compared to when the serial processor is busy. Simulation experiments show that TCLH also results in a significant improvement over the best MBS policy.

The same problem approach was used for a $\beta \rightarrow \delta$ system, and, for all the problem instances considered, the optimal policy at the batch processor is frequently a threshold policy whose threshold is strictly non-increasing with respect to the number of jobs in front of the downstream serial processor. A direct comparison with the optimal policy of the batch processor considered in isolation results in an average reduction that is approximately an order lower than the reduction obtained for a $\delta \rightarrow \beta$ system.

[In *et al.* 2003] shows how the policy that minimizes the average cost of operating a single-product, two-machine line with an unreliable upstream serial processor and a downstream batch processor with infinite size can be obtained, assuming deterministic processing times. The system incurs a constant setup cost per batch processor loading, a delay cost per unit time per unit job, and an idling cost per unit time.

None of the previous work mentioned in this review consider the possibility of a processing time window of a job that exists between the batch processor and its upstream processor. This problem is investigated in Chapter 4. Furthermore, previous work has also ignored the possibility that jobs belong to incompatible job families. We examine the performance of two- and three-stage systems involving a downstream batch processor, when jobs belong to incompatible job families, under simple control policies in both Chapter 5 and Chapter 6.

2.2.2.2 Generalized system structure

Research on the control and behavior of wafer fabs are filed under this category. Unfortunately, consideration of the batch processors inside the wafer fab has rarely been emphasized. Consequently, a significant amount of research on wafer fab control and behavior fail to even mention the presence of batch processors, leaving industry to make its own adjustments to accommodate batch processors. This provides ample opportunity for future research to build upon the results from this dissertation and fully bridge the existing gap.

We divide the discussion into two: prediction of performance measures and optimal control.

2.2.2.2.1 Prediction of performance measures

There has been significant work done in estimating the performance of manufacturing systems involving batch processors, although the batch processor policies assumed are typically simplistic. [Schomig and Kahnt 1995] consider a single-family production system with at least one batch processor. Batch processors are assumed to have a minimum batch size policy, and the production system is under single-card kanban system. [Park *et al.* 2000] propose a modified form of Mean Value Analysis (MVA) to predict the throughput and cycle time of a reentrant flowshop that contains at least one batch processor. The batch processor is assumed to operate under full-batch policy, and an explicit order of priority of buffer exists at each stage. The analysis is subsequently extended to include the system processing different job classes, and where each workstation may have multiple machines ([Park *et al.* 2002]).

[Chung and Huang 1999] propose a block-based cycle time estimation algorithm that segments the wafer fab into blocks of workstations that start and end with a batch processor. Batch processors are assumed to operate under a minimum batch size policy. This estimation algorithm is subsequently modified to allow for multiple priority lots. [Chung *et al.* 2003] propose a hierarchical approach to wafer fab production planning, based on the cycle time estimation method proposed by [Chung and Huang 1999]. Batch processor threshold policies are set based on the product mix, priority mix and target throughput rate.

[Noben *et al.* 2001] assume batch processors operate on a maximum waiting time policy. When a partial batch is available and there are no hot lots, then the batch processor is under a simple threshold policy. On the other hand, hot lots are processed without waiting. Based on the results obtained from a detailed wafer fab simulation model, using more batch processors with smaller maximum sizes can significantly reduce the average cycle time of lots.

2.2.2.2.2 Optimal Control

The control of a wafer fab has been one of the most significant research areas in recent years. Entire journals are dedicated for this problem. We segregate the discussion into three sections, with each succeeding section placing more emphasis on the batch processors.

- ***Fixed control policy at the batch processor***

A fixed control policy at the batch processor essentially indicates that the batch processor control policy is considered a constant, and no attempt is made to optimize the batch processor policy. It is likely that a large majority of wafer fab control research uses this type of policy at the batch processor, if such a policy is not mentioned. We only mention research found on this section to give an idea how batch processors are considered in a manufacturing system dominated by serial processors by a significant number of researchers.

[Kim *et al.* 1998] assume that the batch processors are under a minimum batch size policy and selection of batches is on first-come-first-served policy. One of the wafer fab simulation models used by [Hsieh *et al.* 2001] contains a batch processor that is assumed to operate on full-batch policy. [Tsai *et al.* 2003] consider using different dispatching rules, according to the utilization of the processor. Batch processors are assumed to be under full-batch policy, except when the batch processor is upstream of the bottleneck, and the bottleneck has a low WIP level. In that situation, the batch processor is allowed to process a partial batch.

- ***Local optimal control of the batch processor***

This approach typically considers the control of the batch processor as a separate problem to the control of the serial processors. Consequently, it is popular to adapt one of the suggested policies in Section 2.2.1 for the control of the batch processor. For example, [Chen *et al.* 2001]

use simulation to determine the optimal batch processor threshold policy, [Uzsoy and Wang 1997] propose using the release date update heuristic proposed in [Uzsoy 1995] to solve the local scheduling problem required under their shifting bottleneck approach to controlling the entire manufacturing line. [Rulkens *et al.* 1998] develop a wafer fab simulation model that models the furnace area in greater detail; this model is used to find the optimal minimum batch size policies to be used per process, per priority.

[Akcali *et al.* 2000] use a simplified simulation model of a wafer fab to evaluate several threshold policies. The authors determine that threshold policies based on individual recipes (each product may have several recipes) outperform those based on products, which in turn outperform a single threshold policy across the entire facility.

- ***Integrated control of batch processors with other processors of the system***

[Mason *et al.* 2002] propose a modified shifting bottleneck (MSB) approach (based on the original shifting bottleneck approach by [Adams *et al.* 1988]) for scheduling wafer fabs to reduce the total weighted tardiness. For each stage, the apparent tardiness cost (ATC) heuristic is used for dispatching; a variant that has been adjusted for batching is used for controlling batch processors. Numerical experiments also suggest that scheduling the batch processor ahead of the other stages tend to produce good wafer fab-wide performance. [Oey and Mason 2001] present a cycle elimination procedure for the modified shifting bottleneck approach used in [Mason *et al.* 2002]. [Monch *et al.* 2006] propose using genetic algorithms instead of the ATC heuristic for critical processors inside the system. The genetic algorithms (GA) approach has been previously tested on a single stage of batch processors ([Monch *et al.* 2005]) and is now tested on an entire system. Results suggest that the solution quality is positively correlated with the number of stages the GA is used, although the computational time increases. Additional improvements have also been suggested by [Monch and Driessel 2005].

Aside from the set of interrelated research discussed, there has been little work done on the integrated control of batch and serial processors for complex systems. While work on multi-stage systems with a specific system structure involving a batch processor is not rare, majority

of the work has considered only jobs coming from a single job family. This is a fertile area for future research.

2.3 Batch Processors as Burn-in Ovens

Although the burn-in model is not considered in this dissertation, some of the research done on burn-in ovens is still relevant. In particular, Section 2.3.2.2 discuss perhaps the only instances batch processors explicitly influence the control of a large manufacturing system.

2.3.1 Burn-in ovens as a single entity

Majority of the research involving burn-in ovens assume that the burn-in stage exists in isolation. We segregate the discussion on burn-in ovens in isolation into finite-horizon and infinite-horizon problems. Figure 2.5 illustrates how the discussion is further divided.

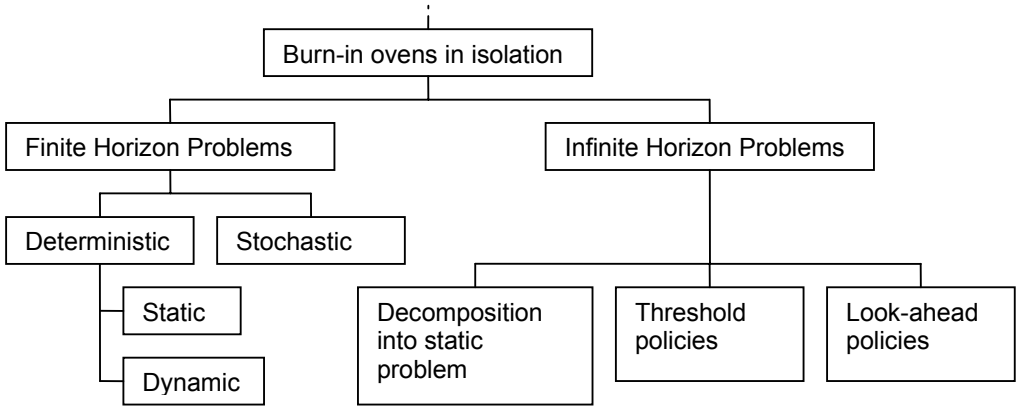


Figure 2.5: Classification scheme for literature on burn-in ovens in isolation.

This figure is a subsection of Figure 2.1. Majority of the work done on burn-in oven control in isolation can be classified under finite-horizon, deterministic problem.

2.3.1.1 Finite Horizon Problems

We divide the discussion into models with purely deterministic data, and models which incorporate uncertainty

2.3.1.1.1 Deterministic Problems

Much research has been poured into the deterministic problem of scheduling jobs on a finite horizon on a single stage containing one or more burn-in ovens. For convenience, we present the work done in tabular form (Table 2.2). We use the *A/B/C/D* notation commonly used in scheduling literature(for example: [Conway *et al.* 1967]) to describe the problem concisely. Let *A* describe the job arrival process, *B* describe the number of machines, *C* describe the flow pattern of the shop, and *D* describe the objective function.

The following notations are used for convenience:

B = burn-in oven with finite size. Literature which assumes infinite batch processor size (for example: [Deng *et al.* 2004] and [Liu *et al.* 2003]) are ignored, as the batch processor size is a strong constraint in semiconductor manufacturing.

r_j = jobs have individual release times. Let $||r_j||$ be the number of distinct values of r_j. Similarly, p_j refers to the individual minimum processing times, w_j refers to jobs having individual weights, and s_j refers to jobs having non-identical job sizes.

C_j refers to the completion time of a job. CMAX is the makespan. LMAX is the maximum lateness of a job. E/T refers to an earliness/tardiness measure, which is $\sum|c_j - d_j|$.

PTAS = Polynomial Time Approximation Scheme. If a **polynomial time approximation scheme** to a certain problem exists, then we can get arbitrarily close to the optimal solution.

TABLE 2.2: DETERMINISTIC FINITE HORIZON SCHEDULING PROBLEMS WITH A SINGLE BURN-IN.

Publication	Problem considered and contribution
[Lee *et al.* 1992]	Proposed optimal efficient DP-based algorithms for the following problems: • *n, r_j /1/B, p_j = p, d_j /TMAX* • *n/1/B, p_j, d_j, p_j and d_j agreeable/TMAX* • *n, r_j /1/B, p_j = p, d_j /TNUM* • *n /1/B, p_j, d_j, p_j and d_j agreeable/TNUM* Provided heuristics with proven worst-case bounds for both the *n/m/B, p_j /CMAX* and the *n/m/B, p_j, d_j /LMAX* problems
[Chandru *et al.* 1993]	Developed a branch and bound algorithm for the *n/1/B, p_j /$\sum C_j$* problem. Heuristics based on the properties of optimal solutions are suggested. The heuristics are also adapted for the *n/m/B, p_j /$\sum C_j$* problem.
[Uzsoy 1994]	Proved that the *n/1/B, p_j = p, s_j /CMAX* problem is NP-Hard, and suggested heuristics for the *n/1/B, p_j, s_j /CMAX* problem. Proved that the *n/1/B, p_j = p, s_j /$\sum C_j$* problem is NP-Hard, and provided a branch-and-bound based optimal algorithm for the *n/1/B, p_i, s_i /$\sum C_j$* problem.
[Hochbaum and Landy 1997]	Proposed an optimal algorithm for the *n/1/B, p_j /$\sum C_j$* that is efficient if $||p_j||$ is fixed. If $||p_j||$ is not fixed, then a heuristic with a known worst-case bound is proposed. A heuristic with known worst-case bound is also proposed for the *n/m/B, p_j /$\sum C_j$* problem, if $||p_j||$ is fixed.

[Uzsoy and Yang 1997]	Proposed an optimal branch and bound algorithm for the $n/1/B, p_j, w_j/\sum w_jC_j$ problem, which is suspected to be NP-hard. Several heuristics that can be combined with a local search procedure were also suggested for larger problem instances.
[Li and Lee 1997]	Proved that both the $n, r_j/1/B, p_j, d_j/TMAX$ and the $n, r_j/1/B, p_j, d_j/TNUM$ problems are NP-hard. Provided efficient solutions to both problems when r_j, p_j and d_j are agreeable.
[Ghazvini and Dupont 1998]	Proposed and evaluated several heuristics for the $n/1/B, s_j, p_j/\sum C_j$ problem.
[Sung and Choung 2000]	Proposed a simple optimal algorithm for the $n/1/B, p_j/CMAX$ problem. Proposed a branch and bound optimal algorithm for the $n,r_j/1/B, p_j/CMAX$ problem. Also developed worst-case bounds for several heuristics, which can be combined to form one heuristic.
[Liu and Yu 2000]	Proved that the problem $n, r_j/1/B, p_j/CMAX$ is NP-hard. Also provided optimal algorithm that runs in polynomial time if $\|\|rj\|\|$ is fixed.
[Cheng *et al.* 2001]	When p_j and d_j are agreeable, the $n, r_j/1/B, p_j, d_j/TNUM = 0$ problem is NP-Complete. Optimal efficient algorithms are proposed for the problem with the following restrictive assumptions: • r_j and d_j are agreeable, • r_j and p_j are agreeable • $\|\|r_i\|\|$, $\|\|d_i\|\|$ and $\|\|p_i\|\|$ are fixed.
[Wang and Uzsoy 2002]	Developed a GA-based heuristic for the $n, r_j/1/B, p_j, d_j/LMAX$ problem. The GA uses a DP-based heuristic that provides an upper bound on LMAX, if the job sequence is fixed, while GA changes the job sequence.
[Dupont and Dhaenens-Flipo 2002]	Determined some properties of the optimal solution to the $n/1/B, s_j, p_j/CMAX$ problem that is used with a branch and bound-based heuristic
[Cai *et al.* 2002]	Provide a PTAS to the $n/1/B,p_j/\sum C_j$ problem
[Sung *et al.* 2002]	Proposed an optimal solution to the $n,r_j/1/B, p_j/CMAX$ problem that is polynomial when $\|\|p_i\|\|$ is fixed.
[Deng *et al.* 2003]	Proposed an optimal algorithm to the $n, r_j/1/B, p_j/CMAX$ that is polynomial if $\|\|r_j\|\|$ is constant. A PTAS to the general problem is also proposed. The algorithm can be adapted for use in an online setting, for which a worst-case bound is proven.
[Melouk *et al.* 2004]	Proposed using simulated annealing to determine the optimal schedule of an $n/1/B,s_i, p_i/CMAX$ problem.
[Poon and Zhang 2004]	Proposed an optimal solution to the $n, r_j/1/B,p_j/CMAX$ problem that is efficient when $\|\|r_i\|\|$ and $\|\|p_i\|\|$ are fixed.
[Li *et al.* 2004]	Provided a PTAS to the $n, r_j/m/B,p_j/LMAX$ problem
[Pei-Chann and Hui-Mei 2004]	Identified properties of optimal solution to the $n, r_j/1/B, s_j/\sum C_j$ problem and used these properties to create a heuristic for the problem.
[Zhang *et al.* 2005]	Considered the $n, r_j/m/B, p_j$ *dependent on the batch processo* $/CMAX$ problem, given the assumption that the order of the jobs in terms of processing times is constant for each processor. A PTAS is provided for the problem, and the optimal solution to the problem is efficient if $\|\|r_j\|\|$ is fixed and all r_j, p_j are integral.
[Cheng *et al.* 2005]	proposed an efficient optimal solution to the $N,r_j/1/B, p_j = p/$*any regular objective* problem The optimal solutions to several variants of the problem where jobs are assumed to have precedence constraints are also proposed.
[Li *et al.* 2005]	Presented an approximation algorithm for the $n, r_j/1/B,p_j,s_j/CMAX$ problem with worst-case ratio $2 + \varepsilon$
[Li *et al.* 2005]	Proposed a PTAS for the $n, r_j/m/B, p_j/CMAX$ problem
[Pei-Chann *et al.* 2005]	Provided a constraint programming representation of the $n, r_j/1/B, p_j, s_j/\sum C_j$ problem. Properties of the optimal solution are also established.
[Monch and Unbehaun 2006]	Proposed three different heuristics for the $n/m/B, p_j, d_j = d/E/T$ problem.
[Li and Yuan 2006]	Provided an optimal solution in polynomial time that also optimizes the time interval the processor was used, and $\sum w_i(CMAX - C_i)$

2.3.1.1.2 Stochastic problems

A finite scheduling horizon typically assumes that all data are deterministic, although there are exceptions. [Koole and Righter 2001] show that the optimal policy for the static problem of scheduling a burn-in oven may have counter-intuitive properties when the distributions for the minimum job processing times are general. [Chen *et al.* 2004] propose a threshold policy, where the threshold is in terms of the time elapsed since the last job arrival. Though the algorithm is intended for a finite horizon problem, the algorithm does not explicitly use the knowledge of *n* jobs to be scheduled.

2.3.1.2 Infinite Horizon Problems

Solutions to infinite horizon problems have to be generated by on-line algorithms. Compared to the wafer fab model, there has been limited work done for these types of problems. We divide the discussion into three stages.

2.3.1.2.1 Decomposition into static problem

This approach reduces the problem complexity by ignoring future job arrivals at each instance a decision has to be made. One implication of using this approach is that the burn-in oven will always be kept busy if the performance measures are regular. [Ganesan *et al.* 2004] use conjunctive simulated scheduling to approximate the Pareto curve for the problem of optimizing the control of a batch processor with future yet unknown job arrivals with respect to two different objectives.

2.3.1.2.2 Threshold policies

The threshold policies that are quite popular for wafer fab models of batch processors are less commonly applied to burn-in ovens, as the processing time of the batch can be affected by the batch composition. [Avramidis *et al.* 1998] propose a threshold policy, which restricts the batch composition to the case where jobs are processed in the order they arrive, and the processing time of all jobs are from an identical distribution type (with possibly different parameters). The job arrival rate is assumed to be Poisson.

2.3.1.2.3 Look-ahead policies

[Neale and Duenyas 2003] consider the problem of controlling a burn-in oven with unknown arrivals and stochastic processing times. The processing time of the batch assumes the distribution of the job inside the batch with the highest expected minimum processing time. A look-ahead policy that is very similar to that proposed by the authors for the batch processor control problem with incompatible families ([Duenyas and Neale 1997]) is also proposed.

[Van Der Zee 2004] also considers the problem of controlling a burn-in oven in an online setting, and propose a look-ahead strategy that considers predictions of future job arrivals when determining whether to process a batch or not.

2.3.2 Multi-stage models involving a burn-in oven

We divide the discussion into two segments: research that assume a specific system structure, and research that have a more generalized structure.

2.3.2.1 Explicit enumeration of system structure

[Damodaran and Srihari 2004] propose mixed integer models for the static problem of scheduling n jobs with unequal sizes and minimum processing times on a $\beta_1 \rightarrow \beta_2$ system. [Oulamara 2005] analyze the static problem of scheduling n jobs in a two-machine system. The first processor is a burn-in oven with finite size, while the processing time of the batch at the second processor is the sum of setup time and the processing times of jobs inside the batch. There's no buffer between the two processors.

[Danneberg *et al.* 1999] look at several algorithms for minimizing CMAX and $\sum w_j C_j$, separately, in scheduling n jobs on a flowshop of batch processors. The jobs have incompatible job families and a setup is required when switching job families. However, not all jobs from the same job family have the same processing time.

2.3.2.2 Generalized system structure

Two papers discuss the idea of subordinating the control of upstream processors to the batch processor. Both [Bhatnagar *et al.* 1999] and [Chandra and Gupta 1997] propose controlling the lot releases based on an ideal burn-in loading plan. In [Chandra and Gupta 1997], the burn-in ovens found in the semiconductor packaging line strongly mimic the characteristics of those found in the wafer fab: jobs belong to job families, and an oven can only contain jobs belonging to a job family. The processing time of the burn-in oven is also assumed to be constant. [Bhatnagar *et al.* 1999] considers a similar system where batch processors have significant setups in between wiring configurations.

[Bhatnagar *et al.* 1999] and [Chandra and Gupta 1997] are significant for their attempt to tie the lot release with a burn-in loading scheme. However, there is no explicit stage-per-stage synchronization present in either paper, since the sequencing at the intermediate stages is not performed with respect to the batch processor requirements. While lot order release typically has a greater impact than the ordering of intermediate steps for systems comprised of serial processors ([Wein 1988] and [Sivakumar 2000]), we believe that there is still significant improvements that can be made in explicitly considering the intermediate stages in front of the batch processor.

2.4 Area of Contribution of Research

The work we present in this dissertation only deals with batch processors that process jobs belonging to incompatible job families; the burn-in model is not considered. While there has been considerable work done in controlling the batch processor with incompatible job families as a single entity, we have not come across any work that proposes an optimal algorithm to this problem. Furthermore, the complexity of this problem has not been formally analyzed.

For the problem of optimally controlling a batch processor with future job arrivals and incompatible job families, we analyze both the finite and infinite horizon versions of the problem. For the finite horizon version, we develop both integer linear programming and

dynamic programming formulations for minimizing mean cycle time of jobs passing through the processor. The optimal DP algorithm is then used as the engine behind a Model Predictive Control-based heuristic that is used for the infinite-horizon problem. This approach is also applied to the case where the batch processing stage contains more than one batch processor in parallel. These contributions are detailed in Chapter 3 and in Appendix A.

From the review of look-ahead methods, we observe two things: firstly, the look-ahead methods and their underlying models become more and more complex. Secondly, while the additional refinements of look-ahead methods have resulted in improvements, the amount of incremental improvement observed has been decreasing. For this reason, we consider looking at the processors immediately upstream of the batch processor, in order to control the amount and type of job that flows into the batch processor. We believe only [Ahmadi *et al.* 1992] has considered an explicitly specified multi-stage system where the jobs belong to different job families. We consider controlling a system that has an infinite horizon, with job arrivals occurring in the future, whereas [Ahmadi *et al.* 1992] looked into a finite horizon problem with all jobs initially available. Furthermore, we consider a different system configuration from what [Ahmadi *et al.* 1992] looked at.

We start our analysis of the upstream processor control by looking at an often-ignored problem in batch processors inside the wafer fab: processing time windows. In wafer fabrication, it is typical that jobs exiting the upstream processor have to be processed by the batch processor within a certain time period. However, the literature we have covered have largely ignored the possibility of jobs having a processing time window, with only [Makis 1985] considering processing time windows for his single batch processor model. We bridge this gap in Chapter 4, where we analyze the problem of controlling the entry of jobs into the batch processor buffer. The resulting upstream processor policy effectively limits the amount of jobs entering the batch processor buffer, and could be viewed as determining the buffer space to be allocated between the batch processor and its upstream processor, with regards to the existence of processing time windows.

There has also been little work done on the control of infinite-horizon, multi-stage systems involving a batch processor. [Gurnani *et al.* 1992], [Neale and Duenyas 2000] and [In *et al.* 2003] all consider the case where jobs belong to only one job family. Even for this simpler problem, the optimal policy is difficult to characterize ([Neale and Duenyas 2000]). We fill this gap by characterizing the performance of two-and three-stage systems involving a downstream batch processor in Chapters 5 and 6, under simple feeder processor control policies. Chapter 5 assumes that the upstream processor is a serial processor, while Chapter 6 assumes that the upstream processor is another batch processor.

We believe our work is the first to demonstrate the feasibility of constraining the control of the upstream processors according to the anticipated needs of the downstream batch processor in reducing cycle time. Figure 2.6 illustrates the main questions we seek to answer.

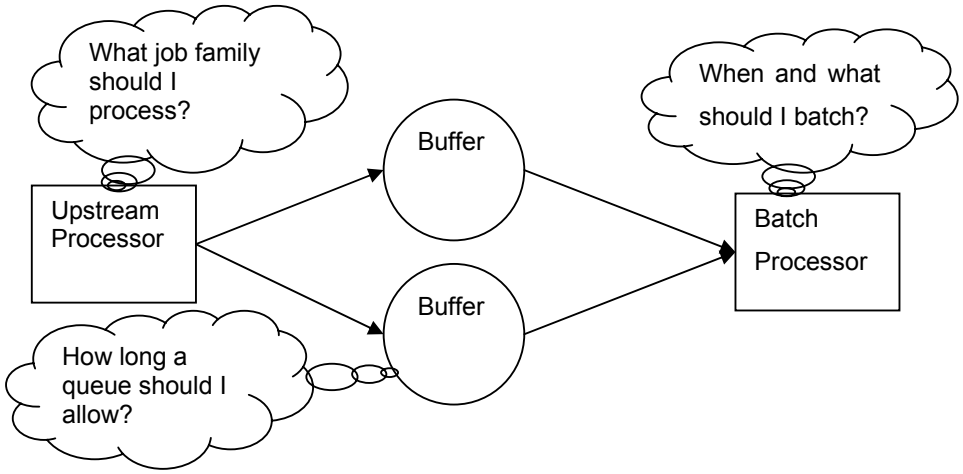


Figure 2.6: Visual summary of contributions of this dissertation.

We answer three questions regarding a manufacturing system with batch processors, to show that constraining the upstream processor to the needs and status of the batch processor can be beneficial for the entire system.

2.5 Chapter Summary

In this chapter, we reviewed the literature present on batch processors for semiconductor manufacturing. In doing so, we generated a classification scheme, which segregates the previous work done according to model family, horizon length and purpose. We also identify several gaps in the literature which need to be bridged before a comprehensive understanding

of the importance of batch processors in the control of large manufacturing systems can be attained, and show how our work bridges these gaps .

Chapter 3 Control of a Batch Processor in Isolation

3.1 Introduction

In semiconductor wafer fabs, the oxidation and diffusion ovens or furnaces are batch processors with incompatible job families. The furnace can process more than one job at once, and the processing time is independent of the number of jobs loaded into the furnace. Jobs queued in front of the furnaces require specific chemical recipes and furnace temperatures, making it impossible to process certain jobs together as a batch. Jobs that can be processed together form a job family.

In this chapter, we look at the control of an individual furnace assumed to be isolated from the remainder of a manufacturing system; jobs arrive in front of the batch processor at certain time instances, and the operator at the batch processor has to decide whether to wait for more jobs to arrive, or to load a batch of jobs belonging to a single job family into the batch processor. This is a nontrivial decision, since the length of the batch processor processing time is assumed to be significantly longer than the average time until next arrival. An extension of the work done in this chapter can be found in Appendix A, where the batch processing stage can contain more than one batch processor.

3.2 Use of Offline Deterministic Methods for Online Batch Processor Control in a Stochastic Environment

The problem of controlling a manufacturing stage containing a single batch processor has been typically approached using one of two methods. The first method typically assumes a finite control horizon (there are a fixed number of jobs *n* that are to be scheduled) and deterministic data. These algorithms can be termed **offline algorithms**, as these algorithms can be executed once before production has started, and the algorithm output dictates at which times each job is to be processed, with certainty.

The second method assumes that the batch processor is part of a complex manufacturing system exhibiting stochastic behavior. Because it is expected that the system will operate indefinitely, the number of jobs to be processed by the batch processor is infinite. Algorithms derived from this type of model are generally called **online algorithms**. These algorithms are executed regularly, and only a limited number of decisions are made.

Online algorithms typically have to run much faster than offline algorithms, as online algorithms are repeatedly executed, and the processor is left idle while waiting for the online algorithm output. One possible method of coping with algorithm processing time constraints is Model Predictive Control (MPC). In **Model Predictive Control (**[Bertsekas 2005]), it is typical to convert stochastic values into deterministic values, as deterministic problems are often more computationally tractable than stochastic problems. At each instance a decision has to be made, model predictive control reduces the original problem into a problem with a shorter horizon. Solving this smaller problem to optimality yields a sequence of controls to be executed at particular time instances. Only the first control is implemented on the original problem, the rest are discarded. This process is repeated at each instance a decision has to be made.

We wish to evaluate the performance of a MPC-based heuristic for controlling the batch processor.

3.3 Complexity of the Deterministic Finite-Horizon Optimal Batch Processor Scheduling Problem

We look into the theoretical complexity of a version of the optimal batch processor scheduling problem, and empirically test a dynamic programming algorithm for solving the problem. This is to determine the largest problem instance that can be solved within a feasible time interval.

3.3.1 Problem Statement

We consider n jobs to be scheduled on a batch processor. The batch processor can simultaneously process up to Q jobs, and the processing time is independent of the number of jobs being processed. Each Job i becomes available at time instance r_i, and belongs to a Job Family j. There are m total job families. Only jobs belonging to the same family can be batched together. The processing time of a batch is dependent only on the job family that is currently being processed. The objective is to minimize the sum of completion times. The batch processor is assumed to be initially available, and jobs are indexed in increasing order of their release times.

3.3.2 Problem Complexity

This problem can be written as an integer linear program.

Variables:

n number of jobs to be scheduled

m number of job families

i index for jobs, $i = 1, 2, \ldots, n$. Let I be the set of all jobs.

j index for families $j = 1, 2, \ldots, m$

k index for batches $k = 1, 2, \ldots, n$. Batches are processed in increasing order of their index. The number of batches cannot exceed n, as there has to be at least one job in one batch.

f_j set of job indices in family j. The sets f_j, $j = 1, 2, ..., m$ form a partition of the set I. If $f_1 = \{1, 2, 3, 4\}$, Only Jobs One, Two, Three and Four belong to Job Family One.

Q batch processor size.

r_i time instance Job i arrives in front of the batch processor.

p_j processing time for Job Family j

C_k time instance Batch k finishes

S_k time instance Batch k starts

The decision variables are:

x_{ik} = 1 if Job i is assigned to Batch k, and = 0 otherwise

y_{jk} = 1 if Job Family j is assigned to Batch k, and = 0 otherwise

The integer linear program formulation becomes:

$$\text{Min} \sum_{k=1}^{n}\sum_{i=1}^{n} C_k x_{ik} \qquad (3.1)$$

Subject to:

$$\sum_{i=1}^{n} x_{ik} \le Q \quad \forall k \qquad (3.2) \qquad \sum_{k=1}^{n} x_{ik} = 1 \quad \forall i \qquad (3.3)$$

$$\sum_{j=1}^{m} y_{jk} \le 1 \quad \forall k \qquad (3.4) \qquad x_{ik} \le y_{jk},\ i \in f_j \quad \forall k, \forall j \qquad (3.5)$$

$$C_k = S_k + \sum_{j=1}^{m} p_j y_{jk} \quad \forall k \qquad (3.6) \qquad S_k \ge r_i x_{ik} \quad \forall k,\ \forall i \qquad (3.7)$$

$$S_k \ge C_{k-1} \quad \forall k > 1 \qquad (3.8) \qquad x_{ik}, y_{jk} \in \{0,1\} \quad \forall k, \forall i, \forall j \qquad (3.9)$$

Equation 3.1: Objective function minimizes the total completion time

Equation 3.2: no batches violate the size constraint.

Equation 3.3: all jobs get assigned to a batch.

Equation 3.4: At most one job family is assigned to any batch.

Equation 3.5: a job is not assigned to a batch if the batch is not assigned to the job family the job belongs to. Equations 3.4 and 3.5 ensure only empty batches can avoid a job family assignment.

Equation 3.6: Computation of completion time of each batch

Equation 3.7: A batch does not start processing until all jobs assigned to the batch has arrived

Equation 3.8: A batch does not start processing until the previous batch has been finished.

Equation 3.9: constrain integer decision variables to 0 and 1

A solution that reduces the sum of completion times also reduces the average cycle time, since

$$\left(\sum_{k=1}^{n} \sum_{i=1}^{n} C_k x_{ik} - \sum_{i=1}^{n} r_i \right) \Big/ n = AveCycleTime .$$

This problem is strongly NP-Hard; if we let $Q = 1$, then the problem is reduced to one that has been analyzed by [Lenstra *et al.* 1977] to be strongly NP-Hard. When there is only one job family, a dynamic programming (DP) algorithm that runs in $O(n^3)$ time can be used to obtain the optimal solution. ([Ahmadi *et al.* 1992] and [Neale and Duenyas 2000]).

NP-hardness of the problem implies that the computational effort, in terms of number of operations, increases at an exponential rate as the problem input size is increased. While this implies that it would eventually be practically infeasible to solve large instances of this problem, small instances of the problem may still be solved within a practical time frame.

3.3.3 Dynamic Programming Algorithm

We propose to use dynamic programming (DP) to solve the optimal batch processor scheduling problem in Section 3.3.1. Two observations are made:

1. decisions need to be made only when
    - a job has arrived when the batch processor is idle, or
    - the batch processor just finished processing
2. the set of possible decisions to be made are:
    - wait for next arrival, or
    - process a batch of jobs belonging to Job Family *j*. The batch size is the minimum between the batch processor size and the number of jobs belonging to Family *j* queued in front of the batch processor at the current time instance.

The variables are:

t – time instance decision has to be made

$q_j(t)$ – queue length for jobs belonging to Family j at time instance t. There are m job families.

a_i – job family to which Job i belongs to. $a_i = \{j \mid i \in f_j\}$.

r_i – time Job i arrives in front of the batch processor.

n_j – total number of jobs to be scheduled that belong to Job Family j. $\sum_{j=1}^{m} n_j = n$

Let the state at Time Instance t be: $(t, q_1(t), q_2(t), \ldots, q_m(t))$.

Let:

- $g_j(t) = max(0, q_j(t) - Q)$. $g_j(t)$ returns the number of jobs belonging to Job Family j still in front of the batch processor right after a batch of Job Family j is loaded.
- $h_j(t) = min(Q, q_j(t))$. $h_j(t)$ returns the number of jobs belonging to Job Family j loaded into the batch processor. $q_j(t) - h_j(t) = g_j(t)$.
- $s(t) = \min\{r_i : r_i > t\}$. $s(t)$ returns the time instance of the next job arrival.
- $V_j(x, y) = \|i \mid (x < r_i \le y) \cap a_i = j\|$. $V_j(x, y)$ returns the number of jobs belonging to Job Family j that arrives in front of the batch processor in the time interval $(x, y]$.

The recursion equation is:

$$f(t, q_1(t), q_2(t), \ldots, q_m(t))$$

$$= \min\left\{\begin{array}{l} f(s(t), q_1(t) + V_1(t, s(t)), q_2(t) + V_2(t, s(t)), \ldots, q_m(t) + V_m(t, s(t))), \\ h_1(t) \times (t + P_1) + f\begin{pmatrix} t + P_1, g_1(t) + V_1(t, t + P_1), q_2(t) + V_2(t, t + P_1), \ldots, \\ q_m(t) + V_m(t, t + P_1) \end{pmatrix}, \\ h_2(t) \times (t + P_2) + f\begin{pmatrix} t + P_2, q_1(t) + V_1(t, t + P_2), g_2(t) + V_2(t, t + P_2), \ldots, \\ q_m(t) + V_m(t, t + P_2) \end{pmatrix}, \\ \vdots \\ h_m(t) \times (t + P_m) + f\begin{pmatrix} t + P_m, q_1(t) + V_1(t, t + P_m), q_2(t) + V_2(t, t + P_m), \ldots, \\ g_m(t) + V_m(t, t + P_m) \end{pmatrix} \end{array}\right\} \quad (3.10)$$

The boundary conditions are:

- $\min\{r_i : r_i > t\} = \infty$ for $t > r_n$ (3.11)
- Let $0 \le \bullet \le n_j$, for all j, then $f(\infty, \bullet, \bullet, \ldots, \bullet) = \infty$ (3.12)

- $f(t,0,0,...,0) = 0$ for $t > r_n$ (3.13)

The minimum completion time is given by *f(0, 0, 0, ... , 0)*.

At a certain state *(t, $q_1(t)$, $q_2(t)$, ..., $q_m(t)$),* there are, at most, *m + 1* controls to choose from. The first control corresponds to waiting for the next job arrival. No costs are incurred for the current state, since no jobs are processed. The future state is obtained by:

- Forwarding the current time instance *t* to the next time instance a job arrives. *s(t)* = $\min\{r_i : r_i > t\}$.
- For each Job Family *j*, determining $V_j(t, s(t))$, and incrementing the number of jobs from Job Family *j* already in front of the batch processor $q_j(t)$ by $V_j(t, s(t))$.

The next *m* controls correspond to the option of processing a batch of jobs belonging to job family *j*, *j* from *1 to m*. If we choose a control corresponding to processing a batch made of jobs from Family *j*, a cost corresponding to the completion time of the jobs selected to be processed is incurred, and the future state is determined by:

- Forwarding the current time instance *t* to $t + p_j$.
- For $q_j(t)$, we first subtract the number of jobs processed by the batch processor, and then add the number of jobs from Job Family *j* that arrived in the time interval $(t, t + p_j]$.
- For $q_k(t)$, $k \neq j$, we add the number of jobs from Job Family *k* that arrived in the time interval $(t, t + p_j]$.

Boundary Conditions from Equations 3.11 and 3.12 ensure that the algorithm will not wait for additional jobs to arrive when no jobs will arrive. Boundary Condition 3.13 sets the cost of states where all jobs have been processed to zero.

3.3.3.1 Theoretical Complexity of Algorithm

The input size of the problem, when all inputs are integer, is calculated as follows:

- Batch processor size *Q* takes *Log(Q)* bits.
- The times *n* jobs arrive take at most *n* $Log(r_n)$ bits.

- The job families for n jobs take at most $n\ Log(m)$ bits.
- The processing time for m job families takes at most $m\ Log\ (max_{\ j\ =\ 1\ to\ m}\ P_j)$ bits.

The problem input size is then bound from above by $Log(Q) + n\ (log(r_n) + log(m)) + m\ Log\ (max_{\ j\ =\ 1\ to\ m}\ P_j)$.

To explicitly generate an upper bound of the problem input size in terms of n and m, we assume:

- $m > max_{\ j\ =\ 1\ to\ m}\ P_j$.
- we store the difference in release times $(r_i - r_{i\ -\ 1})$, instead of r_i, and $m > max_{\ i\ =\ 1\ to\ n}(r_i - r_{i\ -\ 1})$, where r_0 is set to zero.
- $Q < n$. For the batch processor size Q to be meaningful, $Q < n$, otherwise we can assume infinite batch processor size.

The upper bound can then be expressed as $Log(Q) + 2n\ log(m) + m\ Log\ (max_{\ j\ =\ 1\ to\ m}\ P_j)$, or $Log(n) + (2n + m)\ log(m)$. (There can be at most n job families and $Q < n$, so another input size upper bound is $(3n + 1)\ log(n)$)..

There can be at most n distinct time instances a job will arrive. For each Job Family j, the maximum number of processed batches is $n_j < n$. This occurs when each batch is composed of only one job. Thus, the number of different time instances in which we have to make a decision is bounded by $n(n + 1)^m$. The first factor n accounts for the job arrival time instances, the next m factors are $(n_j + 1) \leq (n + 1)$, which take into account the number of batches of Job Family j already processed. One is added to n_j to account for the possibility that there can be zero batches processed under Job Family j.

$n(n + 1)^m$ is an upper bound on the number of different values the first component of the state vector can take. The buffer level for each Job Family j can vary from $0\ to\ n_j$, and $n_j < n$. Each of the remaining components of the state vector can have up to $(n + 1)$ values each, and the total number of states is bounded from above by $n\ (n + 1)^{2m}$. A tighter higher bound on the number of states can be obtained by using the arithmetic mean – geometric mean inequality. Given a positive constant C and the constraints $\sum_{i=0}^{z} x_i = C$ and $x_i \geq 0$ for all i, then $\prod_{i=0}^{z} x_i$ is

maximized if $x_i = C/z$ for all i. Thus, the number of states is maximized if the total number of jobs n is divided evenly among the m job families, and this results in a tighter higher bound of $n\left\lceil nm^{-1}+1\right\rceil^{2m}$ states.

At each state, m additions are performed, and the minimum of $m + 1$ numbers is obtained. If these $m + 1$ values are stored in an array, finding the minimum can be done in $O(m + 1)$ time. The time complexity of the algorithm is then: $O\left((m+1)n\left\lceil nm^{-1}+1\right\rceil^{2m}\right)$. In relation to the input size of the problem, this is not an efficient algorithm, as the computational time increases exponentially with respect to the input size. When $m = 1$, the time complexity of the algorithm becomes: $O(2n(n + 1)^2)\approx O(n^3)$, which is identical to the complexity of the algorithms proposed by [Ahmadi *et al.* 1992] and [Neale and Duenyas 2000]. The time complexity expression also implies that if the number of job families m is held to a constant number, then the problem takes polynomial time to be solved, with respect to n.

We improve on the naïve DP implementation by identifying certain conditions wherein the optimal policy is known.

- Condition A: When $t > r_n$, no more jobs will arrive in the future, and [Uzsoy 1995] has shown that $f(t,\bullet,\bullet,...,\bullet)$ can be obtained by:
    - assigning jobs to batches such that there is at most one partial batch per job family;
    - Assigning a processing time p_k, and a weight w_k equal to the number of jobs in batch k to each batch k;
    - Processing the batches in decreasing order of w_k/p_k.
- Condition B: Assume that the job family j has the shortest processing time among all job families. If the state is $(t,\bullet,q_j(t)\geq Q,...,\bullet)$, then [Neale and Duenyas 2003] has shown that it is optimal to process a full batch of job family j at time t.
- Condition C: At state $(t,\bullet,q_j(t)>1,...,\bullet)$, assume that the next job arrival happens at time instance r_i. If $(r_i - t) \geq$ processing time for Job Family j (p_j), then it is never

optimal to wait for the next job arrival. The proof is via an interchange argument: Suppose we are at a certain state $(t, q_1(t), q_2(t), ..., q_m(t))$ for which Condition C holds true for job family j, with $min(q_j(t), B) = k_j(t)$. Assume that the optimal policy at this state entails waiting for the next job arrival r_i. The minimal optimal completion time of the $k_j(t)$ jobs is $(r_i + p_j)$. Then the optimal total completion time can be reduced by choosing to process $k_j(t)$ jobs belonging to family j at time instance t, which results in $k_j(t)$ jobs having completion time of $(t + p_j)$. The completion time of other jobs cannot be increased by processing the $k_j(t)$ jobs earlier.

Figure 3.1 illustrates how each condition reduces the number of states the DP algorithm has to visit. While Conditions A and B can reduce more states from consideration compared to Condition C at the same height, conditions A and B are more apt to occur further away from the root node, since a significant number of jobs must have already arrived for condition B to occur, while all jobs must have already arrived for Condition A to occur. This limits the potential state reduction capabilities of conditions A and B. In comparison, Condition C can prune branches near or far from the root node. Figure 3.1 shows the extent branches can be pruned using one of the three conditions.

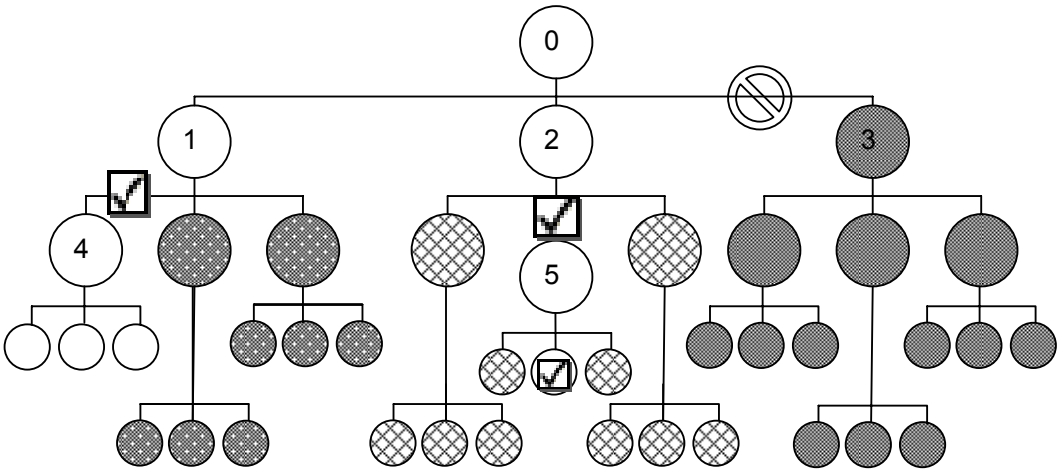


Figure 3.1: A sample 3-ary tree that illustrates node trimming in relation to Conditions A, B and C.

At Node Zero, Condition C is detected, and this causes the branch connecting Node Three to Node Zero to be pruned. At Node One, Condition B is observed, and the optimal decision at Node One is to move to Node Four. At Node Two, Condition A is observed, and the optimal path from Node Two to a leaf of the tree is determined.

At Root Node Zero, Condition C is detected, for Node Three. Consequently, the branch connecting Node Zero to Node Three is pruned. The remaining nodes from Node Zero are Node One and Node Two. At Node One, Condition B is observed, and this allows us to determine that Node four is optimal, and all other nodes that have Node One as a parent (and their subsequent children nodes) can be ignored. At Node Two, Condition A is observed, and the optimal path from Node Two all the way until a leaf of the tree can be determined using the weighted shortest processing time rule.

We also looked into the complexity of a similar scheduling problem, with the added complexity of having more than one batch processor in parallel. Appendix A contains the DP algorithm for the parallel batch processor scheduling problem and its corresponding complexity analysis.

3.3.3.2 Empirical Analysis of Algorithm Complexity

We determine the size of the problem instances that can be solved in a reasonable time. Both the naive and the improved DP algorithm (which incorporates the three pruning conditions) are coded in MATLAB, which is then converted to the programming language C and compiled using MATLAB's built-in C compiler. All experiments are performed on a desktop with a Pentium 4 2.4 GHz processor and 512 MB memory.

We wish to perform a simple comparison of the execution times of the naïve and improved DP algorithms. The batch processor size is set to three, and the processing time per job family is evenly distributed from three to nine time units. The job family is randomly assigned to jobs, using a uniform distribution. At high traffic intensity, the mean job inter-arrival time is two time units, while at low traffic intensity, the mean job inter-arrival time is set at four time units. In both cases, the inter-arrival time is assumed to have a geometric distribution. The number of job families can take three values (3, 4, 5), and the number of jobs processed also takes three values (10, 15, 20). If the resulting job distribution resulted in no jobs getting assigned to a particular job family, the results are discarded and another random job distribution is obtained. For each combination of job family, job quantity and traffic intensity, ten replications are

performed. Table 3.1 contains the summarized results from the experiment performed. CV refers to **Coefficient of Variation**, which is the ratio between the standard deviation and the mean. High values of CV would then imply a large degree of variation between the individual data points.

TABLE 3.1: TIME REQUIRED TO PERFORM NAIVE AND IMPROVED DP ALGORITHMS

No. of job families	Arrival Density	No. of jobs	Mean computational time for naïve DP.	CV for naïve DP	Mean computational time for improved DP.	CV for improved DP	Percentage reduction in computational time
3	High	10	0.86	0.31	0.24	0.56	71.54%
3	High	15	17.98	0.61	1.35	0.47	92.49%
3	High	20	160.05	0.69	10.08	1.04	93.70%
3	Low	10	1.21	0.25	0.24	0.71	80.14%
3	Low	15	24.68	0.41	2.49	1.17	89.93%
3	Low	20	288.51	0.72	11.44	1.13	96.03%
4	High	10	3.04	0.33	0.50	0.79	83.64%
4	High	15	76.62	0.35	12.37	0.43	83.86%
4	High	20	1271.92	0.46	134.78	1.31	89.40%
4	Low	10	4.48	0.42	0.28	0.97	93.82%
4	Low	15	131.39	0.37	8.98	1.31	93.17%
4	Low	20	1858.19	0.23	63.26	1.13	96.60%
5	High	10	11.11	0.39	1.57	0.75	85.86%
5	High	15	285.12	0.38	26.06	0.69	90.86%
5	High	20	6106.14	0.34	245.35	0.58	95.98%
5	Low	10	13.10	0.27	0.97	1.82	92.62%
5	Low	15	569.72	0.22	28.38	1.08	95.02%
5	Low	20	9389.89	0.36	293.33	1.7	96.88%

In general, the percentage reductions in computational time are positively correlated with problem size (in terms of number of jobs). When the number of jobs is small, the overhead related to constantly checking for the pruning conditions would significantly reduce the advantages of using the improved DP algorithm. The computational time reduction is also generally larger when the traffic intensity is low, when Condition C is expected to be most prevalent. This supports the assumption that Condition C can cause a larger reduction in the computational time required.

Since the presence of pruning instances would be dependent on the specific problem instance, the variation in computational time required (as measured by the dimensionless quantity coefficient of variation) is generally higher when the improved DP algorithm is used, in lieu of the naïve DP algorithm.

We also wish to determine empirically the growth of the computational time required with respect to the growth of the number of jobs and the number of job families. We also wish to determine the largest problem that could be solved in a reasonable amount of time, using the improved DP algorithm. Table 3.2 shows the various levels selected for the number of jobs and number of job families.

TABLE 3.2: EXPERIMENTAL SETUP TO DETERMINE ALGORITHM COMPLEXITY

Number of jobs	Number of job families	Traffic intensity	Number of repetitions
10,15,20	3,4,5	High and Low	10

For all experiments, the batch processor size is kept constant at three. The job inter-arrival time is generated from a geometric distribution with a mean of two time units when the traffic intensity is high, and four time units when the traffic intensity is low. Each job is assigned to a job family randomly by using a uniform distribution, and the processing times of job families are also uniformly distributed from three to nine time units. We redistribute jobs to job families if the preceding distribution resulted in a job family having no jobs assigned to it. Table 3.3 contains the mean computational time obtained in seconds.

TABLE 3.3: MEAN COMPUTATIONAL TIME FOR NAÏVE DP ALGORITHM IN SECONDS

	Traffic intensity	3 job families	4 job families	5 job families
10 jobs	High	0.24	0.50	1.57
15 jobs	High	1.35	12.37	26.06
20 jobs	High	10.08	134.78	245.35
10 jobs	Low	0.24	0.28	0.97
15 jobs	Low	2.49	8.98	28.38
20 jobs	Low	11.44	63.26	293.33

3.3.3.3 Discussion of Results

Figure 3.2 plots the computational time growth as the number of job families *m* is increased, while keeping the number of total jobs *n* constant, when the traffic intensity is high. We omit showing the plot for the case where the traffic intensity is low, as the characteristics of the two plots are similar to each other. Due to the large disparity in computational time, it was necessary to plot the axis corresponding to computational time on a logarithmic scale. Although there are insufficient data points to confirm this pattern with a high degree of confidence, the roughly linear plots seen in Figure 3.2 suggest that incrementing the number of jobs causes the computational time required to be multiplied by a factor larger than one. This

implies that the mean computational time grows at an exponential rate with respect to the number of job families.

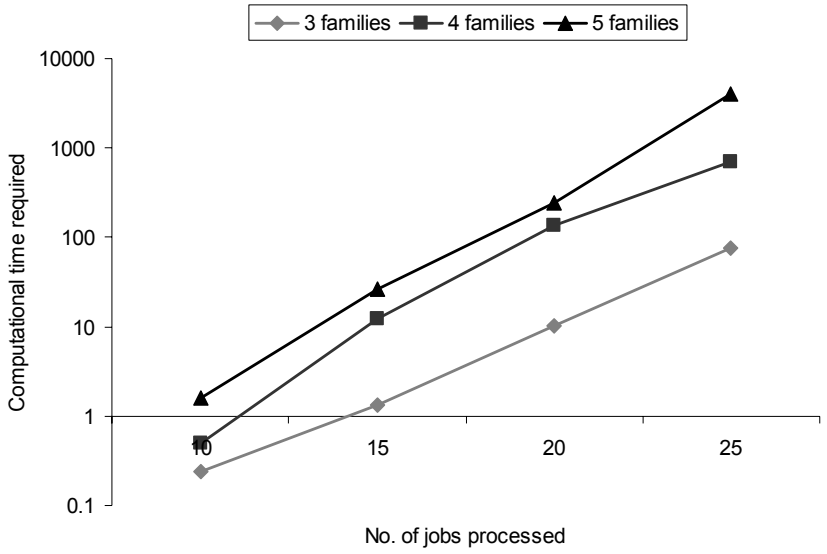


Figure 3.2: Growth of computational time as job family quantity is increased.

The Y-Axis is on a logarithmic scale. The fairly linear lines imply that the growth in computational time is on an order of magnitude, as the job family count gets incremented by one.

Figure 3.3 plots the computational time growth as the number of jobs *n* is increased, while keeping the number of job families *m* constant, when the traffic intensity is high. The computational time growth curves also suggest that the mean computational time increases at an exponential rate, as the number of jobs are increased.

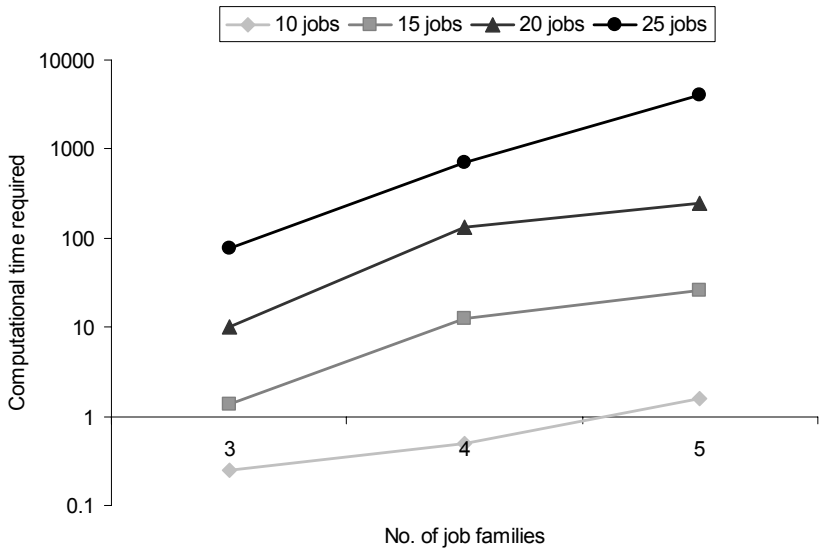


Figure 3.3: Growth of computational time as the number of jobs are increased.

The Y-Axis is on a logarithmic scale. The fairly linear lines imply that the growth in computational time is on an order of magnitude, as the job count gets incremented by five.

3.4 Approximation Method for On-line Batch Processor Scheduling Problem in a Stochastic Environment: Model Predictive Control

The basic tenets of Model Predictive Control (MPC) are described in Section 3.2. In this section we describe the exact MPC implementation to be evaluated.

3.4.1 Description of MPC Algorithm

As an online algorithm, there is a need to set a reasonable time limit for the MPC algorithm to execute. It will be difficult to set an absolute limit, as the execution time is dependent on the problem instance data. This is especially true for this problem, since the variation in computational time required for two problem instances with the same size can be very high. If an absolute limit is required, we can modify the MPC algorithm such that it terminates after a certain amount of time has passed. A simple heuristic can then be used in its place.

From the empirical test conducted in Section 3.3.3.2, it is not only the number of jobs that affects the computational time, but also the number of job families these jobs belong to. Since the number of job families a batch processor can process can easily exceed ten job families, we need to revise the MPC algorithm to reduce the number of job families considered. In this manner, both the state space and the control space are reduced. We propose an MPC-based online algorithm specified by two parameters: the job arrival horizon H (in terms of number of future job arrivals) and the number of job families to consider F.

Let the set of job families selected to be considered be S_f. When there are no more job arrivals, the optimal policy is to process batches in decreasing order of $q_j(t)/p_j$. If the batch processor can process up to χ job families, and computational time considerations only allow us to

consider $\psi < \chi$ job families, then, at time instance t where the batch processor is idle, we select the ψ job families whose batches have the highest positive $q_j(t)/p_j$ ratios to join S_f.

If $|S_f| < \psi$ after this initial selection of job families according to current queue length, we augment S_f by also considering the job families which a large number of job arrivals within the horizon belong to. This is accomplished by looking at the job arrivals within the job arrival horizon, and determining the number of jobs that belong to each job family. Let *f(j)* be the job family with the j^{th} most jobs arriving within the horizon. If $f(j) \notin S_f$, then $S_f = S_f \bigcup f(j)$ and the process is continued until $j = m$, or until $|S_f| = \psi$. While H is the maximum number of jobs that will be passed on to the optimal DP algorithm, when some of the next H jobs belong to job families that are not in S_f, less than H jobs are passed to the optimal DP algorithm.

The manner in which the F job families are selected places more importance on the amount of jobs already in front of the batch processor, rather than on the jobs that will arrive.

3.4.2 Performance of MPC-based Online Algorithm

In this section, we examine the performance of the MPC-based algorithm. A trade-off is required in selecting the number of job families F and the number of future job arrivals H for the MPC-based heuristic *(F, H)*: if the number of job families to consider is increased, the number of future job arrivals should be decreased, if the running time of the heuristic is to be kept roughly constant. We wish to determine the effect of this trade-off and evaluate two variants of the MPC-base heuristic, (3, 15) and (4, 10).

Ideally, we should compare the mean cycle time from the MPC-based algorithm against the mean cycle time from an optimal algorithm. As the problem is NP-hard, only small problem instances can be solved to optimality. Instead, we compare the MPC-based algorithm with a well-known look-ahead heuristic, which is the Next Arrival Control Heuristic for multiple job families (NACHM), as proposed by [Fowler *et al.* 1992].The next arrival control heuristic is a look-ahead heuristic that assumes knowledge of the arrival time of the next job belonging to

each job family, and uses this information to determine whether or not a batch should be processed. We choose to use NACHM as a benchmark for several reasons:

- The logic behind NACHM is easy to understand and implement. This makes NACHM an attractive methodology for industry.
- The manner in which NACHM is different from the way the MPC-based heuristic selects its horizon. If n is the number of job families, then NACHM assumes knowledge of n job arrivals, with one job arrival from each job family. In contrast, the MPC-based algorithm takes, at most, the next H job arrivals into consideration. These H job arrivals can come from a single family.
- NACHM considers all job families in making a decision. In contrast, the MPC-based algorithm may consider only a subset of the set of job families.
- Like other look-ahead methods, NACHM seeks to optimize only the timing of the first batch to be processed, while the MPC-based heuristic seeks to optimize the entire sequence of batches.

3.4.2.1 Experimental Design

The MPC-based algorithm and NACHM were both coded in MATLAB. Two variants of the MPC-based algorithm were evaluated: (3,15) and (4,10). For each simulation run, 2000 jobs are randomly generated, the cycle time from the first 200 jobs to exit the system are discarded; this warm-up period is determined to be sufficient, using Welch's method ([Law and Kelton 2000]). The batch processor size is held at six. The processing time for a particular job family is evenly distributed: the minimum processing time is three time units and the maximum processing time is nine units. The time between job arrivals is assumed to be exponentially distributed; the mean time is determined through the traffic intensity. The traffic intensity is analogous to the utilization rate used to characterize the workload of serial processors, and is equal to the utilization rate if the batch processor was constrained to only process full batches. If the traffic intensity is 1.0, then the arrival rate equals the maximum processing rate of the

batch processor, and the expected steady-state queue size becomes unbounded. If λ is the mean arrival rate, then the mean time between arrivals, 1/ λ, is obtained by:

1/ λ = mean time to process a batch/(traffic intensity × batch processor size).

Two values (four and eight) are assumed for the number of job families, and traffic intensity is assumed to be either 0.5 or 0.8. Each set of simulation parameters were repeated ten times. Within a particular set of parameters, the three heuristics have a common problem instance.

Three sets of experiments are performed. The first set of experiments assumes that the arrival times are deterministic, and that jobs are assigned to job families using a uniform distribution. The second set of experiments also assumes deterministic arrival times, but also assumes there is a positive correlation between the job families of two consecutive jobs. This scenario can exist when the processors upstream of the batch processor select jobs so that a batch is easily formed in front of the batch processor.

Correlation between the job families of two consecutive jobs is introduced in the following manner:

> Let there be m job families. Assume that a certain job X belong to the set f_h. This means job X belongs to job family h. The *a priori* probability that the job family of X, a_X, is h equals $1/m$, since a_X is uniformly distributed. $P(a_x = h) = 1/m$. Let Y be the job that arrives immediately after X. Let φ be the probability that Y will belong to the same job family as X. $\varphi = P(a_Y = h | a_X = h)$. We assume that the probability that Y will belong to a certain job family g when X belongs to a different Job Family f $P(a_Y = g | a_X = h,\ g \neq h)$ has a uniform distribution across the remaining job families, and is equal to $(1 - \varphi)/(m - 1)$. The probability distribution on a_Y, when information on a_X is unavailable, is identical to the probability distribution of a_X.

$$P(a_Y = i) = \sum_{k=1}^{m} P(a_X = k) \times P(a_Y = i \mid a_X = k)$$

Since a_X is uniformly distributed, $P(a_X = k) = 1/m$.

$$P(a_Y = i) = \sum_{k=1}^{m} \frac{1}{m} \times P(a_Y = i \mid a_X = k)$$

$$P(a_Y = i) = \frac{1}{m} \times \left[\varphi + \sum_{k=1, k \neq i}^{m} P(a_Y = i \mid a_X = k) \right]$$

$$P(a_Y = i) = \frac{1}{m} \times \left[\varphi + \sum_{k=1, k \neq i}^{m} \frac{(1-\varphi)}{m-1} \right]$$

$$P(a_Y = i) = \frac{1}{m} \times \left[\varphi + \frac{(1-\varphi) \times (m-1)}{m-1} \right]$$

$$P(a_Y = i) = \frac{1}{m}$$

The correlation coefficient between the job families of two consecutive jobs (represented as random variables *X* and *Y*) can be related to φ and *m* by:

$$Corr(X,Y) = \frac{E[XY] - E[X]E[Y]}{\sqrt{Var(X)Var(Y)}} = \frac{E[XY] - (E[X])^2}{Var(X)} = \frac{12\left(E[XY] - \frac{(m+1)^2}{4} \right)}{(m^2 - 1)}$$

where

$$E[XY] = \sum_{i=1}^{m} i^2 \frac{\varphi}{m} + \sum_{j=1}^{m} \sum_{k=1, k \neq j}^{m} mk \frac{1-\varphi}{m(m-1)}$$

$$E[XY] = \sum_{i=1}^{m} i^2 \left(\frac{\varphi}{m} - \frac{1-\varphi}{m(m-1)} \right) + \sum_{j=1}^{m} \sum_{k=1}^{m} jk \frac{1-\varphi}{m(m-1)}$$

$$E[XY] = \frac{m\varphi - 1}{m(m-1)} \sum_{i=1}^{m} i^2 + \frac{1-\varphi}{m(m-1)} \sum_{j=1}^{J} \sum_{k=1}^{J} jk$$

$$E[XY] == \frac{m\varphi - 1}{m(m-1)} \sum_{i=1}^{m} i^2 + \frac{1-\varphi}{m(m-1)} \sum_{j=1}^{m} j \frac{m(m+1)}{2}$$

$$E[XY] == \frac{m\varphi - 1}{m(m-1)} \sum_{i=1}^{m} i^2 + \frac{(1-\varphi)m(m+1)^2}{4(m-1)}$$

$$E[XY] = \frac{(m\varphi - 1)(m+1)(2m+1)}{6(m-1)} + \frac{(1-\varphi)m(m+1)^2}{4(m-1)}$$

$$Corr(X,Y) = \frac{12\left(\frac{(m\varphi - 1)(m+1)(2m+1)}{6(m-1)} + \frac{(1-\varphi)m(m+1)^2}{4(m-1)} - \frac{(m+1)^2}{4} \right)}{(m^2 - 1)}$$

$$Corr(X,Y) == \frac{2(m\varphi - 1)(m+1)(2m+1) + 3\left((1-\varphi)m(m+1)^2 - (m-1)(m+1)^2 \right)}{(m^2 - 1)(m-1)}$$

$$Corr(X,Y) == \frac{2(m\varphi - 1)(m+1)(2m+1) + 3\left((m+1)^2 (1 - m\varphi) \right)}{(m^2 - 1)(m-1)}$$

$$Corr(X,Y) = \frac{(m\varphi - 1)(m+1)}{(m^2 - 1)} = \frac{m\varphi - 1}{m-1}$$

Two levels of correlation coefficients were selected: weak correlation (*Corr(X,Y) = 0.25*), and strong correlation (*Corr(X,Y) = 0.7*).

In the third set of experiments, consecutive job arrivals are uncorrelated, but the actual arrival time differs from the predicted arrival time. When the batch processor is part of a lager manufacturing system, stochastic events that occur in stages in front of the batch processor may cause planned future arrivals to arrive earlier or later than expected. This is modeled by assuming that the prediction error is normally distributed with zero mean and variance equal to a parameter Ω × (variance of the arrival time distribution) (Ω is the squared coefficient of variation or SCV. The coefficient of variation is the square root of Ω.). Three values for the error coefficient Ω are tested: 0.1 (corresponding to √0.1 or 0.316 coefficient of variation), 0.3 (corresponding to √0.3 or 0.548 coefficient of variation) and 0.5 (corresponding to √0.5 or 0.707 coefficient of variation). Table 3.4 summarizes the various levels varied for each set of experiments.

TABLE 3.4: EXPERIMENTAL SET-UP SUMMARY FOR EVALUATION OF MPC-BASED HEURISTIC

Experiment Set	Job arrival error parameter Ω	Job family correlation between consecutive jobs	Number of job families	Traffic intensity
1	0	0	4,8	0.5, 0.8
2	0	0.25, 0.7	4,8	0.5, 0.8
3	0.1, 0.3, 0.5	0	4,8	0.5, 0.8

3.4.2.2 Performance Evaluation of Approximate DP Algorithm

In each set of experiments, both NACHM and the MPC-based heuristic receive identical job arrival patterns; this allows us to use a paired T-test to determine whether the hypothesized differences in the mean cycle time of the proposed MPC-based heuristics and NACHM is significantly different from zero at a certain confidence interval. The alternative hypothesis is that the difference between two heuristics is different from zero.

The paired T-test assumes that the mean cycle time difference between two heuristics is normally distributed. If the mean cycle time distribution of both heuristics is normal, then the difference between the mean cycle time of two heuristics will be normally distributed, The assumption of normality of the cycle time distribution of a heuristic is supported by results coming from randomly checking the mean cycle time distributions for nonnormality using the Ryan-Joiner normality test. Furthermore, since a single data point is comprised of the mean

cycle time of 1800 jobs, Central Limit Theorem would dictate that the distribution of the mean cycle time will be normal.

3.4.2.2.1 Deterministic Arrival Times with Uncorrelated Job Families

The summarized results of the simulations performed can be found in Table B- 1. Table 3.5 contains the results of the paired T-test, for each pair of heuristic. Across three policies, mean cycle time is positively correlated with traffic intensity and number of job families. Both observations are consistent with intuition: lower traffic intensities generally mean that the mean queue lengths in front of the batch processor is low, while lower number of job families mean that the potential waste in the batch processor size due to job incompatibility is smaller.

TABLE 3.5: PAIRED T-TEST RESULTS FOR EXPERIMENT ONE: UNCORRELATED JOB ARRIVALS

Traffic intensity	number of job families		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$
0.8	8	point estimate of difference	0.279	-1.350	-1.630
		confidence interval at 95%	No significant difference	(-2.106,-0.594)	(-2.432,-0.828)
		cycle time deviation from NACHM (%)		-4.45%	-5.38%
0.8	4	point estimate of difference	1.395	-0.783	-2.178
		confidence interval at 95%	(1.210,1.580)	(-1.07,-0.495)	(-2.441,-1.914)
		cycle time deviation from NACHM (%)		-4.20%	-11.69%
0.5	8	point estimate of difference	-0.131	-4.701	-4.570
		confidence interval at 95%	no significant difference	(-5.253,-4.149)	(-5.193,-3.947)
		cycle time deviation from NACHM (%)		-16.66%	-16.19%
0.5	4	point estimate of difference	1.969	-0.189	-2.159
		confidence interval at 95%	(1.874,2.065)	(-0.304,-0.074)	(-2.306,-2.012)
		cycle time deviation from NACHM (%)		-1.21%	-13.84%

Comparing the two MPC-based heuristics with each other, (3,15) has significantly larger mean cycle time than (4,10), if the number of job families is small (four). When the number of job families is high, the performance of the two policies are not significantly different from each

other. When there are only four job families being processed by the system, (3,15) will only consider three job families for processing. At a particular time instance *t*, if at least three job families have jobs in front of the batch processor, the job families for which the future job arrivals belong to will not be taken into consideration in selecting the job families to be considered for processing under (3, 15). Assume that Job Families One, Two and Three have a large number of jobs in front of the batch processor, and Job Family Four has few or none. The next three job arrivals belong to Job Family Four, and their arrival times are very close to the current time instance *t*. It would then be possible that waiting for future arrivals is better than processing a batch. Since (3,15) effectively ignores the existence of Job Family Four, in this scenario (3,15) might recommend processing a batch instead. It is this possibility that causes (3,15) to generate significantly worse cycle times than (4,10) when there are only four job families.

When there are eight job families being processed by the system, the effect of one less job family considered by the heuristic is reduced. Firstly, the probability of a string of consecutive job arrivals belonging to the same job family is reduced with a larger number of job families. Secondly, the difference in the proportion of job families ignored by (3,15) from what is ignored by (4,10) is reduced; when the number of job families is four, (3,15) ignores 25% of the job families, compared to 0% for (4,10). When the number of job families is eight, (3,15) ignores 62.5%, compared to 50% for (4,10).

Comparing NACHM against both MPC-based heuristics, the mean cycle time for NACHM is significantly larger than either MPC-based heuristic for almost all combinations of parameters. The results suggest that the MPC-based heuristic outperforms NACHM when the arrival times are deterministic, and the job families of consecutive job arrivals are uncorrelated. These results also suggest some guidelines in selecting the parameters *H* and *F* for the MPC-based heuristic:

- If the number of job families is relatively small, it is advisable to select *F* to be equal to the number of job families. A reduction in *H* may be required to reduce the

expected computational time, or an increase in the computational time threshold per iteration may be warranted.

- If the number of job families is high, then F is less than the number of job families, and the choice between longer H and smaller F or shorter H and longer F is less important.

The current selection process for the set S_f places a large emphasis on the jobs in front of the batch processor. One possible improvement is to consider a minimum batch size. This minimum batch size could be used to guarantee stability of the batch processor queue length, as well as reduce the number of job families selected as part of F, due to the number of jobs at the batch processor buffer. This would increase the probability that there will be job families selected, due to the nature of the job arrivals.

While it is certainly possible that either of the MPC-based heuristics consider a larger amount of information than that used by NACHM to make the respective decision at specific instances, it is unlikely that these instances are the main source of the MPC-based heuristic's relative advantage over NACHM. When the number of job families processed by the system is eight, NACHM will use the arrival times of eight future job arrivals in its analysis. In contrast, (3,15) may use up to 15 future job arrivals, while (4,10) may use up to 10 future job arrivals. If we can show that the mean number of future job arrivals considered by either MPC-based heuristic is not significantly higher than the number of future job arrivals considered by NACHM, then the observed cycle time reduction cannot be wholly attributed to the possibly longer horizon length of the MPC-based heuristics.

The selection of the job families to be considered by the MPC-based heuristics is primarily based on the current WIP level seen at the batch processor. Thus, if there are at least three non-empty buffers in front of the batch processor, (3, 15) will not use the future arrival information in determining which job families to consider. Under these circumstances, we can obtain the probability distribution of the actual number of jobs considered by (3,15) in its analysis, given that any of the 15 future job arrivals that do not belong to the three job families are not included in the analysis. The probability that (3,15) will consider more than eight future job

arrivals is only 6.5%, and the mean number of future job arrivals considered by (3,15) is only 5.625 jobs, lower than the eight jobs considered by NACHM. Similarly, the probability that (4,10) will consider more than eight future job arrivals is only 1%, and the mean number of future job arrivals considered is five. This makes it unlikely that the improvements we observed are mainly due to the possibly longer horizon length of the MPC-based heuristics.

To further test the hypothesis that the observed cycle time reduction the MPC-based heuristic has over NACHM cannot be wholly attributed to the difference in horizon length, we perform a small experiment involving two traffic intensities (0.5 and 0.8). We evaluate the performance of the heuristic (4, 8) versus NACHM when there are eight job families, and perform ten simulation runs for each traffic intensity. In this experiment, there does not exist an instance where (4, 8) will use more information about future job arrivals than NACHM. When the traffic intensity is low, the mean cycle time for (4, 8) is 23.58 time units, while the mean cycle time for NACHM is 28.39 time units. A comparison using paired T- test reveals that the estimated cycle time reduction is significant at 95% confidence level. When the traffic intensity is high, the mean cycle time for (4, 8) is 29.32 time units, while the mean cycle time for NACHM is 30.92 time units. The estimated cycle time reduction is also significant at 95% confidence level. Table 3.6 contains the values obtained from the experiment. This experiment shows the MPC-based heuristic results in lower mean cycle time than NACHM even if the horizon length of the MPC-based heuristic is never longer than that of NACHM.

TABLE 3.6: COMPARING (4, 8) WITH NACHM WHEN THERE ARE EIGHT FAMILIES

Traffic Intensity	Mean of (4, 8)	Mean of NACHM	Estimated difference	95% Confidence interval
0.5	23.58	28.39	4.81	(4.35, 5.37)
0.8	29.32	30.92	1.6	(0.72, 2.48)

When there is at least one full batch queuing in front of the batch processor, NACHM automatically processes a full batch. This policy may not be optimal when the processing time of each job family is not identical. Furthermore, the manner in which the horizon is defined between the two policies is different, as shown in Figure 3.4. The MPC-based heuristic uses the information for the next X job arrivals, regardless of their job family, whereas NACHM

uses the information about the next arrival for each of the *Y* job families, regardless of their order of arrival. Other policies (such as MCR detailed in [Weng and Leachman 1993]) establish their horizon length in the same manner as the MPC-based heuristic, and the advantages of these policies, relative to NACHM, are only minimal, even with a longer horizon length than NACHM. This suggests that the differing definitions of horizon is not the main source of the MPC-based heuristic's capability to reduce cycle time over NACHM.

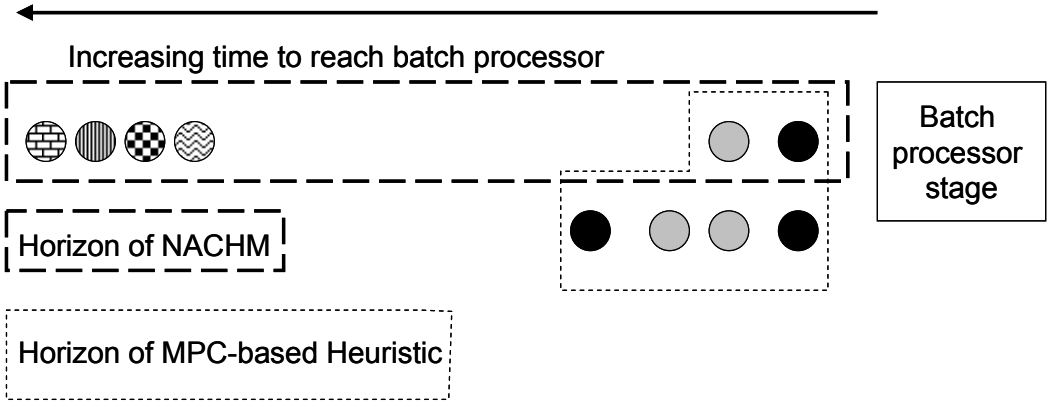


Figure 3.4: Difference in horizon selection between the two policies.

Jobs are denoted by small circles, and differently shaded jobs refer to different job families. At this particular instance, NACHM would not be able to detect the relatively high arrival density experienced by the first two job families.

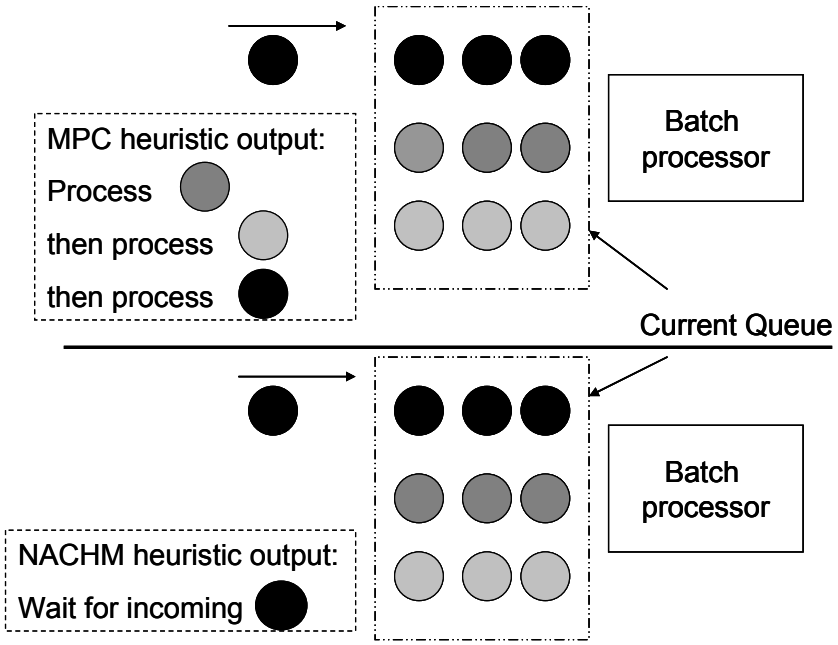


Figure 3.5: Difference between MPC- based heuristic and NACHM.

The MPC-based heuristic will provide a complete sequence of events to empty out the batch processor buffer, although only the first step is actually implemented. For NACHM (and other look-ahead methods), only the best decision regarding the first batch to be processed is considered.

One common aspect of previously proposed look-ahead methods is that these methods aim to optimize only the waiting time incurred due to the next batch. We elaborate through the example shown in Figure 3.5. In this example, the current batch processor queue has jobs from three job families and there must be at least four batches that need to be processed by the batch processor, before the queue is emptied. The MPC-based heuristic will consider the cycle time incurred for processing all three batches, while NACHM (and other look-ahead methods) merely look at the cycle time incurred for the first batch to be processed. We believe this aspect of the newly-proposed heuristic contributes the most towards its relatively improved performance over NACHM.

3.4.2.2.2 Deterministic Arrival Times with Correlated Job Families

The summarized results of the simulations performed can be found in Table B- 2 and Table B-3. Table 3.7 contains the results for the paired T-tests performed for data with the correlation coefficient between two consecutive job arrivals is 0.25, while Table 3.8 contains the results when the correlation coefficient is 0.7. Figure 3.6 illustrates the evolution of mean cycle time under increasing correlation for the three heuristics.

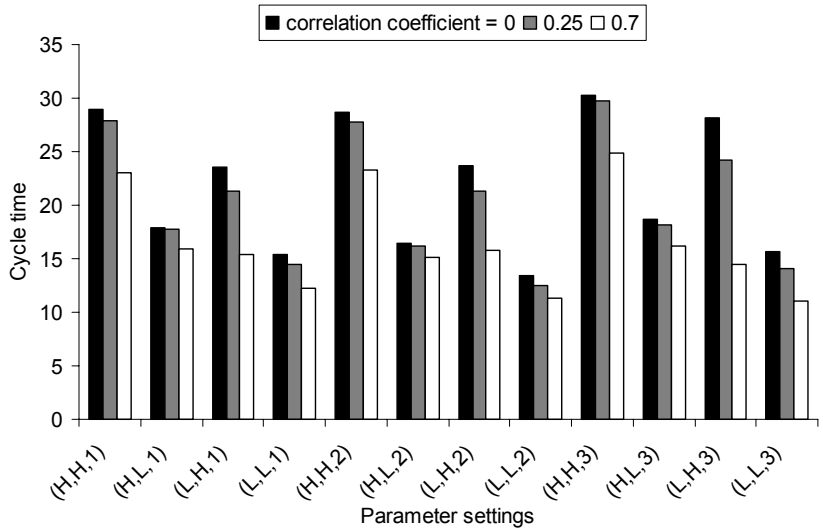


Figure 3.6: Evolution of mean cycle time for the three heuristics upon increase in correlation coefficient between job families of successive job arrivals.

The parameter settings are represented by the triplet (X,Y,Z) where X is the traffic intensity, Y is the number of job families, and Z is the heuristic. H refers to a high setting and L refers to a low setting. Z = 1 refers to (3,15), Z = 2 refers to (4,10) and Z = 3 refers to NACHM.

In comparing (3,15) versus (4,10), the same observations can be made: there is a significant increase in the mean cycle time of the heuristic (3,15) over (4,10) when the number of job families is small. When the number of job families is high, a significant difference between the mean cycle times generated by the two heuristics generally cannot be ascertained within a 95% confidence interval. The only exception is when the correlation coefficient was high, and the traffic intensity was low. High correlation coefficients mean that there is an increased probability that consecutive job arrivals will be from the same job family. When Job Families One, Two and Three are selected by (3,15) due to the present queue length at the batch processor and the next four job arrivals all belong to Job Family Four, the heuristic will think that there are no job arrivals in the near future (since jobs from Family Four are ignored), and will recommend processing a batch, although it might be a better idea to wait for the four jobs from Family Four. Increasing *F* from 3 to 4 increases the probability that some job families would be selected according to future job arrivals.

TABLE 3.7: PAIRED T-TEST RESULTS WHEN CORR(X,Y) = 0.25.

Traffic intensity	number of job families		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$
0.8	8	point estimate of difference	0.137	-1.854	-1.991
		confidence interval at 95%	no significant difference	(-2.52,-1.187)	(-2.59,-1.392)
		cycle time deviation from NACHM (%)		-6.24%	-6.70%
0.8	4	point estimate of difference	1.485	-0.386	-1.871
		confidence interval at 95%	(1.363,1.607)	(-0.634,-0.139)	(-2.060,-1.683)
		cycle time deviation from NACHM (%)		2.13%	10.32%
0.5	8	point estimate of difference	0	-2.830	-2.830
		confidence interval at 95%	no significant difference	(-3.301,-2.359)	(-3.159,-2.501)
		cycle time deviation from NACHM (%)		-11.70%	-11.70%

		point estimate of difference	1.945	0.366	-1.579
		confidence interval at 95%	(1.776,2.114)	(0.048,0.685)	(-1.771,-1.387)
0.5	4	cycle time deviation from NACHM (%)		2.59%	-11.19%

The relative performance of (4,10) (as measured by the percentage reduction in mean cycle time over NACHM) generally worsens as the correlation coefficient is increased. Furthermore, when the traffic intensity is low and the correlation coefficient is high, NACHM outperforms the MPC-based heuristic. This effect is magnified when there are more job families, as some job families will not be considered for processing by the MPC-based heuristic.

TABLE 3.8: PAIRED T-TEST RESULTS WHEN CORR(X,Y) = 0.7.

Traffic intensity	number of job families		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$
		point estimate of difference	-0.366	-1.849	-1.483
		confidence interval at 95%	no significant difference	(-2.669,-1.029)	(-2.537,-0.43)
0.8	8	cycle time deviation from NACHM (%)		-7.44%	-5.97%
		point estimate of difference	0.803	-0.263	-1.066
		confidence interval at 95%	(0.526,1.081)	no significant difference	(-1.343,-0.789)
0.8	4	cycle time deviation from NACHM (%)			-6.60%
		point estimate of difference	-0.31	0.959	1.269
		confidence interval at 95%	(-0.566,-0.054)	(0.69,1.228)	(0.929,1.609)
0.5	8	cycle time deviation from NACHM (%)		6.62%	8.76%
		point estimate of difference	0.839	1.053	0.214
		confidence interval at 95%	(0.654,1.024)	(0.858,1.248)	(0.085,0.343)
0.5	4	cycle time deviation from NACHM (%)		9.47%	1.92%

We hypothesize that the relative performance deterioration of (4,10) over NACHM as correlation coefficients increase can be attributed to the rapid improvement of the efficacy of NACHM as correlation coefficients increase. We support this hypothesis by initially showing that the introduction of correlation reduces the mean cycle time of jobs, regardless of policy used, and then show that NACHM improves at a faster rate than the MPC-based heuristic when positive correlation is introduced.

The introduction of positive correlation in the job families of succeeding jobs seems to lower the mean cycle time of jobs, regardless of the batch processor's control policy. We use a T-test to determine the statistical significance of the observed reduction in mean cycle time. A 95% confidence level is still used. The null hypothesis would be (H_0: $\mu_0 \leq \mu_1$), while the alternative hypothesis is H_1: $\mu_0 > \mu_1$, as we believe that an increase in correlation (from μ_1 to μ_0) in the job families of successive job arrivals will reduce mean cycle time. (In a large majority of the comparisons made, the difference in alternative hypotheses does not matter.) Two sets of comparisons are performed: comparing uncorrelated with lowly correlated job families among successive job arrivals, and comparing lowly correlated with highly correlated job families among successive job arrivals. Table 3.9 contains the estimated reduction in mean cycle time if low correlation between the job families of successive job arrivals were introduced, Table 3.10 contains the estimated incremental reduction in cycle time if the correlation was increased to a high level. In both tables, positive values mean that the increase in correlation resulted in reduced mean cycle time.

TABLE 3.9: ESTIMATED PERCENTAGE REDUCTION IN MEAN CYCLE TIME IF CORRELATION OF JOB FAMILIES OF SUCCESSIVE JOB ARRIVALS ARE INCREASED FROM 0.0 TO 0.25.

Heuristic	Traffic intensity (high = 0.8, low = 0.5)	Number of job families	Estimated percent reduction in mean cycle time	Probability null hypothesis is true
(3, 15)	High	8	3.85%	3.28×10^{-2}
(4, 10)	High	8	3.40%	4.64×10^{-2}
NACHM	High	8	2.02%	2.39×10^{-2}
(3, 15)	High	4	0.62%	0.37
(4, 10)	High	4	1.22%	0.25
NACHM	High	4	2.72%	2.97×10^{-2}
(3, 15)	Low	8	9.20%	1.36×10^{-9}
(4, 10)	Low	8	9.70%	2.9×10^{-8}
NACHM	Low	8	14.29%	3.27×10^{-10}
(3, 15)	Low	4	6.07%	2.49×10^{-8}
(4, 10)	Low	4	6.78%	9.21×10^{-9}
NACHM	Low	4	9.56%	1.56×10^{-11}

The test results show that there is generally a significant reduction in the mean cycle time when correlation between the job families of successive job arrivals is increased. The only cases where the T-test failed to conclude that the mean cycle time was reduced was when the number of job families were low, the traffic intensity was high, and the correlation was increased from 0 to 0.25, for the two MPC-based heuristics. In general, the reduction in mean cycle time is greater when the number of job families is high; the larger the number of job families, the greater the significance of successive job arrivals having the same job family.

TABLE 3.10: ESTIMATED PERCENTAGE REDUCTION IN MEAN CYCLE TIME IF CORRELATION OF JOB FAMILIES OF SUCCESSIVE JOB ARRIVALS ARE INCREASED FROM 0.25 TO 0.7.

Heuristic	Traffic intensity (high = 0.8, low = 0.5)	Number of job families	Estimated percent reduction in mean cycle time	Probability null hypothesis is true
(3, 15)	High	8	17.45%	5.7×10^{-8}
(4, 10)	High	8	15.72%	2.1×10^{-6}
NACHM	High	8	16.37%	1.59×10^{-7}
(3, 15)	High	4	10.40%	7.76×10^{-5}
(4, 10)	High	4	7.16%	2.56×10^{-3}
NACHM	High	4	10.86%	1.71×10^{-5}
(3, 15)	Low	8	27.71%	1.08×10^{-15}
(4, 10)	Low	8	26.24%	5.74×10^{-14}
NACHM	Low	8	40.12%	3.19×10^{-17}
(3, 15)	Low	4	15.86%	1.04×10^{-10}
(4, 10)	Low	4	9.50%	2.98×10^{-8}
NACHM	Low	4	21.14%	4.83×10^{-15}

The results also suggest that NACHM will benefit more than the MPC-based heuristic from the increase in correlation. The introduction of correlation causes job arrivals for a particular job family to arrive in groups. We believe this has two effects: first, it alleviates the problem of estimating individual job family arrival rates when NACHM selects its job horizon, since it becomes more likely to see a succession of job arrivals with the same job family. Since NACHM only knows the first job arrival for each job family, it does not have information on the relative frequency of job arrivals. Secondly, because job families of future arrivals are positively correlated, the loss in solution quality due to NACHM optimizing only the first batch to be processed is reduced. If the next X arrivals all belong to Job Family j, then it is likely that Job Family j would be the dominant factor in determining the appropriate batch processor control, whether we optimize just the first batch, or optimize the emptying of the queue. This effect is more pronounced when the traffic intensity is low, since the time

instances between jobs are long. At high correlation coefficients and low traffic intensity, NACHM becomes superior to the MPC-based heuristics.

In comparing the two MPC-based heuristics, results suggest that (3,15) benefits more than (4,10) from introduced correlation, particularly when there are only four job families. When the correlations are low, the current WIP levels seen at the batch processor (which is a function of past job arrivals) are less representative of the future job arrivals expected, than when the correlations are high. Since the MPC-based heuristic uses the current WIP levels to determine which job family to ignore, high correlation benefits (3,15) more than (4,10), as it makes (3,15) less likely to ignore a crucial job family.

If it is possible to increase the job family correlation of successive job arrivals significantly, then the reduction in cycle time obtained through this method is generally larger than the benefit of using a more sophisticated control policy. This suggests that controlling the arrival pattern of jobs into the batch processor can conceivably result in larger improvement in the batch processor performance, even with a simplistic batch processor control policy.

3.4.2.2.3 Stochastic Arrival Times with Uncorrelated Job Families

The summarized results of the simulations performed can be found in Tables B-4, B-5 and B-6. Figure 3.7 illustrates the effect of prediction errors on the mean cycle time for various heuristics. From Figure 3.7, the effect of prediction errors, over the range of coefficient values evaluated, is small. This suggests making one decision at a time (a characteristic all three evaluated policies share) is sufficient to impart a large degree of robustness into a policy.

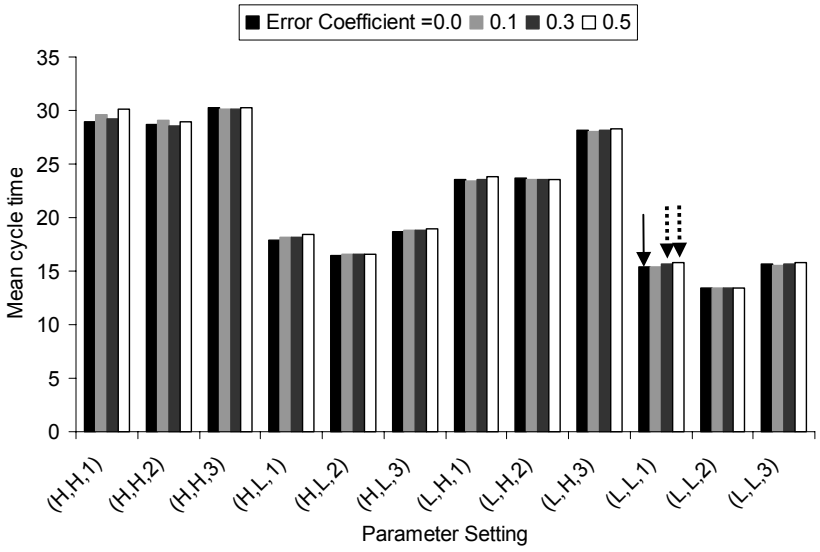


Figure 3.7: Evolution of mean cycle time as the error coefficient is increased.

The parameter settings are represented by (X,Y,Z) where X is the traffic intensity, Y is the number of job families, and Z is the heuristic. H refers to high setting and L refers to low setting. Z = 1 refers to (3,15), Z = 2 refers to (4,10) and Z = 3 refers to NACHM. Dotted arrows are only instances a significant difference in mean cycle time distribution is detected over the base case (solid arrow).

There are only two parameter settings where a significant difference can be detected (at 95% confidence level) between the mean cycle time distribution for perfect information, and that of imperfect information, using 2-sample T-tests (assuming unequal variances). These are when the policy is (3,15), the number of families are low, the traffic intensity is low, and the error coefficient is either 0.3 or 0.5. When the error coefficient is 0.3, the estimated deterioration in mean cycle time is 0.243 (with confidence interval of (0.081, 0.404), while the estimated deterioration in mean cycle time when error coefficient is 0.5 is 0.408 (with confidence interval of (0.236, 0.580)). It is not surprising that (3,15) is the policy that is most sensitive to prediction errors, as (3,15) sacrifices perfect state information (current job queues) in favor of imperfect future information (job arrival horizon). As discussed in Section 3.4.2.2.1, (3,15) performs poorly when both the traffic intensity and the number of job families is low, and thus it is intuitive to observe performance deterioration under these settings due to wrong arrival times.

TABLE 3.11 PAIRED T-TEST RESULTS WHEN ERROR COEFFICIENT $\Omega = 0.1$.

Traffic	No. of job		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$

intensity	families				
		point estimate	0.474	-0.630	-1.105
		confidence interval at 95%	no significant difference	no significant difference	(-1.753,-0.456)
0.8	8	cycle time deviation (%)			-3.66%
		point estimate	1.534	-0.657	-2.191
		confidence interval at 95%	(1.262, 1.806)	(-1.132,-0.182)	(-2.497,-1.885)
0.8	4	cycle time deviation (%)		-3.50%	-11.68%
		point estimate	-0.119	-4.569	-4.450
		confidence interval at 95%	No significant difference	(-5.006,-4.132)	(-4.851,-4.049)
0.5	8	cycle time deviation (%)		-16.31%	-15.89%
		point estimate	2.012	-0.116	-2.128
		confidence interval at 95%	(1.869, 2.154)	No significant difference	(-2.249, -2.006)
0.5	4	cycle time deviation (%)			-13.70%

We also compare relative performance of the policies under imperfect information. Table 3.11, Table 3.12 and Table 3.13 contain the summarized results of the paired T-Tests performed among the set of heuristics. Table 3.11 contains the results when the predicted arrival time error coefficient Ω is 0.1 (coefficient of variation CV = 0.316), Table 3.12 contains the results when Ω is 0.3 (CV = 0.548), and Table 3.13 contains the results when Ω is 0.5 (CV = 0.707). Cycle time deviation is computed as a percentage from the cycle time generated by NACHM. The three error coefficients are selected to observe the robustness of the behavior under low, medium and high levels of uncertainty. While it is still true that the general order of the heuristics in tems of solution quality are identical whether arrival information is perfect or not, it remains to be seen whether the performance of some heuristics deteriorate faster than others under the influence of arrival prediction errors.

TABLE 3.12: PAIRED T-TEST RESULTS WHEN ERROR COEFFICIENT Ω = 0.3.

Traffic intensity	number of job families		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$
		point estimate	0.677	-0.863	-1.539
0.8	8	confidence interval at 95%	(0.146,1.208)	(-1.586,-0.139)	(-2.294,-0.785)

		cycle time deviation (%)		-2.86%	-5.11%
0.8	4	point estimate	1.597	-0.619	-2.215
		confidence interval at 95%	(1.419,1.774)	(-0.845,-0.392)	(-2.454,-1.976)
		cycle time deviation (%)		-11.79%	-3.29%
0.5	8	point estimate	-0.078	-4.576	-4.496
		confidence interval at 95%	No significant difference	(-4.959,-4.192)	(-4.805,-4.186)
		cycle time deviation (%)		-15.99%	-16.28%
0.5	4	point estimate	2.247	-0.016	-2.263
		confidence interval at 95%	(2.107,2.386)	No significant difference	(-2.375,-2.150)
		cycle time deviation (%)			-14.44%

To determine whether certain heuristics have deteriorating performances relative to the performance of other heuristics, we use a two-sample test (assuming unequal variances) to evaluate whether the distribution of the differences between the mean cycle time of two heuristics are identical, despite changes in the error coefficient. Thus, the null hypothesis is of the form: $(\mu_{heuristic1} - \mu_{heuristic2})_{\Omega = 0} = (\mu_{heuristic1} - \mu_{heuristic2})_{\Omega = x}$, where x can be 0.1, 0.3 or 0.5. There are only few instances where a significant difference between the relative performances of policies under differing error coefficients can be detected, at 95% confidence level. These settings, and the corresponding estimates and confidence intervals, are detailed in Table 3.14.

TABLE 3.13: PAIRED T-TEST RESULTS WHEN ERROR COEFFICIENT $\Omega = 0.5$.

Traffic intensity	number of job families		$\mu_{(3,15)} - \mu_{(4,10)}$	$\mu_{(3,15)} - \mu_{NACHM}$	$\mu_{(4,10)} - \mu_{NACHM}$
0.8	8	point estimate of difference	1.287	-0.120	-1.407
		confidence interval at 95%	(0.033,2.541)	No significant difference	(-2.070,-0.745)
		cycle time deviation from NACHM (%)		-4.64%	
0.8	4	point estimate of difference	1.534	-0.657	-2.191
		confidence interval at 95%	(1.262,1.806)	(-1.132,-0.182)	(-2.497,-1.885)
		cycle time deviation from NACHM (%)		-12.15%	-2.20%

		point estimate of difference	0.119	-4.569	-4.450
		confidence interval at 95%	No significant difference	(-6.006,-4.132)	(-4.851,-4.049)
0.5	8	cycle time deviation from NACHM (%)		-16.56%	-15.77%
		point estimate of difference	2.012	0.116	-2.128
		confidence interval at 95%	(1.869,2.154)	No significant difference	(-2.249,-2.006)
0.5	4	cycle time deviation from NACHM (%)			-15.18%

In Table 3.14, a negative estimate for the change in the differences in cycle time means that the performance of the minuend deteriorates with respect to the performance of the subtrahend as the error coefficient is increased. For the first row, the estimated change in ($\mu_{(3,15)}$ - μ_{NACHM}) is negative, and this implies that the performance of (3,15), relative to that of NACHM, deteriorated as the error coefficient was increased from 0.0 to 0.5. The results support the contention that (3,15) is a poor combination of heuristic parameters when the number of job families is low.

Table 3.14 also suggests that when both the traffic intensity and the number of job families are low, the performance of (4,10) slightly improves, relative to NACHM. On the other hand, for the same set of parameters, the performance of (3,15) deteriorates, relative to NACHM, for lower error coefficients. This result highlights the importance of the proper selection of heuristic parameters, especially when errors in arrival times are significant.

TABLE 3.14: SETTINGS FOR WHICH A SIGNIFICANT DIFFERENCE EXISTS IN THE DIFFERENCES BETWEEN MEAN CYCLE TIMES OF TWO POLICIES.

Parameter settings	Error coefficient	Policy pair compared	estimated change	95% confidence level
High traffic intensity, low number of job families	0.0 and 0.5	$\mu_{(3,15)}$ - μ_{NACHM}	-0.485	(-0.815, -0.156)
Low traffic intensity, low number of job families	0.0 and 0.3	$\mu_{(3,15)}$ - $\mu_{(4,10)}$	-0.277	(-0.436,-0.228)
Low traffic intensity, low number of job families	0.0 and 0.5	$\mu_{(3,15)}$ - $\mu_{(4,10)}$	-0.475	(-0.630,-0.321)
Low traffic intensity, low number of job families	0.0 and 0.3	$\mu_{(3,15)}$ - μ_{NACHM}	-0.173	(-0.338,-0.009)
Low traffic intensity, low number of job families	0.0 and 0.5	$\mu_{(4,10)}$ - μ_{NACHM}	0.234	(0.002,0.467)

3.5 Chapter Summary

In this chapter, we examine the problem of minimizing the mean cycle time of a batch processor with incompatible job families and future job arrivals. We provide both ILP and DP formulations for solving small instances of this NP-Hard problem to optimality. When the number of jobs is large, we develop a heuristic based on model predictive control. The computational time of the proposed MPC-based heuristic depends on both F (the number of job families considered) and H (the number of future job arrivals). We determine that when the total number of job families is low, letting H be equal to the number of job families, at the expense of shorter F or longer computational time per iteration, will improve heuristic performance significantly, even with significant errors in the arrival time predictions.

We evaluate the performance of the MPC-based heuristic (4,10) against a popular look-ahead heuristic, NACHM. The MPC-based heuristic has lower mean cycle time than NACHM when the arrival time is deterministic. This difference is more evident when the job arrival rate is low. The observations also hold true when the predicted arrival times are erroneous.

When processors upstream to the batch processor process jobs with the needs of the batch processor in mind, there can be positive correlation between the job families of two consecutive job arrivals. Although the cycle times observed are reduced for all policies evaluated, the relative performance of NACHM, as compared to that of the MPC-based heuristics, improves at a faster rate when correlations between the job families of consecutive job arrivals are increased.

An important by-product of the experimental results has been the discovery that increasing correlation between the job families of consecutive job arrivals can result in significant cycle time reduction for the heuristics evaluated. Furthermore, the magnitude of cycle time reduction observed is generally large, compared to the improvement observed when changing from a simple control policy (NACHM) to a more complex control policy (MPC-based heuristic). This supports the hypothesis that controlling the processor immediately upstream of the batch

processor, to introduce positive correlation, may be more rewarding than using a complicated control policy for the batch processor in isolation.

Another advantage of controlling the upstream processor with the batch processor in mind is in avoiding too many jobs from reaching their processing time windows. This facet of batch processor control is discussed in Chapter 4.

Chapter 4 Control of Job Arrivals into Batch Processor Buffer

4.1 Introduction

In the previous chapter, we have seen how controlling the upstream processor such that the arrival process observed by the downstream batch processor is positively correlated can reduce the time spent by jobs inside the system. We now look at another possible advantage when controlling the upstream processor with the batch processor status in mind.

A frequently ignored facet of controlling diffusion furnaces in wafer fabs is the presence of processing time windows for jobs that are queuing in front of the batch processor. A **processing time window** is a time interval within which a job is deemed to be clean enough for processing at the batch processor. Upon exiting the processor immediately upstream of the batch processor (which we call the feeder processor), a job is deemed to be contaminated if it has waited in front of the batch processor for at least T time units; this job may need to be reprocessed, or validated by process experts before the job can continue to be processed. We refer to this scenario as the job reaching its processing time window at the batch processor. This is unwanted for several reasons. Firstly, a furnace's feeder processor frequently also serves as the feeder processor for a different Processor M. Each job that reaches its processing time window is equivalent to lost throughput for Processor M, without increasing the furnace's

throughput. Secondly, a job that reaches its processing time window incurs the costs of rework, or validation by process engineers. Increased rework or validation also increases the cycle time for at least the expired jobs. Increased cycle time is undesirable, due to the larger yield losses positively correlated with higher cycle times.

Figure 4.1 shows a subsection of a wafer fab process flow involving the furnace and its feeder processors. The furnace can have more than one feeder processor, and the feeder processors typically feed more than one processor. Thus, a feeder processor has to decide whether to process a job meant for the oven, or process a job meant for a different downstream processor, given the possibility that a job processed for the oven might reach its processing time window.

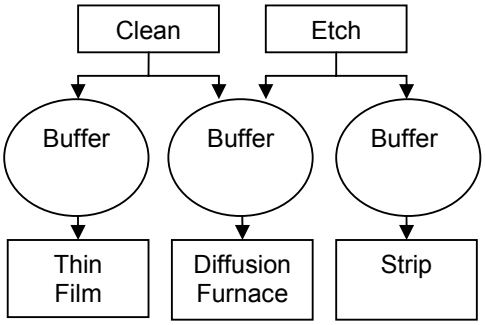


Figure 4.1: A subsection of a process flow inside the wafer fab.

The diffusion furnace is fed work-in-progress by either Clean or Etch. Both Clean (a batch processor) and Etch (a serial processor) can also process jobs to feed other processors. The Clean station can also process jobs for Thin Film, while Etch can also process jobs for Strip.

The control of the feeder processor, in accordance with the existence of processing time windows at the batch processor, is a problem that has been largely ignored by researchers. We address the problem of controlling job flow into the batch processor queue, considering the presence of processing time window present in jobs.

There are other systems in which processing time windows are present. Processing time windows also exist in manufacturing and transport systems where the product is perishable, for example. In this case, when there is a considerable amount of products waiting to be transported, it may be optimal for the manufacturing facility to stop processing products with

finite shelf life. [Makis 1985] looks into the steady-state behavior of a system with a batch processor with infinite size whose jobs have a processing time window, and also determined the optimal policy at the batch processor is a threshold policy. The policy is of the form: process a batch only if at least X jobs are in queue OR if the first job in queue is about to reach its processing time window. Due to the assumption that the batch processor size is infinite, the queue is immediately emptied once a batch is processed.

Jobs processed by wafer fab batch processors typically belong to one of several job families; each job can only be batched together with other jobs belonging to the same job family. This complicates the analysis, as the order in which the prospective job will be processed by the batch processor is sensitive to the batch processor control policy. We make the simplifying assumptions that all jobs belong to the same job family. This allows us to consider only minimum batch size policies, where the batch processor processes a batch only if there are at least MBS = X jobs queued. In particular, we consider two policies, the no-idling policy (minimum batch size is one) and the full-batch policy (minimum batch size is full batch).

4.2 Problem Statement

A two-stage manufacturing system is comprised of an upstream feeder processor feeding a downstream batch processor via a buffer with infinite capacity. The buffer level does not include the jobs that are being processed by the batch processor. In wafer fabrication, the downstream batch processor corresponds to a diffusion oven, while the feeder processor can be either a serial processor (etch station) or a batch processor with size two (clean station). We initially assume that the feeder processor is a serial processor. The batch processor can process up to Q jobs at a time, and its processing time is independent of the number of jobs loaded into the batch processor. We initially assume that upstream of the serial processor is an infinite source of jobs, and downstream of the batch processor is an infinite sink for jobs. Both processors cannot be preempted. See Figure 4.2 for an illustration of the manufacturing system model initially used. The manufacturing system is assumed to operate on a discrete time scale.

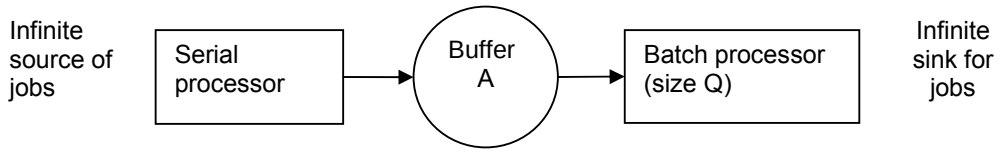


Figure 4.2: Simple 2-stage manufacturing model.

Without a penalty associated with jobs reaching their processing time windows, the optimal policy at the feeder processor (in this case, the feeder processor is initially assumed to be a serial processor) would be to process a job at each instance the feeder processor is available.

There is a processing time window between the batch processor and its feeder processor. For each instance the feeder processor is idle and a prospective job is available at the batch processor, the feeder processor can either decide to process the prospective job or not. If a job has not started processing at the batch processor T time units after it has exited the feeder processor, a penalty C is incurred. This penalty C quantifies lost throughput at the feeder processor, yield loss due to longer cycle time and other ancillary effects of rejecting a job. For each job that the feeder processor processes (including jobs that end up reaching the processing time window), a reward R is generated. This reward is the incremental benefit of processing a job for the batch processor, as compared to feeding a different processor. We wish to determine when it is appropriate to process at the feeder processor, given the number of jobs in front of the batch processor, and the status of the batch processor. We assume $C > R$, otherwise, a profit is generated even when the prospective job is guaranteed to reach its processing time window.

The use of R and C to control the system provides a large degree of flexibility to the methodology. Priority between jobs and between job routes can be controlled by adjusting R and C values, without large amounts of recomputation required.

4.3 Objective Function Discussion

Determining when it is "appropriate" to process a job requires further discussion. An intuitive objective function for this system is to maximize the mean reward per time instance of the

system. The problem can then be modeled as a dynamic programming problem, with the optimal policy obtained via value iteration or policy iteration. However, there are some issues with applying this methodology to the system discussed. In the "real-life" system, deciding not to process a prospective Job *i* at the feeder processor typically means that a possibly different Job *j* is loaded at the feeder processor; Job *j* is destined to reach a different processor buffer (the feeder processor typically also feeds processors other than the batch processor). Additional assumptions about the processing rate of Job *j* would then be needed to determine the transition probabilities of the system. Thus, if the objective function were to maximize the average reward per instance, a better choice would be to explicitly model the three processors (feeder processor, batch processor and the alternative downstream processor of the feeder processor) and their interactions.

As an alternative to maximizing the mean reward per instance, we maximize the expected reward generated by a job processed by the feeder processor, at each time instance the feeder processor is free. In this case, the multi-instance model becomes a single instance model. At each instance, the feeder processor chooses to process a job for a batch processor or not, based solely on the expected reward of the prospective job. Adopting this objective function does not require information on whether an 'idle' feeder processor is actually processing a job for a processor other than the batch processor or not.

An alternative view of the objective function is to consider rewards and penalties attached to a job rather than to a processor. This could be the case for systems that adopt activity-based costing, for example. In this case, each job seeks to maximize its own expected reward, and will only choose to be processed by the feeder processor for the batch processor if the expected reward is positive. Otherwise the prospective job is processed for a processor other than the batch processor.

Throughout this chapter, we maximize the expected reward generated by a job at each instance, and an 'optimal policy' refers to this criterion for optimality. Maximizing the expected reward generated by a job at each instance the feeder processor is free can also be viewed as a greedy heuristic to maximize the mean reward per instance of the feeder processor.

4.4 Problem Methodology

We look at the probability that the prospective job at the serial processor will have to queue for at least T time units in front of the batch processor before it can start processing at the batch processor. Similar to the "news-vendor" model, we obtain expressions for the expected reward of processing a job, and stop processing at the feeder processor when the expected reward to process a job is negative, given the reward R and the possibility of incurring a penalty C. Let R be the reward earned for all jobs that the feeder processor processes, and let $P(X_t)$ be the probability of the prospective job incurring a penalty C given state X_t, then we process jobs if and only if $R > C \times P(X_t)$ or $R/C > P(X_t)$.

An implicit assumption in this analysis is that the batch processor is not lightly utilized. Otherwise, the batch processor is frequently empty when the feeder processor finishes a job, and a job will rarely have to queue in front of the batch processor.

4.4.1 Batch Processor Control Policies Evaluated

To determine whether the optimal feeder processor policy is sensitive to the policy at the batch processor, we assume that the batch processor is operated according to two policies:

a.) **no-idling policy** – If there are jobs waiting in front of the batch processor, the batch processor will immediately start a batch, regardless of the number of jobs that can be processed. This is equivalent to MBS = 1.

b.) **full-batch policy** –If a full batch can be processed, the batch processor will process a full batch. Otherwise, the batch processor waits for additional jobs. This is equivalent to MBS = Q.

Analysis of the problem when the batch processor is under full-batch policy entails estimating when additional jobs after the prospective job is processed will arrive. We assume the feeder processor will continue to process jobs until it is told to stop, which simplifies the problem of estimating the amoung of time for future jobs to arrive at the batch processor buffer.

4.5 Notation used

We use the following notation throughout the first three models.

R	Reward for serial processor processing a job
C	Penalty incurred by a job reaching its processing time window
T	processing time window length
P_S	Probability of a serial processor finishing a job within a time instance, if processor is busy.
P_B	Probability of a batch processor finishing a job within a time instance, if processor is busy and does not fail.
Z_t	Number of jobs inside the batch processor buffer at time instance t
S_t	Status of the batch processor at time instance t. Let $S_t = 1$ mean the batch processor is busy, $S_t = 0$ mean the batch processor is idle and $S_t = -1$ mean the batch processor is down.
F_t	Status of the serial processor at time instance t. Let $F_t = 1$ mean the serial processor is busy and $F_t = 0$ mean the serial processor is idle
X_t	System state at time instance t
Q	Batch processor size

Model four discusses a system that is different from the first three systems. In addition to what has been previously defined, the following notation is used for Model four.

P_A	Probability of a job arrival into the serial processor buffer within a time instance.
$P(i)$	Probability of i arrivals into the serial processor buffer while the serial processor is processing prospective job
$P_{(i)}$	Probability of prospective job incurring a penalty if there were i arrivals into the serial processor buffer while the prospective job is being processed at the serial processor.
Y_t	Number of jobs inside the serial processor buffer at time instance t
B_t	Number of additional job arrivals the prospective job needs to form a full batch

4.6 Model One Assumptions and Development

A series of models with increasing complexity were analyzed. Figure 4.3 highlights the main differences between the four models to be investigated. Model One assumes that the only source of variability in the system is the processing time of the batch processor. The serial processor has constant unit time, with the batch processor processing time effectively an integer multiple of the serial processor processing time (due to the assumption of a discrete time scale). This assumption is appropriate when the serial processor is very fast compared to the batch processor, as seen in wafer fabs.

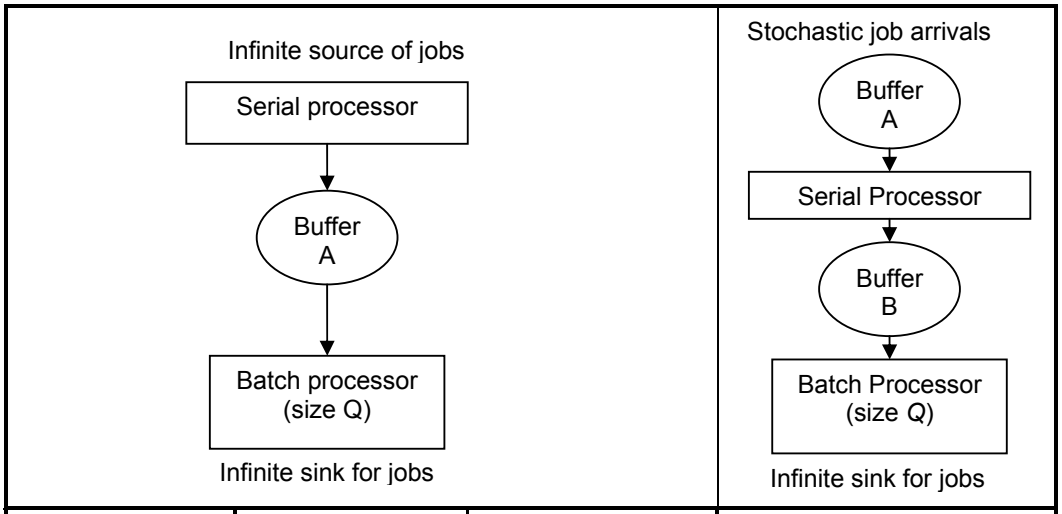


Figure 4.3: Summarizing the differences between the four models investigated.

4.6.1 Model One Assumptions

The following assumptions are made in Model One:

- Both the serial and the batch processor are perfectly reliable
- Serial processor processing time is deterministic. It takes one time unit to process one job at the serial processor.
- Batch processor processing time is geometrically distributed.

4.6.2 Model One Development

At any time instance t, the state of the system X_t can be described as $X_t = (Z_t, S_t)$. Due to the assumptions made for this particular scenario, S_t plays a role only when:

- $Z_t = 0$ when the batch processor is under no-idling policy, and
- $Z_t < Q$ when the batch processor is under full-batch policy.

For all other states, the batch processor will be busy and S_t will be one. The serial processor takes one time unit to process a job. We can omit tracking F_t, the state of the serial processor, as the serial processor is guaranteed to be available at the start of each time instance.

Table 4.1 provides an outline as to how analysis of Model One is segregated.

TABLE 4.1: DIVISION OF ANALYSIS FOR MODEL ONE

Batch processor policy	Segregation of state space
No-idling	Case one: processor idle
	Case two: less than one full batch available at buffer
	Case three: at least one full batch available at buffer
Full-batch	Case one: processor idle
	Case two: less than one full batch available at buffer
	Case three: at least one full batch available at buffer

4.6.2.1 When the Batch Processor is under No-Idling Policy

We divide the model development into three different cases, based on the state of the system.

4.6.2.1.1 Case One: $X_t = (0, 0)$

In Case one, the batch processor is idle, and its buffer is empty. Once the prospective job enters the batch processor queue, it is immediately processed by the batch processor. In this case, the prospective job is guaranteed to earn the full reward R.

4.6.2.1.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$

In Case two, the batch processor is busy, and there is less than one full batch already waiting in front of the batch processor. The only way the prospective job reaches its processing time window is if the batch processor takes a long time to process the current batch. We process the prospective job if and only if $R > C \times$ Probability (current batch will take at least $T + 1$ units to process). We check for the probability that a sequence of events will take at least $T + 1$ time units, since the prospective job will take one time unit to enter the batch processor queue, and the job's processing time window T only starts when the job enters the batch processor queue.

We process the prospective job if and only if: $R/C > (1-P_B)^T$ (4.1)

4.6.2.1.3 Case Three: $X_t = (> Q\text{-}1, 1)$

In Case Three, at least one full batch is ahead of the prospective job if it joins the batch processor buffer. Let $\lfloor x \rfloor$ be the largest integer less than or equal to x. (Conversely, let $\lceil x \rceil$ be the smallest integer greater than or equal to x.) The number of batches the batch processor needs to process before the prospective job can be loaded is $\lfloor Z_t Q^{-1} \rfloor + 1$. ($\lfloor Z_t Q^{-1} \rfloor$ is the number of full batches ahead of the prospective job at the batch processor buffer, plus the current batch already inside the batch processor.) The prospective job should be processed if and only if: $R > C \times P(\lfloor Z_t Q^{-1} \rfloor + 1$ batches will take at least $T + 1$ units to process). We define the function $G(\lfloor Z_t Q^{-1} \rfloor + 1, x, P_B)$ that returns the probability that $\lfloor Z_t Q^{-1} \rfloor + 1$ batches will be processed in exactly x time units, with batch number $\lfloor Z_t Q^{-1} \rfloor + 1$ finishing at the x^{th} time unit, which has a Pascal distribution.

$$G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B) = ({}^{i-1}C_{\lfloor Z_t Q^{-1} \rfloor})(P_B)^{\lfloor Z_t Q^{-1} \rfloor + 1}(1-P_B)^{i-\lfloor Z_t Q^{-1} \rfloor - 1} \quad (4.2)$$

We sum up the probabilities of batch number $\lfloor Z_t Q^{-1} \rfloor + 1$ processed in exactly x time units, for $\lfloor Z_t Q^{-1} \rfloor + 1 \le x \le T$ to obtain the probability of not incurring a penalty. This probability is subtracted from unity to obtain the probability of incurring a penalty. Let

Let ${}^xC_y = x!/(y!)(x-y)!$, then $R/C > 1 - \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{T} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)$. (4.3)

$T > \lfloor Z_t Q^{-1} \rfloor$ is a necessary condition for the prospective job to have a positive probability of avoiding a penalty. Since the system runs in discrete time, it will take at least $\lfloor Z_t Q^{-1} \rfloor + 1$ time units to process all the batches in front of the prospective job, while the serial processor takes one time unit to process the batch.

4.6.2.2 When the Batch Processor is under Full-Batch Policy

When the batch processor is under full-batch policy, the minimum amount of time a job needs to wait before processing is the amount of time needed to wait for future job arrivals needed by the prospective job to form a full batch. We assume that the processing time window $T > Q - 1$. If $T \leq Q - 1$, then the first job to arrive at the empty queue will reach its processing time window. Given that this may cause the job to be reworked, any subsequent jobs will need to wait for another $Q - 1$ job arrivals, which will cause the subsequent jobs to also reach its processing time windows. This would result in each job reaching its processing time window and being flagged for rework.

We consider three different cases. The cases are similar to the cases established when the batch processor is under no-idling policy. However, the number of states in which the batch processor is idle is higher when the batch processor is under full-batch policy.

4.6.2.2.1 Case One: $X_t = (0 \text{ to } Q - 1, 0)$

In Case One, the batch processor is idle. The prospective job will have queue time equal to the time it takes to form a full batch, which is $Q - 1 - Z_t$. Given the assumption that $T > Q - 1$, then it is always optimal to process under Case one.

4.6.2.2.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$

Given the assumption $T > Q - 1$, the prospective job can incur a penalty only if the current batch takes a long time to finish processing.

Prospective job should be processed if and only if $R/C > (1 - P_B)^T$ (4.4)

4.6.2.2.3 Case Three: $X_t = (> Q\text{-}1, 1)$

Let B_t be the number of additional jobs the prospective job will have to wait for to form a full batch. Let $(Z_t \setminus Q) = Z_t - \lfloor Z_t Q^{-1} \rfloor \times Q$. If $(Z_t \setminus Q) = 0$, then the prospective job starts a new batch, and $B_t = Q - 1$. Otherwise, $B_t = Q - (Z_t \setminus Q) - 1$. Prospective job should be processed if and only if $T \geq B_t$ AND $R > C \times probability(\lfloor Z_t Q^{-1} \rfloor + 1$ batches will take at least $T + 1$

units to process). The latter condition is identical to Case Three when the batch processor is under no-idling policy.

$$R/C > 1 - \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{T} ({}^{i-1}C_{\lfloor Z_t Q^{-1} \rfloor})(P_B)^{\lfloor Z_t Q^{-1} \rfloor + 1}(1-P_B)^{i-\lfloor Z_t Q^{-1} \rfloor - 1} \quad \text{or}$$

$$R/C > 1 - \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{T} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B) \quad (4.5),$$ where $G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)$ has the same definition as shown in Equation (4.2).

4.6.3 Model One Discussion

For Model one, the optimal policy will be identical regardless of whether the batch processor policy is no-idling or full-batch, assuming that $T > Q - 1$. The optimal policy is:

If the batch processor is idle, always process the prospective job (Case One)

Else **If** there is less than a full batch in front of the batch processor and the batch processor is busy, process the prospective job if $R/C > (1-P_B)^T$ (Case Two). If this condition is not met, then no job should wait in front of the batch processor when the batch processor is not idle.

Else there is at least one full batch in front of the batch processor (Case Three). Process the prospective job if: $R/C > 1 - \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{T} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)$.

The optimal policy at the feeder processor is a threshold policy. This is because the probability of a prospective job incurring a queuing time of at least T time units is a monotonically non-decreasing function with respect to the number of jobs in front of the batch processor. Figure 4.4 illustrates a typical graph of the probability of the prospective job incurring a penalty, with respect to the number of jobs in front of the batch processor. The probability of a job incurring a penalty is a step function. If we know that it is profitable to process a job when there are $n \times Q$ jobs in the buffer (this means that the prospective job will end up starting a new batch), then we immediately know that it is also profitable to process a job when the number of jobs in the

buffer are from $n \times Q + 1$ *to* $(n + 1) \times Q - 1$, and we only need to test for profitability for when the buffer is at $(n + 1) \times Q$. This simplifies the search for the threshold value as we only need to test for threshold values that are integer multiples of Q.

Figure 4.4 shows sample results obtained by coding the developed closed-form equations in MATLAB, and running the program for a specific set of parameters. The resulting probability plot is a step-function.

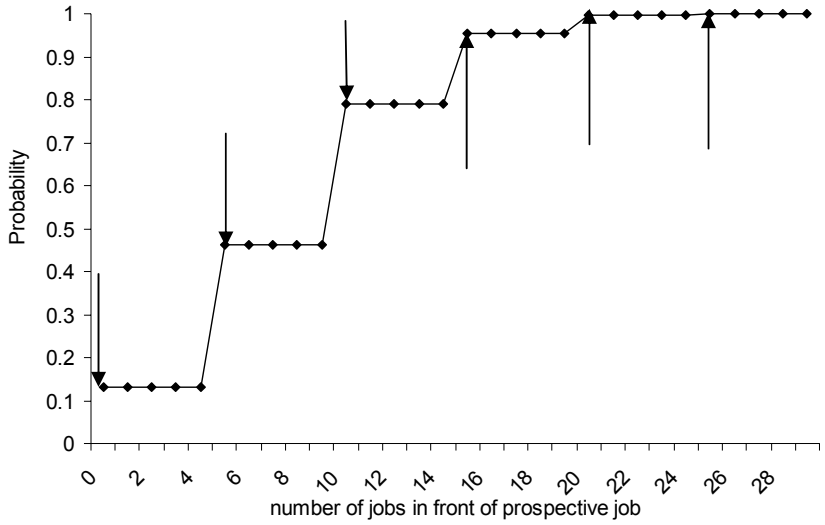


Figure 4.4: Probability of the prospective job incurring a penalty as a function of Z_t when the batch processor is busy and under no-idling policy.

Parameters are $T = 5$, $Q = 5$ and $P_B = 1/3$. The arrows point to the instances when the prospective job will form a new batch. When the job forms a 6th batch queuing in front of the batch processor, the probability of incurring a penalty becomes unity. Since the batch processor is busy, the system will have to process 6 batches in $T = 5$ time instances for the prospective job not to incur a penalty, which is not possible, due to the problem assumptions.

The performance of a system under a threshold policy can be determined by modeling the system as a Discrete Time, Discrete State Markov Chain. At Time Instance t, Let the state of the system be $X_t = (Z_t, S_t)$, with Z_t and S_t as defined in Section 4.6.2. Let L be the threshold value selected, and assume $L > Q$. The transition probabilities when the batch processor is under no-idling policy are detailed in Table C-1 in Appendix C. Figure 4.5 illustrates the Markov Chain for the system under no-idling policy. Once the system leaves state (0, 0), it can not return to (0, 0) anymore. This means that state (0, 0) is transient, and will have a steady state probability of zero.

The only time instances the serial processor is idle are when the buffer is full ($Z_t = L$). To obtain the steady-state throughput rate of the system, we obtain the steady-state probability that $Z_t < L$. To obtain the steady-state mean queue time of jobs at the batch processor buffer, we use the steady-state distribution to obtain the mean number of jobs in the buffer, and invoke Little's Law to obtain the mean queue time of jobs inside the buffer. The state space (including the transient state) is $L + 2$, and the transition probability matrix is of dimension $L + 2$. However, each of the $L + 2$ rows of the transition probability matrix has only two non-zero entries. If L is large, then the transition probability matrix is sparse, and the steady-state distribution can be obtained in $O(L + 2)^2 \sim O(L)^2$ time ([Gershwin 2005]).

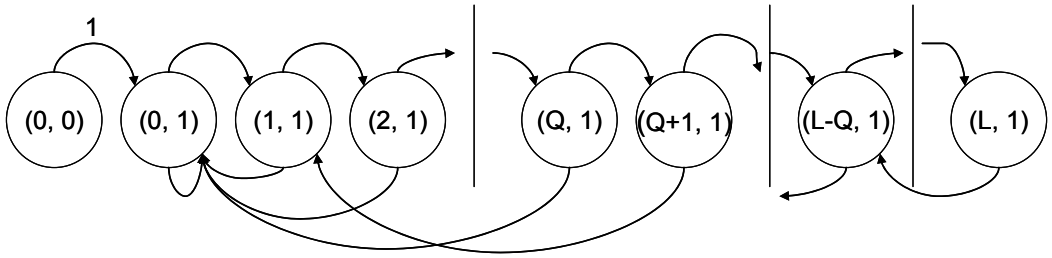


Figure 4.5: Markov Chain of Model One system under threshold policy, with the batch processor under no-idling policy.

All unlabelled transition probabilities at the top of the circles have probability $1\text{-}P_B$ (representing the batch processor not successfully finishing a batch), while all transition probabilities at the bottom of the circle have probability P_B (representing the batch processor finishing a batch). The state (0,0) is a transient state.

Figure 4.6 illustrates the Markov chain of the system with a buffer size limit of L and with the batch processor under full-batch policy.

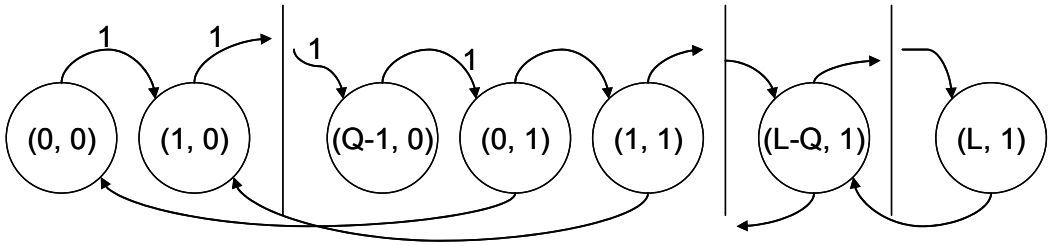


Figure 4.6: Markov Chain of Model One system under threshold policy, with the batch processor under full-batch policy.

All unlabelled transition probabilities at the top of the circles have probability $1-P_B$ (representing the batch processor not successfully finishing a batch), while all transition probabilities at the bottom of the circle have probability P_B (representing the batch processor finishing a batch). There are no transient states.

When the batch processor is under full-batch policy, the state space is $L + Q + 1$, since the number of states the batch processor can be idle is increased from one to Q. Table C-2 in Appendix C contains the transition probabilities for the system operated under a threshold policy when the batch processor is under full-batch policy.

4.7 Model Two Assumptions and Development

The assumption of deterministic serial processor processing time in Model One may be quite restrictive for certain systems. In this section, we discuss the development of a more complex model. The difference between Model One and Model Two is the assumption of a stochastic processing time distribution at the serial processor for Model Two.

4.7.1 Model Two Assumptions

The following assumptions are made in Model Two:

- Both machines perfectly reliable.
- Serial processor processing time is geometrically distributed.
- Batch processor processing time is geometrically distributed.
- Buffer level does not include the jobs being processed by the batch processor

4.7.2 Model Two Development

At a Time Instance t, the state of the system X_t can be described as $X_t = (Z_t, S_t, F_t)$. However, we can only make decisions whenever $F_t = 0$, thus, we only consider the states wherein $F_t = 0$. We will then omit F_t from the notation, and express states as $X_t = (Z_t, S_t)$.

Table 4.2 provides an outline as to how model development for Model Two will proceed.

TABLE 4.2: DIVISION OF ANALYSIS FOR MODEL TWO

Batch processor policy	Segregation of state space	Further subdivision into scenarios
No-idling	Case One: processor idle	
	Case Two: less than one full batch available at buffer	Two different scenarios possible
	Case Three: at least one full batch available at buffer	Case 3.1: Buffer is integer multiple of processor size
		Case 3.2: Buffer is not integer multiple of processor size. Two different scenarios possible
Full-batch	Case One: processor idle	
	Case Two: less than one full batch available at buffer	
	Case Three: at least one full batch available at buffer	

4.7.2.1 When the Batch Processor is under No-Idling Policy

We consider three different cases. The segregation of states into cases is identical to that in Model One, when the batch processor is under no-idling policy.

4.7.2.1.1 Case One: $X_t = (0, 0)$

Once the prospective job enters the queue, it is immediately processed by the batch processor. Thus, the waiting time of the prospective job is always zero, and we always process the job.

4.7.2.1.2 Case Two: $X_t = (0 \text{ to } Q - 1, 1)$

There are two scenarios that could cause the prospective job in the serial processor to incur enough queuing time to incur a penalty.

- **Scenario One**: Current batch at the batch processor finishes processing only after the prospective job reaches its processing time window.

Probability (job will incur penalty due to Scenario One happening) = Probability (current batch still not finished $T - 1$ units after serial processor finishes processing prospective job)

Probability (serial processor finishing in x time units) = $(1 - P_S)^{x-1} P_S$.

Probability (job incurring penalty due to Scenario One happening | serial processor took x time units) = $(1 - P_B)^{x+T-1}$. Probability (job incurring penalty due to Scenario One occurring)

$$= \sum_{x=1}^{\infty} (1-P_S)^{x-1} P_S (1-P_B)^{T+x-1} \text{ or } = P_S (1-P_B)^T \Big/ 1-(1-P_S)(1-P_B) \qquad (4.6)$$

- **Scenario Two**: current batch at the batch processor finishes processing, and loads a new batch before the serial processor finishes processing the prospective job. The newly loaded batch finishes processing only after the prospective job reaches its processing time window. This scenario is not possible if $Z_t = 0$, as there will be no jobs at the batch processor buffer to load into the batch processor.

Probability (serial processor finish processing after x time units) = $(1 - P_S)^{x-1} P_S$

If it took the serial processor x time units to finish the prospective job, the current batch must be processed by the batch processor at the $(x - 1)^{th}$ time unit, at the latest. This ensures that the new batch loaded into the batch processor will not include the prospective job.

Let i be the number of time units the batch processor initially spends unsuccessfully processing the current batch, then the probability (job will incur penalty due to Scenario Two happening | serial processor took x time units) $= \sum_{i=0}^{x-2} (1-P_B)^i P_B (1-P_B)^{T+x-i-2}$

$= (x - 1)(1 - P_B)^{T+x-2} P_B$ *if* $x > 1$, and zero otherwise

Probability (job incurring penalty due to Scenario Two)

$$= \sum_{x=2}^{\infty} (1-P_S)^{x-1} P_S (x-1)(1-P_B)^{T+x-2} P_B$$

$$= \sum_{x=1}^{\infty} (1-P_S)^{x-1} P_S (x-1)(1-P_B)^{T+x-2} P_B$$

$$= P_S P_B (1-P_B)^{T-1} \sum_{x=0}^{\infty} x(1-P_S)^x (1-P_B)^x$$

$$= P_S P_B (1-P_B)^T (1-P_S) \Big/ \left[1-(1-P_S)(1-P_B)\right]^2$$

Since Scenarios One and Two are mutually exclusive, the probability of incurring a penalty when $Z_t > 0$ is the sum of the probability of incurring a penalty under either scenario.

Probability (prospective job incurring a penalty | $Z_t > 0$) under Case Two =

$$\left[P_S(1-P_B)^T\right]\times\left[\frac{1}{\left[1-(1-P_S)(1-P_B)\right]}+\frac{P_B(1-P_S)}{\left[1-(1-P_S)(1-P_B)\right]^2}\right]$$

$$=\left[P_S(1-P_B)^T\right]\times\left[\frac{1-(1-P_S)(1-P_B)+P_B(1-P_S)}{\left[1-(1-P_S)(1-P_B)\right]^2}\right]$$

$$=\left[P_S(1-P_B)^T\right]\times\left[\frac{1-\left[(1-P_S)(1-P_B)-P_B(1-P_S)\right]}{\left[1-(1-P_S)(1-P_B)\right]^2}\right]$$

$$=\left[P_S(1-P_B)^T\right]\times\left[\frac{1-(1-P_S)(1-2P_B)}{\left[1-(1-P_S)(1-P_B)\right]^2}\right]$$

If $Z_t > 0$, we process if and only if $R/C > \left[P_S(1-P_B)^T\right]\times\left[\frac{1-(1-P_S)(1-2P_B)}{\left[1-(1-P_S)(1-P_B)\right]^2}\right]$ (4.7)

If $Z_t = 0$, then only Scenario One can occur, and we process if and only if $R/C > P_S(1-P_B)^T / 1-(1-P_S)(1-P_B)$. (4.8)

4.7.2.1.3 Case Three: $X_t = (> Q - 1, 1)$

The prospective job incurs a penalty when the total time needed to process all the previous batches ahead of it takes at least T time units after the prospective job joins the batch processor queue. The minimum number of batches ahead of the prospective batch is $\lfloor Z_t Q^{-1} \rfloor + 1$. The maximum number of batches ahead of the prospective batch is $\lceil Z_t / Q \rceil + 1$. If $\lfloor Z_t Q^{-1} \rfloor + 1 = Z_t Q^{-1} + 1$, then the prospective job will surely have to form a new batch. This is Case 3.1. Otherwise, two scenarios are possible under Case 3.2, where $\lfloor Z_t Q^{-1} \rfloor + 1 < \lceil Z_t / Q \rceil + 1$:

Scenario One: if $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finish processing before the prospective job joins the queue, then the batch number $\lceil Z_t / Q \rceil + 1$, which contains ($Z_t \backslash Q$) jobs, is loaded into the batch processor. This forces the prospective job to join the next batch instead. Assuming the serial processor finishes at time instance x, the prospective job incurs a penalty when $\lfloor Z_t Q^{-1} \rfloor + 1$ batches are processed before time instance x, but the $\lceil Z_t / Q \rceil + 1^{st}$ batch does not finish until after time instance $x + T - 1$.

Scenario Two: if $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finish processing only after the prospective job joins the queue, then the prospective job gets to join batch number $\lceil Z_t / Q \rceil + 1$. Assuming the serial processor finishes at time instance *x*, the prospective job incurs a penalty when $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finish processing only after time instance $x + T - 1$.

Figure 4.7 illustrates the relationship between Scenarios One and Two. Let the entire rectangle be the set of events that can occur between the time the prospective job is loaded into the feeder processor, and the expiry of its processing time window. This is the universal set **U**. Assume that the feeder processor takes *x* time units to process the prospective job. Let Set C denote the set of events that form the larger ellipse. C is the set of events that cause $\lfloor Z_t Q^{-1} \rfloor + 1$ batches to be finished by $x + T - 1$ time units. Let Set A be the set of events that form the area of the rectangle less C. A=**U**\C. Then the probability of Scenario Two is the probability of being in A. Let Set B denote the set of events that form the smaller ellipse. B is the set of events that cause $\lfloor Z_t Q \rfloor + 1$ batches to be finished by $x - 1$ time units, but the batch number $\lfloor Z_t Q \rfloor + 2$ is not yet finished by $x + T - 1$ time units. B is a subset of C, and the probability of Scenario One is the probability of being in Set B.

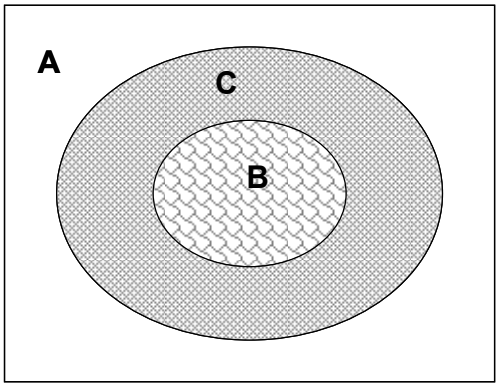


Figure 4.7: Relationship between Scenarios One and Two under Case 3.2.

When the buffer level is an integer multiple of the batch processor size, Set B is a null set, and has probability zero. Thus, set of events that lead to incurring a penalty is Set A, which is the universal set (denoted by the rectangle) less Set C.

Case 3.1 $\lfloor Z_t Q^{-1} \rfloor + 1 = Z_t Q^{-1} + 1$

If it takes *x* steps for the serial processor to finish processing the job, the job gets penalized if $Z_t Q^{-1} + 1$ batches are not finished by *x* + *T* - *1* time units. Let $G(Z_t Q^{-1} + 1, i, P_B)$ be the probability that $Z_t Q^{-1} + 1$ batches are finished in exactly *i* time units, with batch number $Z_t Q^{-1} + 1$ finishing at time unit *i*. ($G(Z_t Q^{-1} + 1, i, P_B)$ is defined in Equation (4.2).) Then the probability of $Z_t Q^{-1} + 1$ batches finishing by *x* + *T* - *1* steps

$$= \sum_{i=Z_t Q^{-1}+1}^{x+T-1} G(Z_t Q^{-1} + 1, i, P_B).$$

A necessary requirement for the prospective job to avoid incurring penalty is $x > Z_t Q^{-1} + 2 - T$. Otherwise, the probability of incurring a penalty is unity. Probability (prospective job incurring penalty):

$$= 1 - \sum_{x=\max(1, Z_t Q^{-1}+2-T)}^{\infty} (1-P_S)^{x-1} P_S \left\{ \sum_{i=Z_t Q^{-1}+1}^{x+T-1} G(Z_t Q^{-1} + 1, i, P_B) \right\}$$

We process a job only if: $R/C > 1 - \sum_{x=\max(1, Z_t Q^{-1}+2-T)}^{\infty} (1-P_S)^{x-1} P_S \left\{ \sum_{i=Z_t Q^{-1}+1}^{x+T-1} G(Z_t Q^{-1} + 1, i, P_B) \right\}$,

or $R/C > 1 - \sum_{x=\max(1, \lfloor Z_t Q^{-1} \rfloor + 2-T)}^{\infty} (1-P_S)^{x-1} P_S \left\{ \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x+T-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B) \right\}$ (4.9)

We use the ratio test to determine whether the series converges. The series converges if:

$$\left| \frac{(1-P_S)^x P_S \sum_{i=Z_t Q^{-1}+1}^{x+T} ({}^{i-1}C_{Z_t Q^{-1}})(1-P_B)^{i-Z_t Q^{-1}-1} (P_B)^{Z_t Q^{-1}+1}}{(1-P_S)^{x-1} P_S \sum_{i=Z_t Q^{-1}+1}^{x+T-1} ({}^{i-1}C_{Z_t Q^{-1}})(1-P_B)^{i-Z_t Q^{-1}-1} (P_B)^{Z_t Q^{-1}+1}} \right| < 1 \text{ for every } x > N$$

$$\left| \frac{(1-P_S)^x P_S \sum_{i=Z_t Q^{-1}+1}^{x+T} ({}^{i-1}C_{Z_t Q^{-1}})(1-P_B)^{i-Z_t Q^{-1}-1} (P_B)^{Z_t Q^{-1}+1}}{(1-P_S)^{x-1} P_S \sum_{i=Z_t Q^{-1}+1}^{x+T-1} ({}^{i-1}C_{Z_t Q^{-1}})(1-P_B)^{i-Z_t Q^{-1}-1} (P_B)^{Z_t Q^{-1}+1}} \right|$$

$$\Rightarrow (1-P_S)\left|\frac{\sum_{i=Z_tQ^{-1}+1}^{x+T}({}^{i-1}C_{Z_tQ^{-1}})(1-P_B)^{i-Z_tQ^{-1}-1}(P_B)^{Z_tQ^{-1}+1}}{\sum_{i=Z_tQ^{-1}+1}^{x+T-1}({}^{i-1}C_{Z_tQ^{-1}})(1-P_B)^{i-Z_tQ^{-1}-1}(P_B)^{Z_tQ^{-1}+1}}\right|$$

$$\Rightarrow (1-P_S)\left|\frac{({}^{x+T-1}C_{Z_tQ^{-1}})(1-P_B)^{x+T-Z_tQ^{-1}-1}(P_B)^{Z_tQ^{-1}+1}}{({}^{x+T-2}C_{Z_tQ^{-1}})(1-P_B)^{x+T-Z_tQ^{-1}-2}(P_B)^{Z_tQ^{-1}+1}}\right|$$

$$\Rightarrow (1-P_S)(1-P_B)\left|\frac{\frac{(x+T-1)!}{(Z_tQ^{-1})!(x+T-1-Z_tQ^{-1})!}}{\frac{(x+T-2)!}{(Z_tQ^{-1})!(x+T-2-Z_tQ^{-1})!}}\right|$$

$$\Rightarrow (1-P_S)(1-P_B)\left|\frac{(x+T-1)!}{(x+T-1-Z_tQ^{-1})!}\times\frac{(x+T-2-Z_tQ^{-1})!}{(x+T-2)!}\right|$$

$$\Rightarrow (1-P_S)(1-P_B)\left|\frac{x+T-1}{x+T-1-Z_tQ^{-1}}\right|<1 \text{ since } \lim_{x\to\infty}\left|\frac{x+T-1}{x+T-1-Z_tQ^{-1}}\right|=1.$$

This rate of convergence is dependent on P_S and P_B.

Let S_j be the summation of the first j terms of the left hand side. Let S^j_∞ be the upper bound for S_∞, and is obtained through the equation: $S^j_\infty = S_j + (1-P_S)^{j+1}$. The term $\sum_{i=Z_tQ^{-1}+1}^{x+T-1}({}^{i-1}C_{Z_tQ^{-1}})(1-P_B)^{i-Z_tQ^{-1}-1}(P_B)^{Z_tQ^{-1}+1}$ is a probability, and is bounded from above by unity. Since the serial processor processing time distribution is geometric, it is possible to determine the probability that the serial processor will take at least $n + 1$ time units to finish processing the prospective job.

$$S_\infty = S_j + \sum_{x=j+1}^{\infty}(1-P_S)^{x-1}P_S\sum_{i=Z_tQ^{-1}+1}^{x+T-1}\left({}^{i-1}C_{Z_tQ^{-1}}\right)(1-P_B)^{i-Z_tQ^{-1}-1}P_B{}^{Z_tQ^{-1}+1} < S_j + (1-P_S)^{j+1},$$

where $(1-P_S)^{j+1} = \sum_{x=j+1}^{\infty}(1-P_S)^{x-1}P_S$. If $1 - S_j < R/C < 1 - S^j_\infty$, we can compute for S_k, for $k > j$, until $1 - S_k < 1 - S^k_\infty < R/C$, or $1 - S_k < 1 - S^k_\infty < R/C$.

Case 3.2 $\lfloor Z_t Q^{-1} \rfloor + 1 < Z_t Q^{-1} + 1$

Scenario One Development:

If it takes *x* steps for the serial processor to finish processing the job, the job gets penalized if: $\lfloor Z_t Q^{-1} \rfloor + 1$ batches are finished by *x* time units but the $\lfloor Z_t Q^{-1} \rfloor + 2^{nd}$ batch is not finished by $x + T - 1$ time units.

Probability of $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finishing by $x - 1$ steps:

$$= \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B) \quad \text{if } x > \lfloor Z_t Q^{-1} \rfloor + 1, \text{ and zero otherwise}$$

Probability of $\lfloor Z_t Q^{-1} \rfloor + 2^{nd}$ batch finishing after $x + T - 1$ steps, conditioned on first $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finished in *i* time units $= (1 - P_B)^{x+T-1-i}$

Probability (job incurring penalty due to Scenario One| serial processor finished in *x* steps)

$$= \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)(1 - P_B)^{x+T-1-i} \text{ if } x > \lfloor Z_t Q^{-1} \rfloor + 1, \text{ and zero otherwise.}$$

Probability (job incurring penalty due to Scenario One)

$$= \sum_{x=\lfloor Z_t Q^{-1} \rfloor + 2}^{\infty} (1 - P_S)^{x-1} P_S \left\{ \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)(1 - P_B)^{x+T-1-i} \right\} \quad (4.10),$$

which can also be expressed as:

$$= \sum_{x=\lfloor Z_t Q^{-1} \rfloor + 2}^{\infty} (1 - P_S)^{x-1} P_S \left\{ \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x-1} ({}^{i-1}C_{\lfloor Z_t Q^{-1} \rfloor})(1 - P_B)^{x+T-\lfloor Z_t Q^{-1} \rfloor - 2} (P_B)^{\lfloor Z_t Q^{-1} \rfloor + 1} \right\}.$$

Using the ratio test, convergence can be shown to hold.

Scenario Two Development

If it takes *x* steps for the serial processor to finish processing the job, the job gets penalized if: $\lfloor Z_t Q^{-1} \rfloor + 1$ batches are not finished by $x + T - 1$ time units.

Probability of $\lfloor Z_t Q^{-1} \rfloor + 1$ batches finishing by $x + T - 1$ steps $= \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x+T-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)$.

$x + T - 1 > \lfloor Z_t Q^{-1} \rfloor + 1$ is a necessary assumption to have a positive probability of avoiding the penalty. Probability (job incurring penalty due to Scenario Two)

$$= 1 - \sum_{x=\max(1,\lfloor Z_t Q^{-1} \rfloor + 2 - T)}^{\infty} (1 - P_S)^{x-1} P_S \left\{ \sum_{i=\lfloor Z_t Q^{-1} \rfloor + 1}^{x+T-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B) \right\} \qquad (4.11)$$

Equation (4.11) is identical to (4.9), the expression derived for the probability of incurring a penalty when we are under Case 3.1. When we are computing for the probabilities of incurring a penalty by incrementing the number of jobs in front of the processor, we can reduce the computational effort by storing the value obtained from (4.9) and reusing it for (4.11). Since Scenarios One and Two are mutually exclusive, the total probability of a prospective job incurring a penalty C under Case 3.2 is the sum of probabilities for Scenario One and Scenario Two. Thus, we should process if and only if: $R/C > (4.10) + (4.11)$

Since (4.10) is always positive, the probability of incurring a penalty is always larger for Case 3.2, compared to Case 3.1, given the same number of full batches in front of the batch processor.

4.7.2.2 When the Batch Processor is under Full-Batch Policy

This section derives the equations used to obtain the optimal policy when the batch processor is under full-batch policy.

4.7.2.2.1 Case One: X_t = (0 to Q - 1)

The waiting time of prospective job = time to have $Q - Z_t - 1$ arrivals needed to form a full batch. Probability of $Q - Z_t - 1$ arrivals taking at least T time units

$$= 1 - \sum_{i=Q-Z_t-1}^{T-1} ({}^{i-1}C_{Q-Z_t-2})(1 - P_S)^{i-Q+Z_t+1} P_S{}^{Q-Z_t-1}$$

if $Q - Z_t - 1 > 0$, and 0 otherwise. This can be represented as $1 - \sum_{i=Q-Z_t-1}^{T-1} G(Q - Z_t - 1, i, P_S)$ if $Q - Z_t - 1 > 0$, and 0 otherwise.

We process if and only if: $R/C > 1 - \sum_{i=Q-Z_t-1}^{T-1} G(Q - Z_t - 1, i, P_S)$ if $Q - Z_t - 1 > 0$, and we always process the prospective job if $Z_t = Q - 1$, as this means that the prospective job will be the last job to join the batch, and will not wait for additional job arrivals.

Similar to the assumption made in Model One to ensure that the first job to arrive at an empty queue has a positive reward, we assume: $R/C > 1 - \sum_{i-Q-1}^{T-1} G(Q-1, i, P_S)$ (4.12).

If this assumption is violated, the very first job to enter the buffer is expected to incur a negative reward while waiting for a full batch to be formed. This assumption is slightly different from that found in Model One. In Model One, the processing time of the serial processor is deterministic and violation of the assumption $T > Q - 1$ means that the very first job to enter the buffer will surely reach its processing time window and incur a penalty. In Model Two, violation of Equation (4.12) means that the very first job to enter the buffer will reach its processing time window a certain proportion of time such that the expected reward of processing the first job is negative.

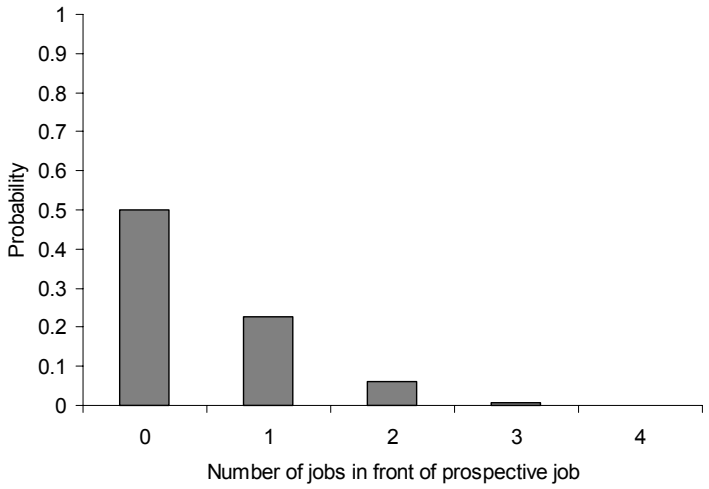


Figure 4.8: Probability of the prospective job incurring a penalty when the batch processor is idle and under full-batch policy.

Parameters: $T = 8$, $Q = 5$, $P_S = 0.5$ and $P_B = 0.25$. When the batch processor is idle, the probability of incurring a penalty is only dependent on future job arrival patterns. This implies that P_B does not affect the probabilities, under Case one.

The probability of incurring a penalty may not be strictly increasing with respect to the number of jobs in front of the batch processor. Under Case One, the probability of incurring a penalty is decreasing with respect to the number of jobs in front of the batch processor, as illustrated in Figure 4.8.

4.7.2.2.2 Case Two: X_t = (0 to Q - 1, 1)

The probability of a prospective job incurring a penalty is the probability that either of two events occur:

- arrival of $Q - Z_t - 1$ jobs takes at least T time units (Event T_A for "tardy arrivals"), and
- processing time of current batch at batch processor is at least T time units more than processing time of prospective job at serial processor (Event T_B for "tardy batch processor")

The two events are independent of each other. Thus, the probability of incurring a penalty $P(C)$ is equal to probability of the first event $P(T_A)$ + the probability of the second event $P(T_B)$ – $P(T_A) \times P(T_B)$.

At time t, let $V_t = \left(Z_t - Q\left\lfloor Z_t Q^{-1} \right\rfloor\right)$. V_t is the number of jobs waiting for a full batch to be formed. Let B_t be the number of additional jobs the prospective job needs to form a full batch. If $V_t = 0$, then the prospective job will start a new batch and it has to wait for $Q - 1$ additional jobs to arrive into the batch processor buffer ($B_t = Q - 1$). Otherwise, $B_t = V_t - 1$.

$$P\left(T_A(X_t)\right) = 1 - \sum_{i=B_t}^{T-1} \left({}^{i-1}C_{B_t-1}\right)(1-P_S)^{i-B_t} P_S^{B_t}, \text{ or } P\left(T_A(X_t)\right) = 1 - \sum_{i=B_t}^{T-1} G(B_t, i, P_S), \text{ if } B_t > 0.$$

This expression generalizes the expression in Case One and is also true for Case Three.

Given $P(T_A(X_t))$, we only need to derive $P(T_B(X_t))$ to obtain the probability of incurring a penalty $P(C(X_t))$, which we compare to R/C to determine whether a job should be processed.

The probability $P(T_B(X_t))$ of the processing time of the current batch taking at least T time units more than the processing time of the prospective job at the serial processor has already been derived under Case Two, when the batch processor is under no-idling policy.

$$P(T_B(X_t)) = \sum_{x=1}^{\infty}(1-P_S)^{x-1}P_S(1-P_B)^{T+x-1}$$
$$P(T_B(X_t)) = P_S(1-P_B)^T \Big/ 1-(1-P_S)(1-P_B) \qquad (4.13).$$

We process if and only if: $R/C > P(T_B(X_t)) + (1-P(T_B(X_t))) \times P(T_A(X_t))$

4.7.2.2.3 Case Three: $X_t = (> Q\text{-}1, 1)$

In Case Three, $Z_t > Q - 1$, and hence batch processor is automatically busy. Let

$c_1 = \max(1, \lfloor Z_t Q^{-1} \rfloor + 2 - T)$,

$P(B(Z_t)) = 1 - \sum_{x=c_1}^{\infty}(1-P_S)^{x-1}P_S\left\{\sum_{i=\lfloor Z_t Q^{-1} \rfloor+1}^{x+T-1} G(\lfloor Z_t Q^{-1} \rfloor + 1, i, P_B)\right\}$. (4.14) This equation has already been derived under Case 3.1 when the batch processor is under no-idling policy.

We process if and only if: $R/C > P(T_B(X_t)) + (1-P(T_B(X_t))) \times P(T_A(X_t))$

4.7.3 Model Two Discussion

We divide the analysis into two segments, depending on the batch processor control policy.

4.7.3.1 When the batch processor is under no-idling policy

When the serial processor has deterministic processing time, the waiting time is an increasing function with respect to the number of full batches in front of the prospective job. This is because the number of batches to be processed ahead of the prospective job is constant. For Model Two, this is no longer true; the number of batches processed ahead of the prospective job may vary. Thus, the probability of incurring a penalty is no longer solely based on the number of full batches ahead of the prospective job. For example, it is possible that the optimal policy instructs us to process a job when batch processor size is 10 and there are 30 jobs in front of the batch processor (which forces the incoming job to form a new batch), but tells us to stop when there are 31 jobs in front of the batch processor. This is counter-intuitive; we would expect that the decision to start a new batch (processing a job when there are 30 jobs

in front of the batch processor) would imply that it is also profitable to fill this batch up to size. From comparing Cases 3.1 and 3.2, this may not always be the case. It is true, however, that a decision to process a job when there are 31 jobs in front of a batch processor, implies that it is also profitable to process enough jobs to form a full batch.

It remains to be seen whether the probability of incurring a penalty is non-decreasing with respect to the number of jobs in front of the batch processor. This is true if:

$$\sum_{x=\max(1,y+2-T)}^{\infty}(1-P_S)^{x-1}\left\{\sum_{i=y+1}^{x+T-1}({}^{i-1}C_y)(1-P_B)^{i-y-1}P_B^{\,y+1}\right\}$$
$$-\sum_{x=\max(1,y+3-T)}^{\infty}(1-P_S)^{x-1}\left\{\sum_{i=y+2}^{x+T-1}({}^{i-1}C_{y+1})(1-P_B)^{i-y-2}P_B^{\,y+2}\right\} \qquad (4.15)$$
$$>\sum_{x=y+2}^{\infty}(1-P_S)^{x-1}\left\{\sum_{i=y+1}^{x-1}({}^{i-1}C_y)(1-P_B)^{x+T-y-2}P_B^{\,y+1}\right\}$$

The presence of combinations makes it difficult to analytically prove Equation (4.15). We argue in a non-rigorous manner that this has to be true. Figure 4.9 serves as a guide through the logic. The notation employed in Figure 4.9 is initially described.

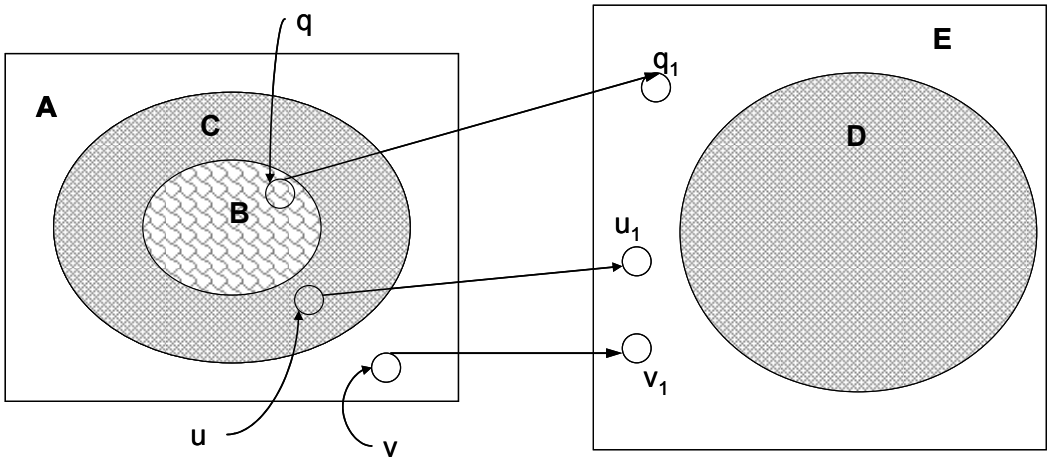


Figure 4.9: Why the probability of incurring a penalty is non-decreasing when the batch processor is under no-idling policy.

Assume that the feeder processor takes x time units to process the job. Let the left rectangle be the set of events that can occur within $x + T - 1$ time units when the initial buffer level is $n \times Q - 1$, and let the right rectangle be the set of events that can occur within $x + T - 1$ time units when the initial buffer level is $n \times Q$. Each element of the left rectangle is a parent of at least one element to the right, and can serve as a parent of more than one element to the right. Set B∪A is the set of events where a penalty is incurred when the initial buffer level is $n \times Q - 1$, and any element inside B (element q) or A (element v) will be parents to elements that belong to Set E, which is the set of events where a penalty is incurred when the initial buffer level is $n \times Q$. Furthermore, there may exist an element u in E whose parent is in Set C\B.

Assume that it takes the serial processor x time units to process the prospective job. Let the left rectangle be the set of events that could occur in $x + T - 1$ time units, if the initial queue length is $n \times Q - 1$, for integer n. In the left rectangle, we are in Case 3.2. Let the right rectangle be the set of events that could occur in $x + T - 1$ time units, if the initial queue length is $n \times Q$. In the right rectangle, we are in Case 3.1. Each element in each set is a set of transitions from the common initial state to a particular final state, after $x + T - 1$ time units. Each Element e of the left rectangle is a parent to at least one element at the right rectangle; if e represents a series of transitions that do not empty cause the batch processor to be idle, then the entire $x + T - 1$ time units is used, and e has exactly one descendant element in the right rectangle, which is itself. If e represents a series of transitions that causes the batch processor to be idle, then the addition of jobs (moving from left to right rectangle) can result in e having more than one descendant, whose series of transitions are identical to e until the batch processor is idle.

Let Set C denote the set of events that form the larger ellipse at the left rectangle, with u being an element of C. C is the set of events that cause $\lfloor Z_t Q^{-1} \rfloor + 1$ batches to be finished by $x + T - 1$ time units. Let Set A be the set of events that form the area of the rectangle less C, and let v be an element of A. A is the set of events for which $\lfloor Z_t Q^{-1} \rfloor + 1$ batches are not finished by $x + T - 1$ time units, which incurs a penalty, under scenario 2 of case 3.2. Let Set B denote the set of events that form the smaller ellipse in the left rectangle, with q as an element of B. B is the set of events that cause $\lfloor Z_t Q^{-1} \rfloor + 1$ batches to be finished by $x - 1$ time units, but the batch number $\lfloor Z_t Q^{-1} \rfloor + 2$ is not yet finished by $x + T - 1$ time units. B is a subset of C, and the probability of Scenario 1 is the probability of Set B. For the right rectangle, Set D is the set of events that form the ellipse, while Set E is the set of events that do not belong to D. D is the set of events that cause $\lfloor Z_t Q^{-1} \rfloor + 1$ batches to be finished by $x + T - 1$ time units, so E is the set of events that result in the prospective job incurring a penalty. The elements q_1, v_1 and u_1 all belong to E.

We wish to check the probability of incurring a penalty when the buffer size is $n \times Q - 1$ and $n \times Q$, for integer n. When $Z_t = n \times Q - 1$, we operate under case 3.2 (Left rectangle). Under Case 3.2, the sequence of events that cause the prospective job to incur a penalty are:

(1) batch processor manages to process n batches within $x - 1$ time units, but the $n + 1^{th}$ batch does not finish processing until after $x + T - 1$ time units. (Set C).

(2) batch processor fails to process n batches within $x + T - 1$ time units. (Set A).

In both sequences of events, the series of transitions will not empty the batch processor. Thus, each element in C and A will have exactly one descendant in the right rectangle. The probabilities of each individual transition remain unchanged with the addition of jobs into the buffer, so the individual probabilities of each element in C and A remain unchanged for their descendant elements in the right rectangle.

When $Z_t = n \times Q$, the prospective job will be forced to form a new batch, and we operate under Case 3.1 (Right rectangle). The prospective job will incur a penalty if the batch processor is not able to process $n + 1$ jobs within $x + T - 1$ time units. Any sequence of events in $x + T - 1$ time units that would cause the prospective job to incur a penalty when $Z_t = n \times Q - 1$ will cause the prospective job to incur a penalty when $Z_t = n \times Q$. When $Z_t = n \times Q$, $n + 1$ batches need to be processed by $x + T - 1$ time units. Both the elements in C and in A are unable to do this. Thus, the descendants of elements C and A belong to E, which implies that the total probability of incurring a penalty for the left rectangle (P(elements in C) + P(elements in A)) cannot be larger than the probability of incurring a penalty for the right rectangle (P(elements in E)). Furthermore, there may exist elements in Set C\B whose descendants may be in E. An example would be an Element u that manages to process the nth batch at the $x + T - 1^{th}$ time unit. This sequence of transitions will not incur a penalty when $Z_t = n \times Q - 1$, but will incur a penalty when $Z_t = n \times Q$. Since the probability of any element is always non-negative, the existence of u implies that the probability of incurring a penalty is non-decreasing with the respect to the queue length. Numerical experiments were conducted to support this argument. Table 4.3 contains the experimental setup used. Figure 4.10 shows the probabilities of incurring a penalty for Model Two, when the batch processor is under no-idling policy.

TABLE 4.3: EXPERIMENTAL SETUP TO SUPPORT HYPOTHESIS

Possible values for P_B	Possible values for P_S	Possible values for T	Possible values for Z_t
0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	10, 20, 30	0 to 20

These experiments suggest that the probability of incurring a penalty is non-decreasing with respect to the number of jobs in front of the batch processor. Furthermore, these experiments also suggest that the probability of incurring a penalty becomes asymptotically dependent only on the number of batches in front of the batch processor, as Z_t is increased. This is because the probability of Scenario one happening under Case 3.2 approaches zero as the number of jobs in front of the batch processor increases. This is most apparent when T has a high value, in which case the optimal policy can be approximated with a high degree of confidence by simply testing integer multiples of the batch processor size.

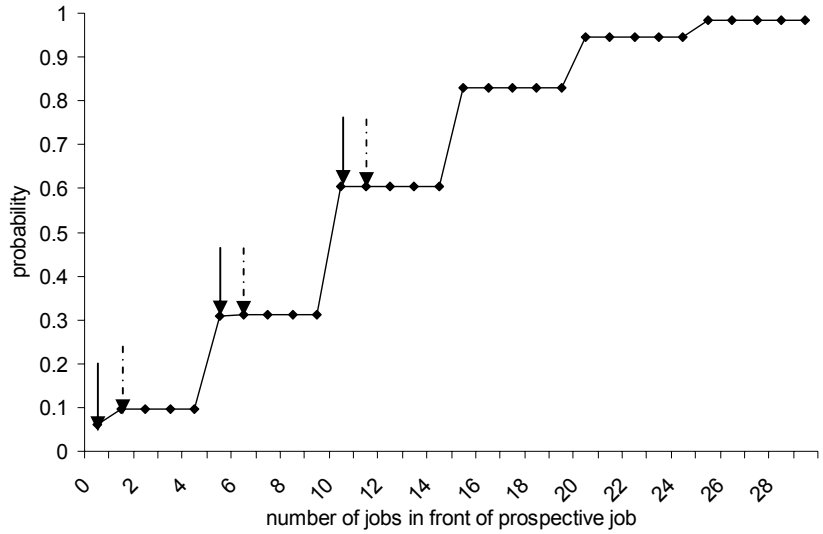


Figure 4.10: Probability of the prospective job incurring a penalty when the batch processor is busy and under no-idling policy.

Parameters: $T = 8$, $Q = 5$, $P_S = 0.5$ and $P_B = 0.25$. When the batch processor is idle, the probability of incurring a penalty is zero. The solid arrows point to a few instances when the number of batches in front of the prospective job is fixed, since the prospective job will form a new batch. The dashed arrows point to a few instances when the number of batches in front of the prospective batch is unknown. The difference between the two instances diminishes very rapidly as the number of full batches in front of the batch processor increases..

If T is low and the exact optimal policy is needed, we can initially evaluate the probability of incurring a penalty at Z_t = integer multiples of Q. Let the integer v be such that R/C is above

the probability of incurring a penalty when $Z_t = v \times Q$, and R/C is below the probability of incurring a penalty when $Z_t = (v + 1) \times Q$. Then, we only need to test for profitability when $Z_t = v \times Q + 1$ to determine the optimal serial processor policy.

4.7.3.2 When the batch processor is under full-batch policy

For instances where the full-batch policy (or any minimum batch size policy with minimum batch size > 1) is used at the batch processor, the probability of incurring a penalty is the union of two probabilities. The first probability $P(T_A)$ corresponds to the time it takes to form a full batch, and it is a decreasing function with respect to the modulus of the buffer level and the batch processor size. This means that, the $n \times Q + 1^{st}$ job will have to wait a longer time to form a full batch compared to the $(n + 1) \times Q^{th}$ job. This component is periodic with respect to the number of jobs in queue; and one cycle corresponds to Q.

The second component $P(T_B)$ corresponds to the time it takes to process all the previous batches at the batch processor, and is strictly increasing on the number of batches in front of the job. Figure 4.11 shows the combined effect of $P(T_A)$ and $P(T_B)$.

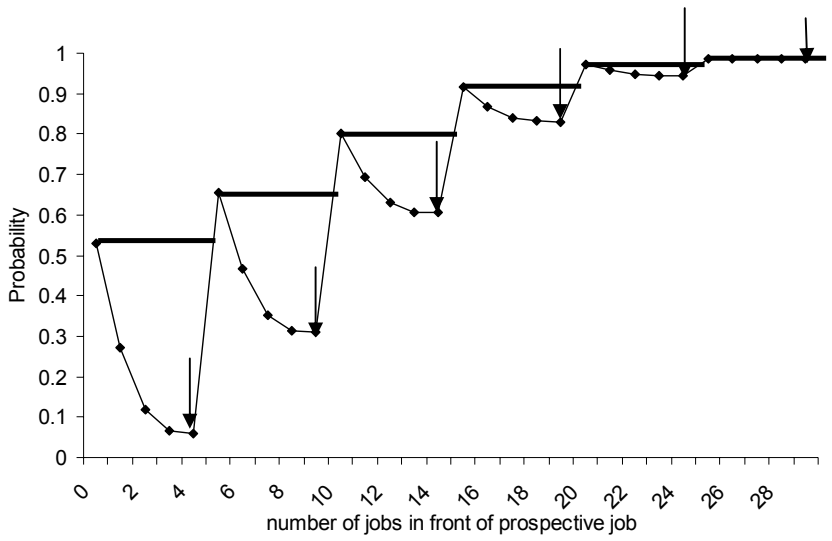


Figure 4.11: Total probability of the prospective job incurring a penalty when the batch processor is busy and under full-batch policy.

Parameters: $T = 8$, $Q = 5$, $P_S = 0.5$ and $P_B = 0.25$. Probability is not increasing with respect to batch processor queue length, due to the need to wait for additional job arrivals. Points with arrows correspond to probabilities that are entirely due to batch processor unavailability, as the jobs no longer need to wait for additional job arrivals to form a full batch. The heavy lines indicate the monotonically increasing step function that approximates the probability of incurring a penalty.

The total probability of incurring a penalty is no longer non-decreasing with respect to the number of jobs in front of the processor. However, for the same number of full batches in front of the batch processor, if the probability of incurring a penalty is lower than R/C when starting a batch (which occurs when $Z_t = n \times Q$) has been made, then the probabilities of incurring a penalty to form a full batch will also be lower than R/C. We can approximate the probability of incurring a penalty with a monotonically increasing step function, where the steps occur at integer multiples of the batch processor. Given this approximate probability function, the optimal policy then becomes a threshold policy. The use of the threshold policy makes it necessary to only test for states with values of the buffer level Z_t which are integer multiples of the batch processor.

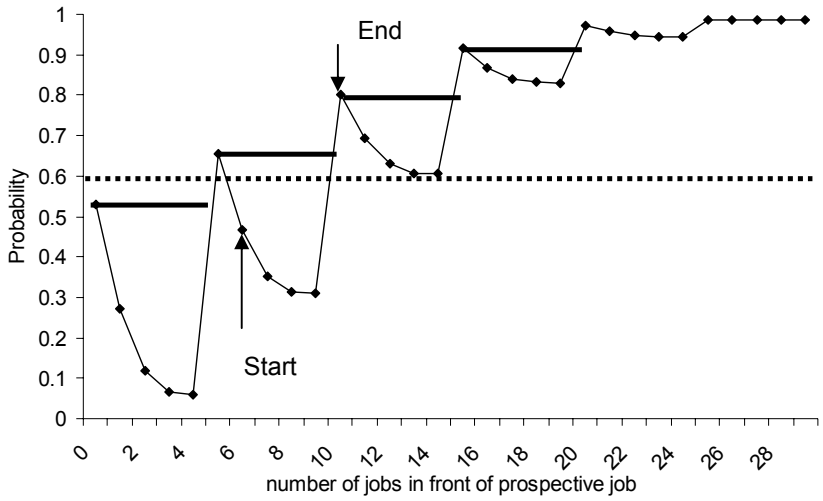


Figure 4.12: Total probability, and approximate probability plot, of the prospective job incurring a penalty when the batch processor is busy and under full-batch policy.

Parameters: $T = 8$, $Q = 5$, $P_S = 0.5$ and $P_B = 0.25$. The heavy horizontal lines represent the approximate probability plot. The horizontal dotted line is the R/C value. If the system is not started empty, and actually starts with 6 jobs (arrow labeled "start"), using the approximate function will cause the feeder processor stop processing jobs, while using the actual probability function will cause the serial processor to process jobs until the queue length reaches 10 (arrow labeled "end") or the batch processor finishes and causes the queue length to drop to below 5. However, when either event occurs, both functions will be either under the R/C line (if the batch processor depletes the queue length to below 5 jobs), or over the R/C line (if the batch processor queue reaches 10 jobs.)

Given the assumption that any job with negative expected reward is not processed, we determine the effect of approximating the actual probability curve with the step-wise function. If the system is started with an empty buffer, following the actual probability function will cause no decisions to differ from following the approximate stepwise function. If the system starts with a non-empty buffer, there might be an initial phase where using the approximate and actual probability functions may have different decisions, but the number of different states for which the two policies may differ in its decision is bounded from above by the batch processor size. Figure 4.12 provides an example as to how the decision making based on the two probability functions converge in their decisions.

Figure 4.13 shows a typical probability plot for two systems that are identical except for their batch processor control policy. Neither policy is guaranteed to have a strictly higher probability of incurring a penalty. There are particular states where the batch processor under no-idling policy is guaranteed to have a higher probability of incurring a penalty. This occurs when the prospective job does not have to wait for additional jobs to form a full batch.

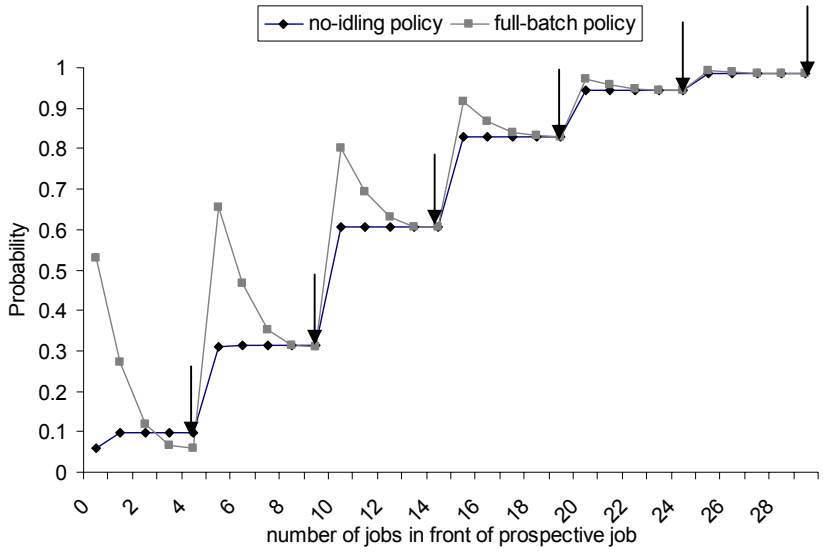


Figure 4.13: Comparing probabilities of the prospective job incurring a penalty when the batch processor is busy and under differing policies.

Parameters: $T = 8$, $Q = 5$, $P_S = 0.5$ *and* $P_B = 0.25$. When the batch processor is under full-batch policy, the probability of incurring a penalty while waiting for additional job arrivals has a large effect, and this effect diminishes as the queue length increases. Arrows point to states where the batch processor under no-idling policy has higher probability of incurring penalty.

Figure 4.13 also shows the proportion of jobs incurring at least a certain queuing time T under two different batch processor policies. For the particular set of parameters chosen, when the number of jobs queued in front of the batch processor is low, the time spent waiting for a full batch causes a significant number of jobs to incur long cycle times. Thus, when the system is experiencing low traffic intensity and often has only a few jobs in front of the batch processor, using the full-batch policy results in more jobs having their queue time reach a particular value. The performance of a system under a threshold policy can also be determined by modeling the system as a Discrete Time, Discrete State Markov Chain. At Time Instance t, Let the state of the system be $X_t = (Z_t, S_t, F_t)$, with Z_t, S_t and F_t as defined in Section 4.7.2. Let L be the threshold value selected, and assume $L > Q$. The transition probabilities for the system when the batch processor is under no-idling policy are detailed in Table C-3 of Appendix C, while Table C-4 (also in Appendix C) contains the transition probabilities for the system when the batch processor is under full-batch policy. In both cases, the throughput rate can be obtained by obtaining the steady-state probability that the serial processor is busy, and multiplying this with the mean processing rate of the serial processor ($1 / P_S$).

4.8 Model Three Assumptions and Development

In this section, we discuss the development of a model where the batch processor is unreliable.

4.8.1 Model Three Assumptions

The following assumptions are made in Model Three:

- Serial processor is perfectly reliable.
- Serial processor processing time is geometrically distributed.

- At any time instance the batch processor is busy, there is a probability P_F that the batch processor will fail. At any instance the batch processor is down, there is a probability P_R that the batch processor will be repaired. Thus, the Time To Fail and the Time To Repair are both geometrically distributed. Furthermore, we assume that a batch processor cannot finish a batch and also fail at the same time instance. A processor can only fail if it is busy.
- When the batch processor fails, the current batch it is operating on exits the system. Since the *MTTF* and *MTTR* of the batch processor is not subject to control, we do not associate any penalty to a batch failure.
- Batch processor processing time is geometrically distributed.
- Buffer level does not include the jobs already being processed by the batch processor

4.8.2 Model Three Development

The state of the system X_t can be described as $X_t = (Z_t, S_t, F_t)$. We omit F_t, since we can only make decisions if $F_t = 0$. Table 4.4 contains an outline on how development of Model Three will proceed.

TABLE 4.4: DIVISION OF ANALYSIS FOR MODEL THREE

Batch processor policy	Batch processor status	Segregation of state space	Further subdivision into scenarios
No-idling	Up	Case One: processor idle	
		Case Two: less than one full batch available at buffer	Six different scenarios possible
		Case Three: at least one full batch available at buffer	Case 3.1: Buffer is integer multiple of processor size
			Case 3.2: Buffer is not integer multiple of processor size. Two different scenarios possible
	Down	Case One: less than one full batch available at buffer	Three different scenarios possible
		Case Two: at least one full batch available at buffer	Case 2.1: Buffer is integer multiple of processor size
			Case 2.2: Buffer is not integer multiple of processor size. Two different scenarios possible
Full-batch	Up	Case One: processor idle	
		Case Two: less than one full batch available at buffer	Two different scenarios possible
		Case Three: at least one full	

		batch available at buffer	
	Down	Case One: less than one full batch available at buffer	
		Case Two: at least one full batch available at buffer	

4.8.2.1 When the Batch Processor is under No-Idling Policy

We derive the equations used to determine the optimal policy when the batch processor is under no-idling policy in this section.

4.8.2.1.1 When the Batch Processor is Initially Up

Development is once again divided into three cases:

Case One: $X_t = (0, 0)$

We should always process the job, since the waiting time will be zero. This is a consequence of the assumption that current batch does not return to the buffer when batch processor fails.

Case Two: $X_t = (0 \text{ to } Q - 1, 1)$

There are six scenarios that could lead to the prospective job to incur waiting time greater than its processing time window.

- *Scenario One:* Batch processor is not yet finished with the current batch when the prospective job's processing time window is reached.

Probability (job will incur penalty due to Scenario One | serial processor took x time units)

$$= \left[(1-P_F)(1-P_B)\right]^{x+T-1}$$

Probability (job will incur penalty due to Scenario One)

$$= \sum_{x=1}^{\infty} (1-P_S)^{x-1} P_S \left[(1-P_F)(1-P_B)\right]^{x+T-1}$$
$$= P_S \left[(1-P_F)(1-P_B)\right]^{T} \Big/ 1-\left[(1-P_S)(1-P_F)(1-P_B)\right] \quad (4.16)$$

- *Scenario Two:* batch processor gets spoilt while processing current batch, and repair of batch processor does not take place until after the prospective job's processing time window is reached.

Let i be the number of time instances incurred before the batch processor fails. Then the probability (job will incur penalty due to Scenario Two | serial processor took x time units)

$$= \sum_{i=0}^{x+T-2} P_F \left[(1-P_F)(1-P_B)\right]^i (1-P_R)^{T+x-i-2} .$$

Probability (job will incur penalty due to Scenario Two)

$$= \sum_{x=1}^{\infty} \sum_{i=0}^{x+T-2} (1-P_s)^{x-1} P_s P_F \left[(1-P_F)(1-P_B)\right]^i (1-P_R)^{T+x-i-2}$$

$$= \sum_{x=1}^{\infty} (1-P_s)^{x-1} P_s P_F (1-P_R)^{T+x-2} \sum_{i=0}^{x+T-2} \left[\frac{(1-P_F)(1-P_B)}{1-P_R}\right]^i$$

$$= \sum_{x=1}^{\infty} (1-P_s)^{x-1} P_s P_F (1-P_R)^{T+x-2} \frac{(1-P_R)^{x+T-1} - \left[(1-P_F)(1-P_B)\right]^{x+T-1}}{(1-P_R)^{x+T-2}\left[1-P_R-(1-P_F)(1-P_B)\right]}$$

$$= \sum_{x=1}^{\infty} (1-P_s)^{x-1} P_s P_F \frac{(1-P_R)^{x+T-1} - \left[(1-P_F)(1-P_B)\right]^{x+T-1}}{\left[1-P_R-(1-P_F)(1-P_B)\right]}$$

Letting $c_1 = \dfrac{P_s P_F}{\left[1-P_R-(1-P_F)(1-P_B)\right]}$, P(job will incur penalty due to Scenario Two) is

$$= c_1 \sum_{x=1}^{\infty} (1-P_s)^{x-1} \left[(1-P_R)^{x+T-1} - \left[(1-P_F)(1-P_B)\right]^{x+T-1}\right]$$

$$= c_1 \sum_{x=1}^{\infty} (1-P_s)^{x-1} \left[(1-P_R)^{T}(1-P_R)^{x-1} - \left[(1-P_F)(1-P_B)\right]^{T}\left[(1-P_F)(1-P_B)\right]^{x-1}\right]$$

$$= c_1 \sum_{x=0}^{\infty} (1-P_s)^{x} \left[(1-P_R)^{T}(1-P_R)^{x} - \left[(1-P_F)(1-P_B)\right]^{T}\left[(1-P_F)(1-P_B)\right]^{x}\right]$$

$$= c_1 \left[\frac{(1-P_R)^T}{1-(1-P_s)(1-P_R)} - \frac{\left[(1-P_F)(1-P_B)\right]^T}{1-(1-P_s)(1-P_B)(1-P_F)}\right]$$

$$= \frac{P_F P_s}{(1-P_R-(1-P_F)(1-P_B))}\left[\frac{(1-P_R)^T}{1-(1-P_s)(1-P_R)} - \frac{\left((1-P_F)(1-P_B)\right)^T}{1-(1-P_s)(1-P_F)(1-P_B)}\right] \qquad (4.17)$$

- *Scenario Three:* batch processor fails while processing current batch and is repaired before the prospective job exits the serial processor. New batch is loaded and the new batch finishes processing only after the prospective job reaches its processing time window. This is not possible if $Z_t = 0$, as the new batch will not contain any jobs.

Let j refer to the number of time instances before the batch processor fails, while $i + 1$ refers to the number of time instances before the batch processor gets repaired. Then the probability of penalty being incurred under Scenario Three conditioned on serial processor finishing at x:

$$= \sum_{i=0}^{x-3} P_F P_R \left[(1-P_F)(1-P_B)\right]^{T+x-i-3} \sum_{j=0}^{i} \left[(1-P_F)(1-P_B)\right]^{j} (1-P_R)^{i-j} \quad \text{if } x > 2, \text{ and zero}$$

otherwise. The failure and subsequent repair of the processor takes at least two time units, and these events happen before the prospective job joins the queue, which is why $x > 2$ for Scenario Three to be possible.

Probability of penalty being incurred due to Scenario Three is:

$$\sum_{x=3}^{\infty} (1-P_S)^{x-1} P_S \sum_{i=0}^{x-3} P_F P_R \left[(1-P_F)(1-P_B)\right]^{T+x-i-3} \sum_{j=0}^{i} \left[(1-P_F)(1-P_B)\right]^{j} (1-P_R)^{i-j}$$

Let $c_0 = (1-P_F)(1-P_B)$, then the probability of incurring a penalty due to Scenario Three is:

$$\sum_{x=3}^{\infty} (1-P_S)^{x-1} P_S \sum_{i=0}^{x-3} P_F P_R {c_0}^{T+x-i-3} \sum_{j=0}^{i} {c_0}^{j} (1-P_R)^{i-j}$$

$$= \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} \sum_{j=0}^{i} (1-P_S)^{x-1} P_S P_F P_R {c_0}^{T+x-i-3+j} (1-P_R)^{i-j}$$

$$= \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} \sum_{j=0}^{i} (1-P_S)^{x-1} P_S P_F P_R {c_0}^{T+x-i-3} (1-P_R)^{i} \left[c_0 \Big/ 1-P_R \right]^{j}$$

$$= \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} (1-P_S)^{x-1} P_S P_F P_R {c_0}^{T+x-i-3} (1-P_R)^{i} \left[\frac{1-\left[c_0 \Big/ 1-P_R \right]^{i+1}}{1-\left[c_0 \Big/ 1-P_R \right]} \right]$$

$$= \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} (1-P_S)^{x-1} P_S P_F P_R {c_0}^{T+x-i-3} (1-P_R)^{i} \left[\frac{(1-P_R)^{i+1} - {c_0}^{i+1}}{(1-P_R)^{i}\left[(1-P_R)-c_0\right]} \right]$$

$$= \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} (1-P_S)^{x-1} P_S P_F P_R {c_0}^{T+x-i-3} \left[\frac{(1-P_R)^{i+1} - {c_0}^{i+1}}{\left[(1-P_R)-c_0\right]} \right]$$

$$= \left[\frac{P_S P_F P_R {c_0}^{T-3}}{1-P_R-c_0} \right] \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} (1-P_S)^{x-1} \frac{{c_0}^{x}}{{c_0}^{i}} \left[(1-P_R)^{i+1} - {c_0}^{i+1} \right]$$

$$= \left[\frac{P_S P_F P_R {c_0}^{T-3}}{1-P_R-c_0} \right] \sum_{x=3}^{\infty} \sum_{i=0}^{x-3} (1-P_S)^{x-1} {c_0}^{x} \left[\left(\frac{1-P_R}{c_0} \right)^{i} (1-P_R) - \frac{{c_0}^{i+1}}{{c_0}^{i}} \right]$$

$$= \left[\frac{P_S P_F P_R {c_0}^{T-3}}{1-P_R-c_0} \right] \sum_{x=3}^{\infty} (1-P_S)^{x-1} {c_0}^{x} \left[\left[\frac{(1-P_R)\left[{c_0}^{x-2} - (1-P_R)^{x-2} \right]}{{c_0}^{x-3}\left[c_0 - (1-P_R)\right]} \right] - (x-2)c_0 \right]$$

Letting $c_1 = \dfrac{P_S P_F P_R {c_0}^{T-3}}{1-P_R-c_0}$, then P(incurring penalty due to Scenario Three) is

$$= c_1\left[\sum_{x=3}^{\infty}(1-P_S)^{x-1}{c_0}^{x}\left[\frac{(1-P_R)\left[{c_0}^{x-2}-(1-P_R)^{x-2}\right]}{{c_0}^{x-3}\left[c_0-(1-P_R)\right]}\right]-\sum_{x=3}^{\infty}(1-P_S)^{x-1}{c_0}^{x}(x-2)c_0\right]$$

$$= \frac{(1-P_S)c_1{c_0}^{3}(1-P_R)}{c_0-(1-P_R)}\left[\sum_{x=3}^{\infty}(1-P_S)^{x-2}\left[{c_0}^{x-2}-(1-P_R)^{x-2}\right]\right]$$

$$-(1-P_S)c_1{c_0}^{3}\sum_{x=3}^{\infty}(1-P_S)^{x-2}{c_0}^{x-2}(x-2)$$

substituting the constants $c_2 = \frac{(1-P_S)c_1{c_0}^{3}(1-P_R)}{c_0-(1-P_R)}$ and $c_3 = c_1{c_0}^{3}(1-P_S)$, we get:

$$= c_2\left[\sum_{x=3}^{\infty}(1-P_S)^{x-2}\left[{c_0}^{x-2}-(1-P_R)^{x-2}\right]\right]-c_3\sum_{x=3}^{\infty}(1-P_S)^{x-2}{c_0}^{x-2}(x-2)$$

$$= c_2\left[\sum_{x=1}^{\infty}(1-P_S)^{x}\left[{c_0}^{x}-(1-P_R)^{x}\right]\right]-c_3\sum_{x=0}^{\infty}x(1-P_S)^{x}{c_0}^{x}$$

$$= c_2\left[\sum_{x=0}^{\infty}(1-P_S)^{x}\left[{c_0}^{x}-(1-P_R)^{x}\right]\right]-c_3\sum_{x=0}^{\infty}x(1-P_S)^{x}{c_0}^{x}$$

$$= c_2\left[\frac{1}{1-(1-P_S)c_0}-\frac{1}{1-(1-P_S)(1-P_R)}\right]-c_3\left[\frac{(1-P_S)c_0}{\left[1-(1-P_S)c_0\right]^2}\right]$$

$$= \frac{(1-P_S)c_1{c_0}^{3}(1-P_R)}{c_0-(1-P_R)}\left[\frac{1}{1-(1-P_S)c_0}-\frac{1}{1-(1-P_S)(1-P_R)}\right]$$

$$-c_1{c_0}^{3}(1-P_S)\left[\frac{(1-P_S)c_0}{\left[1-(1-P_S)c_0\right]^2}\right]$$

$$= \frac{(1-P_S)c_1{c_0}^{3}(1-P_R)}{c_0-(1-P_R)}\frac{c_0-(1-P_R)}{\left[1-(1-P_S)c_0\right]\times\left[1-(1-P_S)(1-P_R)\right]}-\frac{c_1{c_0}^{4}(1-P_S)^2}{\left[1-(1-P_S)c_0\right]^2}$$

$$= \left[\frac{P_SP_FP_R{c_0}^{T}}{1-P_R-c_0}\right]\frac{(1-P_S)(1-P_R)}{\left[1-(1-P_S)c_0\right]\times\left[1-(1-P_S)(1-P_R)\right]}$$

$$-\left[\frac{P_SP_FP_R{c_0}^{T+1}}{1-P_R-c_0}\right]\frac{(1-P_S)^2}{\left[1-(1-P_S)c_0\right]^2} \qquad (4.18)$$

$$\left[\frac{P_SP_FP_R{c_0}^{T}(1-P_S)}{\left[1-P_R-c_0\right]\times\left[1-(1-P_S)c_0\right]}\right]\times\left[\frac{(1-P_R)}{1-(1-P_S)(1-P_R)}-\frac{c_0(1-P_S)}{1-(1-P_S)c_0}\right]$$

If $Z_t > 0$, and 0 otherwise.

- *Scenario Four:* current batch at batch processor finishes processing and loads a new batch (emptying the buffer in the process) before the prospective job exits the serial

processor. The new batch finishes processing only after the prospective job reaches its processing time window. This is only possible if $Z_t > 0$.

Let i refer to the number of time instances before the first batch finishes processing, then the probability of penalty incurred under Scenario Four conditioned on serial processor finishing at x is

$$\begin{aligned} &= \sum_{i=0}^{x-2} (1-P_F)^{i+1} (1-P_B)^i P_B \left[(1-P_B)(1-P_F)\right]^{T+x-i-2} \\ &= (x-1)(1-P_F)^{T+x-1} (1-P_B)^{T+x-2} P_B \end{aligned}$$ if $x > 1$, and zero otherwise

So probability of penalty being incurred due to Scenario Four is:

$$\sum_{x=2}^{\infty} (x-1)(1-P_S)^{x-1} P_S (1-P_F)^{T+x-1} (1-P_B)^{T+x-2} P_B$$

$$= P_S P_B (1-P_F)^T (1-P_B)^{T-1} \sum_{x=2}^{\infty} (x-1)(1-P_S)^{x-1} (1-P_F)^{x-1} (1-P_B)^{x-1}$$

$$= P_S P_B (1-P_F)^T (1-P_B)^{T-1} \sum_{x=0}^{\infty} x(1-P_S)^x (1-P_F)^x (1-P_B)^x$$

$$= P_S P_B (1-P_F)^T (1-P_B)^{T-1} \frac{(1-P_S)(1-P_F)(1-P_B)}{\left[1-(1-P_S)(1-P_F)(1-P_B)\right]^2}$$

$$= \frac{P_S P_B (1-P_F)^{T+1} (1-P_B)^T (1-P_S)}{\left[1-(1-P_S)(1-P_F)(1-P_B)\right]^2}$$ (4.19) If $Z_t > 0$, and 0 otherwise.

- *Scenario Five:* batch processor finishes processing current batch and loads a new batch (emptying the buffer in the process) before the prospective job exits the serial processor. The batch processor gets spoilt but is not repaired before the processing time window of the prospective job is reached. This scenario is only possible if $Z_t > 0$.

Let i be the number of time instances it takes before the batch processor finishes processing the first batch, and j be the number of unsuccessful repair attempts. Then probability (job will incur penalty under Scenario Five | serial processor took x time units)

$$= \sum_{i=0}^{x-2} (1-P_F)^{i+1} (1-P_B)^i P_B \sum_{j=0}^{T+x-i-3} \left[(1-P_B)(1-P_F)\right]^{T+x-i-3-j} P_F (1-P_R)^j$$

$$= \sum_{i=0}^{x-2} \sum_{j=0}^{T+x-i-3} (1-P_F) P_B \left[(1-P_B)(1-P_F)\right]^{T+x-3-j} P_F (1-P_R)^j$$

if $x > 1$ and zero otherwise

Letting $c_0 = (1-P_B)(1-P_F)$, then probability (job will incur penalty) under Scenario Five

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}\sum_{j=0}^{T+x-i-3}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3-j}P_F(1-P_R)^j$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3}P_F\sum_{j=0}^{T+x-i-3}\left(1-P_R\big/c_0\right)^j$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3}P_F\left[\frac{{c_0}^{T+x-i-2}-(1-P_R)^{T+x-i-2}}{{c_0}^{T+x-i-3}\left[c_0-(1-P_R)\right]}\right]$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}\sum_{j=0}^{T+x-i-3}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3-j}P_F(1-P_R)^j$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3}P_F\sum_{j=0}^{T+x-i-3}\left(\frac{1-P_R}{c_0}\right)^j$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{T+x-3}P_F\left[\frac{{c_0}^{T+x-i-2}-(1-P_R)^{T+x-i-2}}{{c_0}^{T+x-i-3}\left[c_0-(1-P_R)\right]}\right]$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_F)P_B{c_0}^{i}P_F\left[\frac{{c_0}^{T+x-i-2}-(1-P_R)^{T+x-i-2}}{c_0-(1-P_R)}\right]$$

Substituting $c_1 = \dfrac{P_S c_0 P_B P_F}{c_0-(1-P_R)}$, we obtain the expression

$$= c_1\sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}{c_0}^{i}\left[{c_0}^{T+x-i-2}-(1-P_R)^{T+x-i-2}\right]$$

$$= c_1\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[\sum_{i=0}^{x-2}{c_0}^{i}\left[{c_0}^{T+x-i-2}\right]-\sum_{i=0}^{x-2}{c_0}^{i}\left[(1-P_R)^{T+x-i-2}\right]\right]$$

$$= c_1\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[\sum_{i=0}^{x-2}\left[{c_0}^{T+x-2}\right]-(1-P_R)^{T+x-2}\sum_{i=0}^{x-2}\left[\frac{c_0}{1-P_R}\right]^i\right]$$

$$= c_1\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[(x-1){c_0}^{T+x-2}-(1-P_R)^{T+x-2}\frac{(1-P_R)^{x-1}-{c_0}^{x-1}}{(1-P_R)^{x-2}\left[(1-P_R)-c_0\right]}\right]$$

$$= c_1\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[(x-1){c_0}^{T+x-2}-(1-P_R)^{T}\frac{(1-P_R)^{x-1}-{c_0}^{x-1}}{1-P_R-c_0}\right]$$

$$= c_1 \sum_{x=2}^{\infty} (1-P_S)^{x-1} \left[(x-1) c_0{}^{T+x-2} - (1-P_R)^T \frac{(1-P_R)^{x-1} - c_0{}^{x-1}}{1-P_R-c_0} \right]$$

$$= c_1 \left[c_0{}^{T-1} \sum_{x=2}^{\infty} (x-1) \left[(1-P_S) c_0 \right]^{x-1} - \frac{(1-P_R)^T}{1-P_R-c_0} \sum_{x=2}^{\infty} (1-P_S)^{x-1} \left[(1-P_R)^{x-1} - c_0{}^{x-1} \right] \right]$$

$$= c_1 \left[c_0{}^{T-1} \sum_{x=0}^{\infty} x \left[(1-P_S) c_0 \right]^{x} - \frac{(1-P_R)^T}{1-P_R-c_0} \sum_{x=0}^{\infty} (1-P_S)^{x} \left[(1-P_R)^{x} - c_0{}^{x} \right] \right]$$

$$= c_1 \left[c_0{}^{T-1} \frac{(1-P_S) c_0}{\left[1-(1-P_S) c_0 \right]^2} - \frac{(1-P_R)^T}{1-P_R-c_0} \sum_{x=0}^{\infty} (1-P_S)^{x} \left[(1-P_R)^{x} - c_0{}^{x} \right] \right]$$

$$= c_1 \left[\frac{(1-P_S) c_0{}^{T}}{\left[1-(1-P_S) c_0 \right]^2} - \frac{(1-P_R)^T}{1-P_R-c_0} \left[\frac{1}{1-(1-P_S)(1-P_R)} - \frac{1}{1-\left[(1-P_s) c_0 \right]} \right] \right]$$

$$= \frac{P_S (1-P_F)(1-P_B) P_B P_F}{c_0 - (1-P_R)} \times \left[\begin{array}{l} \dfrac{(1-P_S) c_0{}^{T}}{\left[1-(1-P_S) c_0 \right]^2} \\ - \dfrac{(1-P_R)^T}{1-P_R-c_0} \times \dfrac{c_0 - (1-P_R)}{\left[1-(1-P_S)(1-P_R) \right] \times \left[1-(1-P_s) c_0 \right]} \end{array} \right]$$

$$= \frac{P_S c_0 P_B P_F}{\left[c_0 - (1-P_R) \right] \times \left[1-(1-P_S) c_0 \right]} \times \left[\frac{(1-P_S) c_0{}^{T}}{1-(1-P_S) c_0} + \frac{(1-P_R)^T}{1-(1-P_S)(1-P_R)} \right]$$

(4.20) If $Z_t > 0$, and 0 otherwise.

- *Scenario Six:* batch processor gets spoilt while processing current batch and gets repaired before serial processor finishes processing prospective job. A new batch is loaded. Unfortunately, the batch processor gets spoilt again and is repaired only until after the prospective job reaches its processing time window. This scenario is only possible if $Z_t > 0$.

Let i be the number of time units needed for the batch processor to initially fail and be repaired. i has to be less than x, otherwise, the prospective job will have already joined the queue. Let k be the number of unsuccessful attempts to process the first batch before the batch processor failed the first time, and let j be the number of unsuccessful attempts to repair the batch processor the second time it fails. The probability (job incurring penalty under Scenario Six | serial processor took x time units)

$$= \sum_{i=2}^{x-1}\sum_{k=0}^{i-2}\left[(1-P_F)(1-P_B)\right]^{i-k-2}\left(P_F{}^2\right)(1-P_R)^k P_R{}^{T+x-i-2}\sum_{j=0}^{}\left[(1-P_B)(1-P_F)\right]^{T+x-i-2-j}(1-P_R)^j$$

$$= \sum_{i=2}^{x-1}\sum_{k=0}^{i-2}\sum_{j=0}^{T+x-i-2}\left(P_F{}^2\right)P_R\left[(1-P_B)(1-P_F)\right]^{T+x-4-j-k}(1-P_R)^{j+k}$$

if $x > 2$ and 0 otherwise.

Let $c_0 = (1-P_B)(1-P_F)$, then the probability (job incurring penalty due to Scenario Six) is:

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}\sum_{k=0}^{i-2}\sum_{j=0}^{T+x-i-2}(1-P_S)^{x-1}P_S\left(P_F{}^2\right)P_R c_0{}^{T+x-4-j-k}(1-P_R)^{j+k}$$

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}\sum_{k=0}^{i-2}(1-P_S)^{x-1}P_S P_F{}^2 P_R c_0{}^{T+x-4-k}(1-P_R)^k\sum_{j=0}^{T+x-i-2}\frac{(1-P_R)^j}{c_0{}^j}$$

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}\sum_{k=0}^{i-2}(1-P_S)^{x-1}P_S P_F{}^2 P_R c_0{}^{T+x-4-k}(1-P_R)^k\frac{c_0{}^{T+x-i-1}-(1-P_R)^{T+x-i-1}}{c_0{}^{T+x-i-2}\left[c_0-(1-P_R)\right]}$$

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}(1-P_S)^{x-1}P_S P_F{}^2 P_R c_0{}^{i-2}\frac{c_0{}^{T+x-i-1}-(1-P_R)^{T+x-i-1}}{c_0-(1-P_R)}\sum_{k=0}^{i-2}\frac{(1-P_R)^k}{c_0{}^k}$$

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}(1-P_S)^{x-1}P_S P_F{}^2 P_R c_0{}^{i-2}\frac{c_0{}^{T+x-i-1}-(1-P_R)^{T+x-i-1}}{c_0-(1-P_R)}\frac{c_0{}^{i-1}-(1-P_R)^{i-1}}{c_0{}^{i-2}\left[c_0-(1-P_R)\right]}$$

$$= \sum_{x=3}^{\infty}\sum_{i=2}^{x-1}(1-P_S)^{x-1}P_S P_F{}^2 P_R\left[c_0{}^{i-1}-(1-P_R)^{i-1}\right]\frac{c_0{}^{T+x-i-1}-(1-P_R)^{T+x-i-1}}{\left[c_0-(1-P_R)\right]^2}$$

$$= \sum_{x=3}^{\infty}\frac{(1-P_S)^{x-1}P_S P_F{}^2 P_R}{\left[c_0-(1-P_R)\right]^2}\sum_{i=2}^{x-1}\left[\left[c_0{}^{i-1}-(1-P_R)^{i-1}\right]\times\left[c_0{}^{T+x-i-1}-(1-P_R)^{T+x-i-1}\right]\right]$$

$$= \sum_{x=3}^{\infty}\frac{(1-P_S)^{x-1}P_S P_F{}^2 P_R}{\left[c_0-(1-P_R)\right]^2}\sum_{i=2}^{x-1}\left[\begin{array}{l}c_0{}^{T+x-2}+(1-P_R)^{T+x-2}\\-c_0{}^{T+x-i-1}(1-P_R)^{i-1}-c_0{}^{i-1}(1-P_R)^{T+x-i-1}\end{array}\right]$$

Defining $c_1 = \dfrac{P_S P_F{}^2 P_R}{\left[c_0-(1-P_R)\right]^2}$, we simplify the expression into

$$= c_1 \sum_{x=3}^{\infty} (1-P_S)^{x-1} \left[\begin{array}{l} (x-2){c_0}^{T+x-2} + (x-2)(1-P_R)^{T+x-2} \\ -\sum_{i=2}^{x-1} \left[{c_0}^{T+x-i-1} (1-P_R)^{i-1} + {c_0}^{i-1} (1-P_R)^{T+x-i-1} \right] \end{array} \right]$$

$$= c_1 \sum_{x=3}^{\infty} \left[\begin{array}{l} (1-P_S)^{x-1}(x-2){c_0}^{T+x-2} + (1-P_S)^{x-1}(x-2)(1-P_R)^{T+x-2} \\ -(1-P_S)^{x-1} \sum_{i=2}^{x-1} \left[{c_0}^{T+x-i-1} (1-P_R)^{i-1} + {c_0}^{i-1} (1-P_R)^{T+x-i-1} \right] \end{array} \right]$$

$$= c_1 \sum_{x=3}^{\infty} \left[\begin{array}{l} (1-P_S){c_0}^{T}(x-2)\left[(1-P_S)c_0\right]^{x-2} \\ +(1-P_R)^{T}(1-P_S)(1-P_S)^{x-2}(x-2)(1-P_R)^{x-2} \\ -(1-P_S)^{x-1} \sum_{i=2}^{x-1} \left[{c_0}^{T+x-i-1} (1-P_R)^{i-1} + {c_0}^{i-1} (1-P_R)^{T+x-i-1} \right] \end{array} \right]$$

$$= c_1 \left[\begin{array}{l} \frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} - \\ \sum_{x=3}^{\infty} (1-P_S)^{x-1} \sum_{i=2}^{x-1} \left[{c_0}^{T+x-i-1} (1-P_R)^{i-1} + {c_0}^{i-1} (1-P_R)^{T+x-i-1} \right] \end{array} \right]$$

$$= c_1 \left[\begin{array}{l} \frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} \\ -\sum_{x=3}^{\infty} (1-P_S)^{x-1} \sum_{i=2}^{x-1} \left[{c_0}^{T+x-2} \left(\frac{1-P_R}{c_0} \right)^{i-1} + (1-P_R)^{T+x-2} \left(\frac{c_0}{1-P_R} \right)^{i-1} \right] \end{array} \right]$$

$$= c_1 \left[\begin{array}{l} \frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} \\ -\sum_{x=3}^{\infty} (1-P_S)^{x-1} \sum_{i=2}^{x-1} \left[{c_0}^{T+x-2} \left(\frac{1-P_R}{c_0} \right)^{i-1} + (1-P_R)^{T+x-2} \left(\frac{c_0}{1-P_R} \right)^{i-1} \right] \end{array} \right]$$

$$= c_1 \left[\begin{array}{l} \frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} - \sum_{x=3}^{\infty} (1-P_S)^{x-1} \frac{{c_0}^{T+x-1}}{1-P_R} \sum_{i=2}^{x-1} \left(\frac{1-P_R}{c_0} \right)^{i} \\ -\frac{(1-P_R)^{T+x-1}}{c_0} \sum_{x=3}^{\infty} (1-P_S)^{x-1} \sum_{i=2}^{x-1} \left(\frac{c_0}{1-P_R} \right)^{i} \end{array} \right]$$

$$= c_1 \left[\frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} \right]$$

$$- c_1 \sum_{x=3}^{\infty} \frac{{c_0}^{T+x-1}(1-P_S)^{x-1}}{(1-P_R)} \left(\frac{\left[{c_0}^{x} + (1-P_R)^{x}\right](1-P_R)}{c_0^{x}\left[c_0 + 1 - P_R\right]} - \frac{1-P_R}{c_0} \right)$$

$$- c_1 \sum_{x=3}^{\infty} (1-P_S)^{x-1} \frac{(1-P_R)^{T+x-1}}{c_0} \times \left(\frac{\left[(1-P_R)^{x} + {c_0}^{x}\right]c_0}{(1-P_R)^{x}\left[c_0 + 1 - P_R\right]} - \frac{c_0}{1-P_R} \right)$$

We define a constant $c_2 = c_1\left[\frac{(1-P_S)^2 {c_0}^{T+1}}{1-[(1-P_S)c_0]^2} + \frac{(1-P_S)^2(1-P_R)^{T+1}}{1-[(1-P_S)(1-P_R)]^2}\right]$ to simplify:

$$= c_2 - c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{{c_0}^{T+x-1}}{1-P_R}\times\left(\frac{\left[{c_0}^x+(1-P_R)^x\right](1-P_R)}{c_0^x\left[c_0+1-P_R\right]}-\frac{1-P_R}{c_0}\right)$$

$$- c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{(1-P_R)^{T+x-1}}{c_0}\times\left(\frac{\left[(1-P_R)^x+{c_0}^x\right]c_0}{(1-P_R)^x\left[c_0+1-P_R\right]}-\frac{c_0}{1-P_R}\right)$$

$$= c_2 - c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{{c_0}^{T+x-1}}{1-P_R}\frac{\left[{c_0}^x+(1-P_R)^x\right](1-P_R)}{c_0^x\left[c_0+1-P_R\right]}$$

$$+ c_1\left(\frac{1-P_R}{c_0}\right)\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{{c_0}^{T+x-1}}{1-P_R} + c_1\left(\frac{c_0}{1-P_R}\right)\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{(1-P_R)^{T+x-1}}{c_0}$$

$$- c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}\frac{(1-P_R)^{T+x-1}}{c_0}\frac{\left[(1-P_R)^x+{c_0}^x\right]c_0}{(1-P_R)^x(c_0+1-P_R)}$$

$$= c_2 - c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}{c_0}^{T-1}\frac{{c_0}^x+(1-P_R)^x}{c_0+1-P_R} + c_1{c_0}^{T-1}\sum_{x=3}^{\infty}(1-P_S)^{x-1}{c_0}^{x-1}$$

$$- c_1\sum_{x=3}^{\infty}(1-P_S)^{x-1}(1-P_R)^{T-1}\frac{(1-P_R)^x+{c_0}^x}{c_0+1-P_R} + c_1(1-P_R)^{T-1}\sum_{x=3}^{\infty}(1-P_S)^{x-1}(1-P_R)^{x-1}$$

$$= c_2 - \frac{c_1{c_0}^{T-1}}{(c_0+1-P_R)(1-P_S)}\sum_{x=3}^{\infty}(1-P_S)^x\left[{c_0}^x+(1-P_R)^x\right] + c_1{c_0}^{T-1}\sum_{x=2}^{\infty}(1-P_S)^x{c_0}^x$$

$$- \frac{c_1(1-P_R)^{T-1}}{(c_0+1-P_R)(1-P_S)}\sum_{x=3}^{\infty}(1-P_S)^x\left[(1-P_R)^x+{c_0}^x\right] + c_1(1-P_R)^{T-1}\sum_{x=2}^{\infty}(1-P_S)^x(1-P_R)^x$$

$$= c_2 - \frac{c_1{c_0}^{T-1}}{(c_0+1-P_R)(1-P_S)}\sum_{x=3}^{\infty}(1-P_S)^x\left[{c_0}^x+(1-P_R)^x\right]$$

$$+ c_1{c_0}^{T}(1-P_S)\left[\frac{1}{1-(1-P_S)c_0}-1\right] - \frac{c_1(1-P_R)^{T-1}}{(c_0+1-P_R)(1-P_S)}\sum_{x=3}^{\infty}(1-P_S)^x\left[(1-P_R)^x+{c_0}^x\right]$$

$$+ c_1(1-P_R)^{T-1}\left[\frac{(1-P_S)(1-P_R)}{1-(1-P_S)(1-P_R)}-(1-P_S)(1-P_R)\right]$$

We define the following intermediate variables:

$$c_3 = c_1{c_0}^{T-1}\left[\frac{(1-P_S)c_0}{1-(1-P_S)c_0}-(1-P_S)c_0\right],\ c_4 = \frac{c_1{c_0}^{T-1}}{\left[c_0+1-P_R\right]\times(1-P_S)},$$

$$c_5 = c_1(1-P_R)^{T-1}\left[\frac{(1-P_S)(1-P_R)}{1-(1-P_S)(1-P_R)}-(1-P_S)(1-P_R)\right], c_6 = \frac{c_1(1-P_R)^{T-1}}{\left[c_0+1-P_R\right]\times(1-P_S)}$$

to simplify the Probability (incurring penalty due to Scenario Six) as:

$$= c_2 + c_3 + c_5 - c_4 \sum_{x=3}^{\infty} (1-P_S)^x \left[{c_0}^x + (1-P_R)^x \right] - c_6 \sum_{x=3}^{\infty} (1-P_S)^x \left[(1-P_R)^x + {c_0}^x \right]$$

$$= c_2 + c_3 + c_5 - (c_4 + c_6) \times \left[\frac{\left[(1-P_S)c_0\right]^3}{1-(1-P_S)c_0} + \frac{\left[(1-P_S)(1-P_R)\right]^3}{1-(1-P_S)(1-P_R)} \right]$$

$$= c_2 + c_3 + c_5 - \left(\frac{c_1 \left({c_0}^{T-1} + (1-P_R)^{T-1} \right)}{(c_0 + 1 - P_R)(1-P_S)} \right) \times \left[\frac{\left[(1-P_S)c_0\right]^3}{1-(1-P_S)c_0} + \frac{\left[(1-P_S)(1-P_R)\right]^3}{1-(1-P_S)(1-P_R)} \right]$$

$$= c_1 \left[\frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2 (1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} \right] + c_1 (1-P_R)^{T-1} \left[\frac{\left((1-P_S)(1-P_R)\right)^2}{1-(1-P_S)(1-P_R)} \right]$$

$$- \left(\frac{c_1 \left({c_0}^{T-1} + (1-P_R)^{T-1} \right)}{(c_0 + 1 - P_R)(1-P_S)} \right) \left(\frac{\left[(1-P_S)c_0\right]^3}{1-(1-P_S)c_0} + \frac{\left[(1-P_S)(1-P_R)\right]^3}{1-(1-P_S)(1-P_R)} \right) + c_1 {c_0}^{T-1} \left[\frac{\left((1-P_S)c_0\right)^2}{1-(1-P_S)c_0} \right]$$

$$= c_1 \left[\frac{(1-P_S)^2 {c_0}^{T+1}}{1-\left[(1-P_S)c_0\right]^2} + \frac{(1-P_S)^2 (1-P_R)^{T+1}}{1-\left[(1-P_S)(1-P_R)\right]^2} \right] + c_1 (1-P_R)^{T-1} \left[\frac{\left((1-P_S)(1-P_R)\right)^2}{1-(1-P_S)(1-P_R)} \right]$$

$$- \left(\frac{c_1 \left({c_0}^{T-1} + (1-P_R)^{T-1} \right)}{(c_0 + 1 - P_R)(1-P_S)} \right) \left(\frac{\left[(1-P_S)c_0\right]^3}{1-(1-P_S)c_0} + \frac{\left[(1-P_S)(1-P_R)\right]^3}{1-(1-P_S)(1-P_R)} \right) + c_1 {c_0}^{T-1} \left[\frac{\left((1-P_S)c_0\right)^2}{1-(1-P_S)c_0} \right]$$

(4.21) If $Z_t > 0$, and 0 otherwise.

Total probability of job incurring penalty under case two is (4.16) + (4.17) + (4.18) + (4.19) + (4.20) + (4.21) if $Z_t > 0$, and = (4.16) + (4.17) otherwise. The condition for processing a prospective job is: $R/C > (4.16) + (4.17) + (4.18) + (4.19) + (4.20) + (4.21)$ if $Z_t > 0$, and $R/C > (4.16) + (4.17)$ otherwise.

Case 3: $X_t = (\geq Q, 1, 0)$

Similar to what was done in Model Two, we divide the analysis to Case 3.1 and Case 3.2.

Case 3.1, $\lfloor Z_t Q^{-1} \rfloor = Z_t Q^{-1}$

If the serial processor finishes in x time units, for a job to not incur a penalty, a total of $Z_t Q^{-1} + 1$ "batch-removal events" have to occur in $x + T - 1$ time units. Batch-removal events could be in form of batch processor finishing a batch (minimum one time unit) OR the batch processor failing and then being repaired (minimum two time units). Each batch-removal event removes one batch in front of the batch processor. Let ${}^aC_b = a!/b!(a-b)!$, and let i be the number of batch-removal events due to processor failure and subsequent repair, then the

removal of $Z_t Q^{-1}+1$ batches corresponds to the factor: $\sum_{i=0}^{Z_t Q^{-1}+1} [P_B(1-P_F)]^{Z_t Q^{-1}+1-i} P_F{}^i P_R{}^i \left({}_{Z_t Q^{-1}+1}C_i\right)$.

The values *i* can assume may be limited by *x*. In particular, assuming *x* can only take values > $Z_t Q^{-1}+2-T$, then *i* can not reach *x* + *T* - *1* - $Z_t Q^{-1}+1$.

The set of batch-removal events requires a minimum of $Z_t Q^{-1}+1+i$ time units. If the removal of $Z_t Q^{-1}+1$ batches takes *j* time units, the remaining $j - Z_t Q^{-1} - 1 - i$ time units are assigned to either failures to finish processing the batch or to failures to repair the batch processor. Let *k* be the total number of failures to repair the processor, then to account for the remaining time events, we include the factor: $\sum_{k=0}^{\min(1,i)\times\left(j-Z_t Q^{-1}-1-i\right)} \left[(1-P_B)(1-P_F)\right]^{j-Z_t Q^{-1}-1-i-k} (1-P_R)^k$, where the values *k* may take are dependent on the value of *i*. If *i = 0*, then *k* has to be zero.

We determine how many possible ways are there to partition $j-Z_t Q^{-1}-1-i-k$ failures to process the batch into at most $Z_t Q^{-1}+1$ sets which correspond to the number of batch-removal events. We consider $j-i-k-1$ total items, with $j-Z_t Q^{-1}-1-i-k$ items belonging to one type and the remaining $Z_t Q^{-1}$ items of another type, and determine how many possible permutations can be made. One is subtracted from the number of batch-removal events, since only the possible combinations where the last "item" is a batch-removal event are valid. There are $\frac{(j-i-k-1)!}{(Z_t Q^{-1})!(j-Z_t Q^{-1}-1-i-k)!}$ possible permutations. The same method is used to determine how many possible ways are there to partition *k* failure to repair events into at most *i - 1* sets (assuming *i > 0*) to get $\frac{(k+i-1)!}{k!(i-1)!}$ permutations.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = Z_t Q^{-1}+1 = \lfloor Z_t Q^{-1} \rfloor + 1$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$, $c_7 = max\,(1, c_1 - T + 1)$. The probability of not incurring a penalty =

$$\sum_{x=c_7}^{\infty}\sum_{i=0}^{\min(x+T-1-c_1,c_1)}\sum_{j=c_1+i}^{x+T-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}\left[\frac{{c_5}^{x-1}P_s{c_2}^{c_1-i}{c_4}^{i}{c_0}^{j-c_1-i-k}{c_6}^{k}}{(c_1-1)!(c_1-i)!i!(j-c_1-i-k)!}\times\frac{c_1!(j-i-k-1)!(\max(0,k+i-1))!}{(\max(0,i-1))!k!}\right]$$

$$=\sum_{x=c_7}^{\infty}\sum_{i=0}^{\min(x+T-1-c_1,c_1)}\sum_{j=c_1+i}^{x+T-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}\left[\frac{{c_5}^{x-1}P_s{c_2}^{c_1-i}{c_4}^{i}{c_0}^{j-c_1-i-k}{c_6}^{k}}{(c_1-i)!i!(j-c_1-i-k)!}\times\frac{c_1(j-i-k-1)!(\max(0,k+i-1))!}{(\max(0,i-1))!k!}\right]$$

Probability of incurring a penalty = 1 - probability (not incurring a penalty)

The condition for processing a prospective job becomes:

$$R/C>1-\sum_{x=c_7}^{\infty}\sum_{i=0}^{\min(x+T-1-c_1,c_1)}\sum_{j=c_1+i}^{x+T-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}\left[{c_5}^{x-1}P_s{c_2}^{c_1-i}{c_4}^{i}{c_0}^{j-c_1-i-k}{c_6}^{k}c_1\times\frac{(j-i-k-1)!(\max(0,k+i-1))!}{(c_1-i)!i!(j-c_1-i-k)!(\max(0,i-1))!k!}\right]$$

(4.22)

Case 3.2: $Z_tQ^{-1}>\lfloor Z_tQ^{-1}\rfloor$

In Case 3.2, the number of batches in front of the prospective batch depends on when the first $\lfloor Z_tQ^{-1}\rfloor+1$ batches finish, relative to the time instance the serial processor finishes. If the first $\lfloor Z_tQ^{-1}\rfloor+1$ batches finish before the serial processor finishes (Scenario One), then the prospective job being processed by the serial processor miss joining the $\lfloor Z_tQ^{-1}\rfloor+2^{nd}$ batch. If the first $\lfloor Z_tQ^{-1}\rfloor+1$ batches are not finished before the serial processor finishes (Scenario Two), then the prospective job being processed by the serial processor will join the $\lfloor Z_tQ^{-1}\rfloor+2^{nd}$ batch.

Scenario One Development:

For a prospective job to incur a penalty under Scenario One, the following sequence of events needs to happen:

If the serial processor finishes in *x* time units, for a job to incur a penalty, a total of $\lceil Z_t/Q \rceil = \lfloor Z_t Q^{-1} \rfloor + 1$ batch-removal events have to occur in at most *x - 1* time units, while batch removal event number $\lceil Z_t/Q \rceil + 1$ does not finish until after *x - T - 1* time units.

We initially obtain the probability that $\lceil Z_t/Q \rceil$ batch-removal events occur in *x - 1* time units. The derivation is similar to that done in Case 3.1.

Let $c_0 = (1 - P_B)(1 - P_F)$, , $c_1 = \lceil Z_t/Q \rceil$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$.

Probability of $\lceil Z_t/Q \rceil$ events in at most *x - 1* time units = $\sum_{i=0}^{c_1} \sum_{j=c_1+i}^{x-1} \sum_{k=0}^{\min(1,i)\times(j-c_1-i)} {c_2}^{c_1-i} {c_4}^{i} {c_0}^{j-c_1-i-k} {c_6}^{k} \frac{c_1 (j-i-k-1)! (\max(0,k+i-1))!}{(c_1-i)!\, i!\, (j-c_1-i-k)!\, (\max(0,i-1))!\, k!}$

The remaining *T + x - j - 1* time units are composed of either failures to finish the batch and/or possibly a processor failure followed by failures to repair the processor. This implies the presence of the terms: $(c_0)^{T+x-j} + \sum_{l=0}^{T+x-j-1} {c_0}^{l} P_F {c_6}^{T+x-j-l-1}$.

The probability of incurring a penalty due to Scenario One, conditioned on the serial processor finishing in *x* time units, then becomes:

$$\sum_{i=0}^{c_1} \sum_{j=c_1+i}^{x-1} \sum_{k=0}^{\min(1,i)\times(j-c_1-i)} \left[\frac{{c_2}^{c_1-i} {c_4}^{i} {c_0}^{j-c_1-i-k} {c_6}^{k} c_1 (j-i-k-1)! (\max(0,k+i-1))!}{(c_1-i)!\, i!\, (j-c_1-i-k)!\, (\max(0,i-1))!\, k!} \times \left[{c_0}^{T+x-j-1} + \sum_{l=0}^{T+x-j-2} {c_0}^{l} P_F {c_6}^{T+x-j-l-2} \right] \right]$$

The probability of incurring a penalty due to Scenario One then becomes:

$$\sum_{x=c_1+1}^{\infty}\sum_{i=0}^{\min(c_1,x-c_1-1)}\sum_{j=c_1+i}^{x-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}\left[\begin{array}{l}\dfrac{c_1(j-i-k-1)!(\max(0,k+i-1))!}{(c_1-i)!i!(j-c_1-i-k)!(\max(0,i-1))!k!}\\ \times c_5^{\,x-1}P_s c_2^{\,c_1-i}c_4^{\,i}c_0^{\,j-c_1-i-k}c_6^{\,k}\\ \times\left[c_0^{\,T+x-j-1}+\sum_{l=0}^{T+x-j-2}c_0^{\,l}P_F c_6^{\,T+x-j-l-2}\right]\end{array}\right]$$

$$\sum_{x=c_1+1}^{\infty}\sum_{i=0}^{\min(c_1,x-c_1-1)}\sum_{j=c_1+i}^{x-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}\left[\begin{array}{l}\dfrac{c_1(j-i-k-1)!(\max(0,k+i-1))!}{(c_1-i)!i!(j-c_1-i-k)!(\max(0,i-1))!k!}\\ \times c_5^{\,x-1}P_s c_2^{\,c_1-i}c_4^{\,i}c_0^{\,j-c_1-i-k}c_6^{\,k}\\ \times\left[c_0^{\,T+x-j-1}+P_F\dfrac{c_6^{T+x-j-1}-c_0^{T+x-j-1}}{c_6-c_0}\right]\end{array}\right] \quad (4.23)$$

Scenario Two Development:

If the serial processor finishes in x time units, for a job to not incur a penalty, a total of $\lfloor Z_t Q^{-1}\rfloor+1$ batch-removal events have to occur in $x + T - 1$ time units. The derivation of the probability of incurring a penalty under Scenario Two is thus identical to the derivation in Case 3.1.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = \lfloor Z_t Q^{-1}\rfloor+1$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$, $c_7 = \max(1, c_1 - T + 1)$.

The probability of not incurring a penalty under Scenario Two, conditioned on the prospective job finishing at x time units =

$$\sum_{i=0}^{c_1}\sum_{j=c_1+i}^{x-1}\sum_{k=0}^{\min(1,i)\times(j-c_1-i)}c_2^{\,c_1-i}c_4^{\,i}c_0^{\,j-c_1-i-k}c_6^{\,k}\frac{(j-i-k)!(k+i)!}{(c_1-i)!i!(j-c_1-i-k)!i!k!}$$

The probability of not incurring a penalty under Scenario Two is =

$$\sum_{x=c_7}^{\infty}\sum_{i=0}^{\min(c_1,x+T-1-c_1)}\sum_{j=c_1+i}^{x+T-1}\sum_{k=0}^{\min(0,i)\times(j-c_1-i)}\left[\frac{c_5^{\,x-1}P_s c_2^{\,c_1-i}c_4^{\,i}c_0^{\,j-c_1-i-k}c_6^{\,k}(j-i-k)!(k+i)!}{(c_1-i)!i!(j-c_1-i-k)!i!k!}\right]$$

Probability of incurring a penalty due to Scenario Two = 1 - probability (not incurring a penalty)

$$=1-\sum_{x=c_7}^{\infty}\sum_{i=0}^{\min(c_1,x+T-1-c_1)}\sum_{j=c_1+i}^{x+T-1}\sum_{k=0}^{\min(0,i)\times(j-c_1-i)}\left[\frac{c_5^{\,x-1}P_s c_2^{\,c_1-i}c_4^{\,i}c_0^{\,j-c_1-i-k}c_6^{\,k}(j-i-k)!(k+i)!}{(c_1-i)!i!(j-c_1-i-k)!i!k!}\right] \quad (4.24)$$

Since Scenarios One and Two are mutually exclusive, the total probability of incurring a penalty under Case 3.2 is the sum of the probability of incurring penalty under two scenarios. Under Case 3.2, we process a job only if : $R/C > (4.22) + (4.23)$

4.8.2.1.2 When the Batch Processor is Initially Down

If $S_t = -1$, then the batch processor is down, and has to be repaired before any jobs can be processed. We only have two cases to consider:

Case One: X_t = (0 to Q - 1, -1)

Three scenarios can cause the prospective job to accumulate waiting time greater than the job's processing time window.

- *Scenario One:* Repair of batch processor does not occur until after the prospective job's processing time window is reached.

Probability (job will incur penalty due to Scenario One | serial processor took x time units)

$= (1 - P_R)^{x+T-1}$

Probability (job will incur penalty due to Scenario One)

$$= \sum_{x=1}^{\infty} (1 - P_S)^{x-1} P_S (1 - P_R)^{x+T-1} \text{ or } = P_S (1 - P_R)^T / (P_S + P_R - P_R P_S) \qquad (4.25)$$

- *Scenario Two:* batch processor gets repaired before the serial processor finishes processing. New batch is loaded and the new batch is not finished before the processing time window of the prospective job is reached. This scenario is only possible if $Z_t > 0$.

Let i be the number of time units before the repair event occurs, then the probability of penalty incurred under Scenario Two conditioned on serial processor finishing at x

$= \sum_{i=0}^{x-2} (1 - P_R)^i P_R \left[(1 - P_F)(1 - P_B)\right]^{T+x-i-2}$ if $x > 1$, and 0 otherwise

So probability of penalty incurred due to Scenario Two is:

$$\sum_{x=2}^{\infty} \sum_{i=0}^{x-2} (1 - P_S)^{x-1} P_S (1 - P_R)^i P_R \left[(1 - P_F)(1 - P_B)\right]^{T+x-i-2}$$

Let $c_0 = (1-P_F)(1-P_B)$, then probability of penalty incurred due to Scenario Two is:

$$\sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_S(1-P_R)^i P_R {c_0}^{T+x-i-2}$$

$$=\sum_{x=2}^{\infty}(1-P_S)^{x-1}P_S P_R {c_0}^{T+x-2}\sum_{i=0}^{x-2}\left[\frac{1-P_R}{c_0}\right]^i$$

$$=\sum_{x=2}^{\infty}(1-P_S)^{x-1}P_S P_R {c_0}^{T+x-2}\left[\frac{{c_0}^{x-1}-(1-P_R)^{x-1}}{{c_0}^{x-2}\left[c_0-(1-P_R)\right]}\right]$$

$$=\frac{P_S P_R {c_0}^{T}}{\left[c_0-(1-P_R)\right]}\sum_{x=0}^{\infty}(1-P_S)^x\left[{c_0}^{x}-(1-P_R)^x\right]$$

$$=\frac{P_S P_R {c_0}^{T}}{\left[c_0-(1-P_R)\right]}\left[\frac{1}{1-(1-P_S)c_0}-\frac{1}{1-(1-P_S)(1-P_R)}\right]$$

$$=\frac{P_S P_R {c_0}^{T}(1-P_S)}{\left[c_0-(1-P_R)\right]}\left[\frac{c_0-(1-P_R)}{\left[1-(1-P_S)c_0\right]\times\left[1-(1-P_S)(1-P_R)\right]}\right]$$

$$=\left[\frac{(1-P_S)P_S P_R\left((1-P_F)(1-P_B)\right)^T}{\left[1-(1-P_S)(1-P_F)(1-P_B)\right]\times\left[1-(1-P_S)(1-P_R)\right]}\right] \qquad (4.26)$$

If $Z_t > 0$, and 0 otherwise.

- *Scenario Three:* batch processor gets repaired before the serial processor finishes processing. New batch is loaded and the batch processor fails again, and does not get repaired until after the serial processor finishes the prospective job. This scenario is not possible if $Z_t = 0$.

Let j be the number of time units spent unsuccessfully processing the new batch, then the probability of penalty incurred conditioned on serial processor finishing at x

$$=\sum_{i=0}^{x-2}\sum_{j=0}^{x+T-i-3}P_R\left[(1-P_B)(1-P_F)\right]^j P_F(1-P_R)^{T+x-3-j}$$ if $x > 1$, and zero otherwise.

Let $c_0 = (1-P_F)(1-P_B)$, then probability of penalty incurred due to Scenario Three is:

$$\sum_{x=2}^{\infty}\sum_{i=0}^{x-2}\sum_{j=0}^{x+T-i-3}(1-P_S)^{x-1}P_S P_R {c_0}^{j}P_F(1-P_R)^{T+x-3-j}$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}(1-P_S)^{x-1}P_SP_RP_F(1-P_R)^{T+x-3}\sum_{j=0}^{x+T-i-3}\left[c_0 \big/ 1-P_R\right]^j$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}P_F(1-P_R)^{T+x-3}(1-P_S)^{x-1}P_SP_R\frac{1-\left[c_0 \big/ 1-P_R\right]^{x+T-i-2}}{1-\left[c_0 \big/ 1-P_R\right]}$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}P_F(1-P_R)^{T+x-3}(1-P_S)^{x-1}P_SP_R\frac{(1-P_R)^{x+T-i-2}-{c_0}^{x+T-i-2}}{(1-P_R)^{x+T-i-3}\left[1-P_R-c_0\right]}$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}P_F(1-P_R)^{i}(1-P_S)^{x-1}P_SP_R\frac{(1-P_R)^{x+T-i-2}-{c_0}^{x+T-i-2}}{1-P_R-c_0}$$

$$= \sum_{x=2}^{\infty}\sum_{i=0}^{x-2}P_F(1-P_S)^{x-1}P_SP_R(1-P_R)^{i}\frac{(1-P_R)^{x+T-i-2}-{c_0}^{x+T-i-2}}{1-P_R-c_0}$$

$$= \sum_{x=2}^{\infty}\frac{P_F(1-P_S)^{x-1}P_SP_R}{1-P_R-c_0}\sum_{i=0}^{x-2}\left[(1-P_R)^{x+T-2}-c_0^{x+T-2}\left[\frac{1-P_R}{c_0}\right]^i\right]$$

$$= \sum_{x=2}^{\infty}\frac{P_F(1-P_S)^{x-1}P_SP_R}{1-P_R-c_0}\left[(x-1)(1-P_R)^{x+T-2}-c_0^{x+T-2}\sum_{i=0}^{x-2}\left[\frac{1-P_R}{c_0}\right]^i\right]$$

$$= \sum_{x=2}^{\infty}\frac{P_F(1-P_S)^{x-1}P_SP_R}{1-P_R-c_0}\left[(x-1)(1-P_R)^{x+T-2}-c_0^{x+T-2}\frac{{c_0}^{x-1}-(1-P_R)^{x-1}}{{c_0}^{x-2}\left[c_0-(1-P_R)\right]}\right]$$

$$= \frac{P_FP_SP_R}{1-P_R-c_0}\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[(x-1)(1-P_R)^{x+T-2}-c_0^{T}\frac{{c_0}^{x-1}-(1-P_R)^{x-1}}{\left[c_0-(1-P_R)\right]}\right]$$

$$= \frac{P_FP_SP_R}{(-P_R-c_0}\sum_{x=2}^{\infty}\left[(x-1)(1-P_R)^{x+T-2}(1-P_S)^{x-1}-c_0^{T}(1-P_S)^{x-1}\frac{{c_0}^{x-1}-(1-P_R)^{x-1}}{c_0-(1-P_R)}\right]$$

$$= \frac{P_FP_SP_R}{1-P_R-c_0}\left[\begin{array}{l}(1-P_R)^{T-1}\sum_{x=2}^{\infty}(x-1)(1-P_R)^{x-1}(1-P_S)^{x-1}\\ -\frac{c_0^{T}}{c_0-(1-P_R)}\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[{c_0}^{x-1}-(1-P_R)^{x-1}\right]\end{array}\right]$$

$$= \left[\frac{P_FP_SP_R}{1-P_R-c_0}(1-P_R)^{T-1}\sum_{x=2}^{\infty}(x-1)(1-P_R)^{x-1}(1-P_S)^{x-1}\right]$$

$$-\left[\frac{P_FP_SP_R}{1-P_R-c_0}\times\frac{c_0^{T}}{c_0-(1-P_R)}\sum_{x=2}^{\infty}(1-P_S)^{x-1}\left[{c_0}^{x-1}-(1-P_R)^{x-1}\right]\right]$$

$$= \frac{P_FP_SP_R(1-P_R)^{T-1}}{1-P_R-c_0}\sum_{x=0}^{\infty}x(1-P_R)^{x}(1-P_S)^{x}$$

$$+\frac{P_FP_SP_Rc_0^{T}}{\left[1-P_R-c_0\right]^2}\sum_{x=0}^{\infty}(1-P_S)^{x}\left[{c_0}^{x}-(1-P_R)^{x}\right]$$

$$= \left[\frac{P_F P_S P_R (1-P_R)^{T-1}(1-P_R)(1-P_S)}{(1-P_R-c_0)\left[1-(1-P_R)(1-P_S)\right]^2} \right] + \left[\frac{P_F P_S P_R c_0^T}{\left[1-P_R-c_0\right]^2} \sum_{x=0}^{\infty} (1-P_S)^x \left[{c_0}^x - (1-P_R)^x \right] \right]$$

$$= \frac{P_F P_S P_R (1-P_R)^T (1-P_S)}{(1-P_R-c_0)\left[1-(1-P_R)(1-P_S)\right]^2}$$

$$+ \frac{P_F P_S P_R c_0^T}{\left[1-P_R-c_0\right]^2} \left[\frac{1}{1-(1-P_S)c_0} - \frac{1}{1-(1-P_S)(1-P_R)} \right]$$

$$= \frac{P_F P_S P_R (1-P_S)}{1-P_R-c_0} \left[\begin{array}{l} \dfrac{(1-P_R)^T}{\left[1-(1-P_R)(1-P_S)\right]^2} \\ + \dfrac{c_0^T \left(c_0-(1-P_R)\right)}{\left[1-P_R-c_0\right]\left[1-(1-P_S)c_0\right]\left[1-(1-P_S)(1-P_R)\right]} \end{array} \right]$$

$$= \frac{P_F P_S P_R (1-P_S)}{\left[1-P_R-c_0\right] \times \left[1-(1-P_R)(1-P_S)\right]} \left[\frac{(1-P_R)^T}{1-(1-P_R)(1-P_S)} - \frac{c_0^T}{\left[1-(1-P_S)c_0\right]} \right]$$

(4.27) if $Z_t > 0$, and 0 otherwise.

Total probability of job incurring penalty = (4.25) + (4.26) + (4.27) if $Z_t > 0$, and = (4.25) otherwise. If $Z_t > 0$, we process a job if and only if $R/C > (4.25) + (4.26) + (4.27)$. If $Z_t > 0$, we process a job if and only if $R/C > (4.25)$.

Case Two: $X_t = (> Q - 1, -1)$

We divide the analysis to Case 2.1 and Case 2.2.

Case 2.1: $Z_t Q^{-1} = \lfloor Z_t Q^{-1} \rfloor$

If the serial processor finishes in x time units, for a job to not incur a penalty, a repair event followed by a total of $Z_t Q^{-1}$ batch-removal events have to occur in $x + T - 1$ time units. Let ${}^aC_b = a!/b!(a-b)!$, and let i be the number of failure-repair batch-removal events that occur, then we would have the probability of the job not incurring a penalty with the following factor: $\sum_{i=0}^{Z_t Q^{-1}} \left[(P_B)(1-P_F)\right]^{Z_t Q^{-1}-i} (P_F)^i (P_R)^{i+1} \left({}^{Z_t Q^{-1}}C_i \right)$

There is an additional processor repair event compared to the number of failure events, to take into consideration the need to initially repair the batch processor.

This set of batch-removal events (including the initial repair event) requires a minimum of $Z_t Q^{-1} + 1 + i$ time units. If the batches are removed in j time units, the remaining $j - Z_t Q^{-1} - 1 - i$ time units which are used by either failures to finish processing the batch or by failures to repair the batch processor. Let k be the total number of failures to repair the processor, then the factor $\sum_{k=0}^{j-Z_t Q^{-1}-1-i} \left[(1-P_B)(1-P_F)\right]^{j-Z_t Q^{-1}-1-i-k} (1-P_R)^k$ takes into account the number of excess time units that do not result in a batch being removed.

The number of possible permutations for the non-batch-removal events is obtained in a manner similar to that in Case 3.1 (when the batch processor is initially up).

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = Z_t Q^{-1} = \lfloor Z_t Q^{-1} \rfloor$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$, $c_7 = max(1, c_1 - T + 2)$. Probability of not incurring a penalty under Case 2.1 =

$$\sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(c_1, x+T-2-c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-1-i} \left[\frac{{c_5}^{x-1} P_s {c_2}^{c_1-i} {c_4}^{i} P_R {c_0}^{j-c_1-i-k-1} {c_6}^{k} c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \right]$$

The condition for processing a job then becomes:

$$R/C > 1 - \sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(c_1, x+T-2-c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-1-i} \left[\begin{array}{l} \dfrac{P_R P_s c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \\ \times {c_5}^{x-1} {c_2}^{c_1-i} {c_4}^{i} {c_0}^{j-c_1-i-k-1} {c_6}^{k} \end{array} \right] \quad (4.28)$$

Case 2.2: $Z_t Q^{-1} > \lfloor Z_t Q^{-1} \rfloor$

In Case 2.2, the number of batches in front of the prospective batch depends on when the first $\lfloor Z_t Q^{-1} \rfloor$ batches finish, relative to the serial processor. If the first repair and the $\lfloor Z_t Q^{-1} \rfloor$ batches happen before the serial processor finishes (Scenario One), then the prospective job being processed by the serial processor will miss the $\lfloor Z_t Q^{-1} \rfloor + 1^{st}$ batch. If the

first $\lfloor Z_t Q^{-1} \rfloor$ batches are not finished before the serial processor finishes (Scenario Two), then the prospective job being processed by the serial processor will join the $\lfloor Z_t Q^{-1} \rfloor + 1^{st}$ batch.

Scenario One Development:

A prospective job incurs a processing time window penalty only if the following sequence of events occurs:

If the serial processor finishes in *x* time units, a repair followed by $\lfloor Z_t Q^{-1} \rfloor$ batch removal events have to occur in *x - 1* time units, with batch-removal event number $\lfloor Z_t Q^{-1} \rfloor + 1$ not finishing by *x + T - 1* time units.

The first portion requires a repair event followed by $\lfloor Z_t Q^{-1} \rfloor$ batch-removal events to occur in *x - 1* time units. This is similar to Case 2.1.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = \lfloor Z_t Q^{-1} \rfloor$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$

Probability of a repair and $\lfloor Z_t Q^{-1} \rfloor$ batch-removal events in *x - 1* time units =

$$\sum_{i=0}^{c_1} \sum_{j=c_1+i+1}^{x-1} \sum_{k=0}^{j-c_1-1-i} {c_2}^{c_1-i} {c_4}^{i} P_R {c_0}^{j-c_1-1-i-k} {c_6}^{k} \frac{c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!}$$

The remaining *T + x - j - 1* time units are composed of either failures to finish the batch and/or a processor failure followed by failures to repair the processor. This implies the presence of the terms: ${c_0}^{T+x-j-1} + \sum_{l=0}^{T+x-j-2} {c_0}^{l} P_F {c_6}^{T+x-j-l-2}$

The probability of incurring a penalty due to Scenario One becomes:

$$\sum_{x=c_1+2}^{\infty} \sum_{i=0}^{\min(c_1, x-c_1-2)} \sum_{j=c_1+i+1}^{x-1} \sum_{k=0}^{j-c_1-i-1} \left[\begin{array}{l} \frac{{c_5}^{x-1} {c_2}^{c_1-i} {c_0}^{j-c_1-1-i-k} {c_6}^{k} c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \\ \times {c_4}^{i} P_R P_s \left[{c_0}^{T+x-j-1} + \sum_{l=0}^{T+x-j-2} {c_0}^{l} P_F {c_6}^{T+x-j-l-2} \right] \end{array} \right]$$

$$\sum_{x=c_1+2}^{\infty} \sum_{i=0}^{\min(c_1, x-c_1-2)} \sum_{j=c_1+i+1}^{x-1} \sum_{k=0}^{j-c_1-i-1} \left[\begin{array}{l} \frac{{c_5}^{x-1} {c_2}^{c_1-i} {c_0}^{j-c_1-1-i-k} {c_6}^{k} (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \\ \times {c_4}^{i} c_1 P_s P_R \left[{c_0}^{T+x-j-1} + P_F \frac{{c_0}^{T+x-j-1} - {c_6}^{T+x-j-1}}{c_0 - c_6} \right] \end{array} \right] \quad (4.29)$$

Scenario Two Development:

If the serial processor finishes in x time units, for a job to not incur a penalty, a repair event followed by $\lfloor Z_t Q^{-1} \rfloor$ batch-removal events have to occur in $x + T - 1$ time units. This is identical to Case 2.1.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = \lfloor Z_t Q^{-1} \rfloor$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$, $c_7 = \max(1, c_1 - T + 2)$. Probability of not incurring a penalty due to Scenario Two =

$$\sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-2-c_1,c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-i-1} \left[\frac{{c_5}^{x-1} P_s {c_2}^{c_1-i} {c_4}^{i} P_R {c_0}^{j-c_1-1-i-k} {c_6}^{k} c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \right]$$

Probability of incurring a penalty due to Scenario Two =

$$1 - \sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-2-c_1,c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-i-1} \left[\frac{{c_5}^{x-1} P_s {c_2}^{c_1-i} {c_4}^{i} P_R {c_0}^{j-c_1-1-i-k} {c_6}^{k} c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \right]$$

(4.30)

Equation (4.30) is identical to (4.28). Since Scenarios One and Two are independent, the total probability of incurring a penalty is the sum of (4.29) and (4.30). Under Case 2.1, we process a job if and only if $R/C > (4.29) + (4.30)$.

4.8.2.2 When the Batch Processor is under Full-Batch Policy

Similar to Model Two, the probability of a prospective job incurring a penalty is the probability that either of two events occur:

- arrival of required jobs to form a full batch takes at least T time units (event T_A), and
- processing time of current batch at batch processor is at least T time units more than processing time of prospective job at serial processor (event T_B)

The two events are independent of each other and we process if and only if: $R/C > P(T_B(X_t)) + (1 - P(T_B(X_t))) \times P(T_A(X_t))$

The derivation of the probability of incurring a penalty due to the need to wait for additional job arrivals $P(T_A)$ is unaffected by the reliability at the batch processor, and so the derivation is

no longer repeated. Only the derivation for the probability of incurring a penalty due to the batch processor unavailability $P(T_B)$ is shown.

4.8.2.2.1 When the Batch Processor is Initially Up

Case One: X_t = *(0 to Q - 1,0)*

The waiting time of prospective job = time to have $Q - Z_t - 1$ arrivals. This is identical to Case one of Model Two.

We process if and only if: $$R/C > 1 - \sum_{i-Q-Z_t-1}^{T-1} {}^{i-1}C_{Q-Z_t-2}(1-P_S)^{i-Q+Z_t+1} P_S^{Q-Z_t-1} \quad (4.31)$$

if $Z_t < Q - 1$, and we always process if $Z_t = Q - 1$.

Case Two: X_t = *(0 to Q - 1,1)*

The prospective job can incur a penalty due to batch processor unavailability in two distinct ways.

Scenario One: batch processor does not fail, but also does not finish processing until the prospective job reaches its processing time window. The derivation of this expression is identical to that of (4.16).

Probability (job will incur penalty due to Scenario One)

$$= \sum_{x=1}^{\infty} (1-P_S)^{x-1} P_S \left[(1-P_B)(1-P_F)\right]^{T+x-1}$$
$$= P_S \left[(1-P_B)(1-P_F)\right]^T / 1-(1-P_S)(1-P_B)(1-P_F) \quad (4.32).$$

Scenario Two: batch processor fails while attempting to process the current batch, and is not repaired until the prospective job reaches its processing time window. The derivation of this expression is identical to that of (4.17).

The probability of Scenario Two occurring is:

$$= \frac{P_F P_s}{(1-P_R-(1-P_F)(1-P_B))} \left[\frac{(1-P_R)^T}{1-(1-P_s)(1-P_R)} - \frac{\left((1-P_F)(1-P_B)\right)^T}{1-(1-P_s)(1-P_F)(1-P_B)} \right] \quad (4.33)$$

Thus, we should process a job if and only if:

$R/C > P(T_B(X_t)) + (1-P(T_B(X_t))) \times P(T_A(X_t))$ where $P(T_B(X_t)) = (4.31) + (4.32)$.

Case Three: X_t = *(> Q - 1, 1)*

If serial processor finishes in x time units, for a job to not incur a penalty, a total of $\lfloor Z_t Q^{-1} \rfloor + 1$ batch-removal events have to occur in $x + T - 1$ time units. The derivation is similar to that in Case 3.1 when the batch processor is initially up and is under the no-idling policy.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = \lfloor Z_t Q^{-1} \rfloor + 1$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$,

$c_7 = max\ (c_1 - T + 1,\ 1)$

Probability of not incurring a penalty due to batch processor unavailability =

$$\sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-1-c_1,c_1)} \sum_{j=c_1+i}^{x+T-1} \sum_{k=0}^{\min(1,i)\times(j-c_1-i)} \left[\begin{array}{l} \dfrac{c_1 (j-i-k-1)!\,(\max(0,k+i-1))!}{(c_1-i)!\,i!\,(j-c_1-i-k)!\,(\max(0,i-1))!\,k!} \\ \times {c_5}^{x-1} P_s {c_2}^{c_1-i} {c_4}^{i} {c_0}^{j-c_1-i-k} {c_6}^{k} \end{array} \right]$$

So the probability of incurring a penalty = 1 – probability of not incurring a penalty.

Thus, we should process a job if and only if:

$$R/C > P(T_B(X_t)) + (1 - P(T_B(X_t))) \times P(T_A(X_t)) \text{ where}$$

$$P(T_B(X_t)) =$$

$$1 - \sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-1-c_1,c_1)} \sum_{j=c_1+i}^{x+T-1} \sum_{k=0}^{\min(1,i)\times(j-c_1-i)} \left[\begin{array}{l} {c_5}^{x-1} P_s {c_2}^{c_1-i} {c_4}^{i} {c_0}^{j-c_1-i-k} {c_6}^{k} \times \\ \dfrac{c_1 (j-i-k-1)!\,(\max(0,k+i-1))!}{(c_1-i)!\,i!\,(j-c_1-i-k)!\,(\max(0,i-1))!\,k!} \end{array} \right] \quad (4.34)$$

4.8.2.2.2 When the Batch Processor is Initially Down

There are only two cases to consider when the batch processor is initially down.

Case One: X_t = *(0 to Q - 1,-1)*

Probability of repair taking at least $x + T$ units, if serial processor finished in x time units is derived in an identical manner to (4.24).

Thus, we should process a job only if: $R/C > P(T_B(X_t)) + (1 - P(T_B(X_t))) \times P(T_A(X_t))$

where $P(T_B(X_t)) = P_S (1 - P_R)^T / 1 - (1 - P_S)(1 - P_R)$ (4.35)

Case Two: X_t = *(> Q - 1,-1)*

The probability that repair and production of $\lfloor Z_t Q^{-1} \rfloor$ batches caused delay of at least T units is derived under Case 2.1, when the batch processor is initially down and under no-idling policy.

Let $c_0 = (1 - P_B)(1 - P_F)$, $c_1 = \lfloor Z_t Q^{-1} \rfloor$, $c_2 = P_B(1 - P_F)$, $c_4 = P_F P_R$, $c_5 = 1 - P_S$, $c_6 = 1 - P_R$, $c_7 = max(c_1 - T + 2, 1)$

Probability (not incurring a penalty) due to batch processor unavailability =

$$\sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-2-c_1,c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-1-i} \left[\frac{{c_5}^{x-1} P_S P_R {c_0}^{j-c_1-1-i-k} {c_6}^{k} {c_2}^{c_1-i} {c_4}^{i} c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \right]$$

Probability (incurring a penalty) = 1 - Probability (not incurring a penalty)

Thus, we should process a job if and only if:

$R/C > P(T_B(X_t)) + (1 - P(T_B(X_t))) \times P(T_A(X_t))$ where

$$P(T_B(X_t)) = 1 - \sum_{x=c_7}^{\infty} \sum_{i=0}^{\min(x+T-2-c_1,c_1)} \sum_{j=c_1+i+1}^{x+T-1} \sum_{k=0}^{j-c_1-1-i} \left[\begin{array}{l} \frac{c_1 (j-i-k-2)!(k+i)!}{(c_1-i)!i!(j-c_1-1-i-k)!i!k!} \\ \times {c_5}^{x-1} P_S P_R {c_0}^{j-c_1-1-i-k} {c_6}^{k} {c_2}^{c_1-i} {c_4}^{i} \end{array} \right] \quad (4.36)$$

Having derived the expressions to determine the probability of incurring a penalty for each state, we analyze the characteristics of the probability curves, and hence the optimal solutions, in the succeeding section.

4.8.3 Model Three Discussion

This section characterizes the optimal policy derived from equations in Section 4.8.2.

4.8.3.1 Comparison of Optimal Policy for Different Initial Batch Processor States

We compare the characteristics of the optimal policy when the batch processor is initially up, versus the optimal policy when the batch processor is initially down.

4.8.3.1.1 Batch processor is under no-idling policy

We compare the probabilities of incurring a penalty when the batch processor is initially up or down. We divide the comparison into two stages: the first stage is when less than a full batch

is in front of the batch processor and the batch processor is unavailable (Case Two of Section 4.8.2.1.1 when the batch processor is initially busy, and Case One of Section 4.8.2.1.2 when the batch processor is initially down), and the second stage is when at least a full batch is in front of the batch processor (Case Three of Section 4.8.2.1.1 when the batch processor is initially busy, and Case Two of Section 4.8.2.1.2 when the batch processor is initially down.) We ignore Case One when the batch processor is idle, since the probability of incurring a penalty is zero.

When the batch processor buffer is empty and the batch processor is not idle, it can be algebraically shown that (4.25) < (4.16) if $(1 - P_R) < (1 - P_B)(1 - P_F)$. Thus, if $(1 - P_R) < (1 - P_B)(1 - P_F)$ and $Z_t = 0$, the probability of incurring a penalty is higher when the batch processor is initially busy, compared to the probability when the batch processor is initially down. This condition can be intuitively explained. if $(1 - P_R) < (1 - P_B)(1 - P_F)$, then $P_R > 1-(1 - P_B)(1 - P_F)$, where P_R is the probability of the batch processor getting repaired (which allows the batch processor to process a new batch), while $1-(1 - P_B)(1 - P_F)$ is the probability that the batch processor neither finished a batch (with probability $P_B(1 - P_F)$) nor breaks down (with probability P_F), both of which cause the current batch being processed to be removed from the processor. Unfortunately, if $(1 - P_R) > (1 - P_B)(1 - P_F)$, then it is not always true that the probability of incurring a penalty is greater when the batch processor is initially down, since there exists another failure mode (Equation (4.17)) when the batch processor is initially busy.

We suspect that the probability of incurring a penalty is greater when the batch processor is initially busy whenever $(1 - P_R) < (1 - P_B)(1 - P_F)$, for the same number of jobs queued at the batch processor. The complexity of the derived expressions makes it difficult to show that this is true. Instead, we run numerical experiments to test our hypothesis. Table 4.5 contains the various parameter values used in the numerical experiment.

TABLE 4.5: EXPERIMENTAL SETUP TO COMPARE PROBABILITIES OF INCURRING A PENALTY ACCORDING TO INITIAL BATCH PROCESSOR STATE

Initial batch processor state	Processing time window length	Probability of batch processor finishing batch	Probability of serial processor finishing job	Batch processor reliability	Multiplier
Busy	10, 20	0.3, 0.6, 0.9	0.3, 0.6, 0.9	0.3, 0.6, 0.9	0.3, 0.6, 0.9, 1.2
Down	10, 20	0.3, 0.6, 0.9	0.3, 0.6, 0.9	0.3, 0.6, 0.9	0.3, 0.6, 0.9, 1.2

The batch processor is under no-idling policy, and the size is kept constant at two jobs. The probability of incurring a penalty is determined for batch processor buffer levels from 0 to 20 (20 jobs correspond to 10 full batches in front of the batch processor). If the batch processor is up, it is assumed that the batch processor is initially busy. The multiplier found in the last column of Table 4.5 is used to determine the probability of repairing the batch processor, using the equation: $P_R = max(0.05, 1 - multiplier \times (1 - P_B)(1 - P_F))$.

In all 216 experiments, with each experiment lasting for 20 different queue lengths, the probability of incurring a penalty when the batch processor is initially busy is never lower than when the batch processor is initially busy, if $(1 - P_R) < (1 - P_B)(1 - P_F)$. This supports the hypothesis developed.

4.8.3.1.2 Batch processor is under full-batch policy

We wish to characterize how the optimal policy differs when the batch processor is up or down, while being operated under the full-batch policy. We thus compare the probabilities of incurring a penalty under Case Two of Section 4.8.2.2.1 when the processor is up, with Case One of Section 4.8.2.2.2 when the processor is down, and also compare Case Three of Section 4.8.2.2.1 when the processor is up, with Case Two of Section 4.8.2.2.2 when the processor is down. Furthermore, since the probability of incurring a penalty due to lack of future arrivals $P(T_A)$ would be identical regardless of batch processor state, we can restrict the comparison to the expected waiting time due to batch processor unavailability.

Let the probability of incurring a penalty when the batch processor is initially up be $P(up) = P(T_B(up)) + (1 - P(T_B(up))) \times P(T_A(up))$, and the probability of incurring a penalty when the batch processor is initially down is $P(down) = P(T_B(down)) + (1 - P(T_B(down))) \times P(T_A(down))$. If $P(T_A(up)) = P(T_A(down))$, then $P(up) \geq P(down)$ if and only if $P(T_B(up)) \geq P(T_B(down))$ and $P(up) \leq P(down)$ if and only if $P(T_B(up)) \leq P(T_B(down))$.

When there are 0 to $Q - 1$ jobs in front of the batch processor, and the batch processor is unavailable (either processing or down):

There can be two ways a prospective job can incur a penalty due to batch processor unavailability when the batch processor is initially up. We initially compare (4.32) with the probability of incurring a penalty when the batch processor is down (4.35). If $(1 - P_R) > (1 - P_B)(1 - P_F)$, then the probability of incurring a penalty is higher when the batch processor is down. This does not take into account (4.33). Thus, we cannot establish convenient conditions when the probability of incurring a penalty when the batch processor is initially down is larger. On the other hand, if $(1 - P_R) < (1 - P_B)(1 - P_F)$, then the probability of incurring a penalty is lower when the batch processor is down. This is intuitive, as $(1 - P_R)$ is the probability of not resolving the batch processor unavailability in that particular time step when the batch processor is down, while $(1 - P_B)(1 - P_F)$ is the probability of not resolving the batch processor unavailability when the batch processor is busy. If it is harder to use up a time instance not repairing a batch processor, than it is to use up a time instance not processing a batch, then it would take a relatively shorter amount of time to free up the batch processor if the batch processor was initially down. This means that the threshold when the batch processor has less than Q jobs in front of it and the batch processor is unavailable is lower when the batch processor is down, if $(1 - P_R) < (1 - P_B)(1 - P_F)$.

When there is at least a full batch in front of the batch processor, the nature of the summations makes it difficult to compare the expressions directly. We can run numerical experiments to determine whether the observation for the previous case is also true. The experimental setup is identical to that detailed in Table 4.5, except that the batch processor is under full-batch policy. In all 216 experiments, with each experiment lasting for 20 different queue lengths, the probability of incurring a penalty when the batch processor is currently busy is at least as large as the probability of incurring a penalty when the batch processor is currently busy whenever $(1 - P_R) < (1 - P_B)(1 - P_F)$, supporting the initial hypothesis.

The comparison results suggest it is profitable to force the processor to fail (which removes one batch) and then repair the processor, if the repair probability is high. One possible

consequence of these results is in the control of systems where the batch processor can be preempted. Preemption would then be equivalent to forcing the processor to fail and subsequently repairing it. The expected benefit of preemption (the sum of reaching the processing time window penalties multiplied by the difference in the probabilities of incurring a penalty for all jobs in front of the batch processor) can be compared with the preemption costs to determine the appropriate action.

4.8.3.2 Comparison of Optimal Policy for Different Batch Processor Policies

In comparing optimal policies when the batch processor is under different policies, there is little difference between the inferences we have made in Model Two and those that we have observed in Model Three.

Similar to what can be done in Models One and Two, the summary performance measures of the system under a threshold policy can be modeled as a Markov chain, even if two different threshold values, dependent on the batch processor state, are used. When the batch processor fails, the jobs that are currently loaded do not return to the batch processor buffer. If these jobs are not to be considered as part of the throughput, then the state representation in Section 4.8.2 has to be augmented to count the number of jobs inside the batch processor when it fails. If jobs that are inside the batch processor when it fails are considered part of the throughput rate, then the throughput rate can be obtained in the same manner as the throughput rate computation of Model Two. We assume that jobs that are inside the batch processor when the batch processor fails is counted as throughput. The transition probability equations when the batch processor is under no-idling policy is in Table C-5, of Appendix C and the transition probability equations when the batch processor is under full-batch policy is in Table C-6 of Appendix C.

4.9 Model Four Assumptions and Development

Models One to Three assume that there is zero probability of the serial processor being starved of jobs. Consequently, when the batch processor is under full-batch policy, the waiting time incurred by a job when a full batch is unavailable is dependent only on the processing time of the serial processor. In Model Four, we modify the problem to allow for the possibility that the serial processor may get starved of raw material. We do this by augmenting the system used in Models One to Three with a buffer for the serial processor; jobs arrive at this buffer following a stochastic process.

4.9.1 Problem formulation for Model Four

A two-stage manufacturing system has an upstream serial processor being fed by a Buffer A of infinite capacity. Jobs leaving the serial processor go to a second Buffer B with infinite capacity, which feeds a downstream batch processor. The buffer levels of Buffers A and B do not include the jobs that are being processed by the processors. The batch processor can process up to Q jobs at a time, and its processing time is independent of the number of jobs loaded into the batch processor. Downstream of the batch processor is an infinite sink for jobs. Both processors cannot be preempted. Figure 4.14 illustrates the manufacturing system model. We assume that the manufacturing system is operating on a discrete time scale.

There is a processing time window between the batch processor and its feeder processor. If a job has not started processing at the batch processor T time units after it has exited the feeder processor, a penalty C is incurred. For each job that the feeder processor processes (including jobs that end up reaching the processing time window), a reward R is generated. It is assumed that $C > R$, otherwise, it would be always profitable to process a job, regardless of the probability of the job reaching its processing time window.

We determine when it is appropriate to process at the feeder processor, given the buffer levels at A and B, and the status of the batch processor. For this analysis, we pay attention to the

waiting time incurred by a prospective job while waiting for a full batch to be formed. The results obtained can then be combined with Models Two and Three.

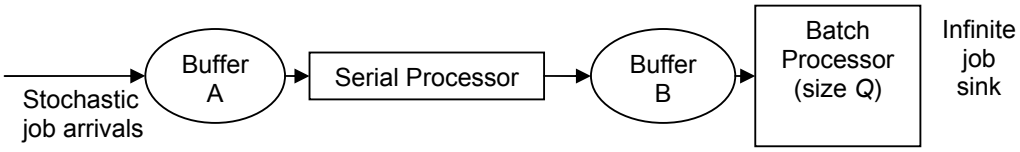


Figure 4.14: Simple 2-stage manufacturing model that allows for the possibility of the serial processor being starved of WIP.

Job arrivals into Buffer A follow a stochastic process. Thus, the serial processor also has to take into account the arrival pattern of jobs into Buffer A when making a decision on whether to process a job or not.

4.9.2 Model Four Assumptions

- Serial processor is perfectly reliable, and the serial processor processing time is geometrically distributed. The serial processor can only process one job at each time instance.
- At each time instance, the probability of a job arriving in front of the serial processor is P_A. Only one job may arrive in front of the serial processor at any time instance. The time between arrivals in front of the serial processor is geometrically distributed.
- A job that arrives in front of the serial processor at time instance x, can only start processing at time instance $x + 1$.
- Batch processor is under full-batch policy. If batch processor is under no-idling policy, the probability of the serial processor starving does not affect the analysis.

Model Four is independent of the reliability and processing rate of the batch processor.

4.9.3 Model Four Development

Let Y_t be the number of jobs inside the serial processor buffer at time instance t. When the number of jobs in front of the serial processor is finite, there exists the possibility that the serial processor will be starved of work. This only affects the analysis for the cases when the

batch processor is under full-batch policy, since the no-idling policy will not force the batch processor to wait for additional jobs to arrive. The results can be incorporated into Models Two and Three by replacing the previous expressions for the probability of incurring a penalty due to excessive waiting for a full-batch to be formed with the new expressions derived in model four. Since the analysis is insensitive to batch processor status, the state at time t, $X_t = (Z_t, F_t, Y_t)$, where Z_t and F_t are defined in the same way as in Section 4.7.2.

A decision can only be made if the serial processor status $F_t = 0$ and buffer A is not empty ($Y_t > 0$); this is an assumption we will make throughout the model development and analysis.

At time t, let $V_t = Z_t - Q\lfloor Z_t Q^{-1} \rfloor$. V_t is the number of jobs waiting for a full batch to be formed. Let B_t be the number of additional jobs the prospective job needs to form a full batch. If $V_t = 0$, then the prospective job will start a new batch and it has to wait for $Q - 1$ additional jobs to arrive into the batch processor buffer. Otherwise, $B_t = V_t - 1$.

The probability of the prospective job incurring a penalty is the probability of either event occurring: (1) time for the serial processor to process an additional B_t arrivals takes more than T time units, and (2) time for the batch processor to process all batches in front of the prospective job – time for the serial processor to finish the prospective job is at least T time units. The second component of the queuing time is independent of what happens at the serial processor, and can be obtained from the previous models.

Y_t is the number of jobs in front of the serial processor at time t, including the prospective job. Thus, $Y_t > 0$. If $Y_t > B_t + 1$, then the waiting time of the prospective job is only dependent on the time the serial processor takes to process B_t jobs. This is because the serial processor is guaranteed to have enough WIP to process enough jobs for the prospective job to form a full batch. Probability of $B_t > 0$ arrivals taking at least T time units, conditioned on $Y_t > B_t + 1$.

$$= 1 - \sum_{i=B_t}^{T-1} {}^{i-1}C_{B_t-1} (1 - P_S)^{i-B_t} P_S^{B_t} \qquad (4.37).$$

This can also be expressed as: $1 - \sum_{i=B_t}^{T-1} G(B_t, i, P_S)$, given $B_t > 0$.

If $Y_t < B_t + 1$, we need to determine the probability that a certain number of jobs arrived in front of the batch processor while the prospective job was being processed. If the serial processor took x time units to process the prospective job, the probability of Buffer A experiencing w arrivals during that interval is: ${}^{x}C_{w}(1-P_A)^{x-w}P_A{}^{w}$ if $w \leq x$, and zero otherwise.

The probability of $w > 0$ arrivals while the serial processor is processing the prospective job is:

$$\sum_{x=w}^{\infty}(1-P_S)^{x-1}P_S{}^{x}C_{w}(1-P_A)^{x-w}P_A{}^{w} = \frac{P_S P_A{}^{w}(1-P_S)^{w-1}}{(1-(1-P_A)(1-P_S))^{w+1}} \qquad (4.38)$$

The probability of $w = 0$ arrivals while the serial processor is processing a job is:

$$\sum_{x=1}^{\infty}(1-P_S)^{x-1}P_S(1-P_A)^{x} = P_S(1-P_A)\Big/1-(1-P_S)(1-P_A) \qquad (4.39)$$

If the prospective job takes x time instances and w arrivals occur in the next x time instances, $Y_{t+x} = Y_t - 1 + w$. The prospective job has to wait for B_t additional job arrivals into the batch processor buffer. This is equivalent to waiting for the serial processor to process an additional B_t jobs. When $w > B_t - Y_t$, there are enough jobs in buffer A at time $t + x$ for supply the serial processor with all the jobs needed by the prospective job to form a full batch, in which case we already know how to compute for the probability the prospective job will expire.

Let $P(i)$ be the probability of i arrivals while the serial processor is processing the prospective job. The probability $P(\circ)$ that $w > B_t - Y_t$ arrivals occur is $P(\circ) = 1 - \sum_{i-0}^{B_t - Y_t} P(i)$ (4.40).

For $w > B_t - Y_t$, the probability of the prospective job incurring a penalty is identical to equation (4.36). Let $P_{(i)}$ be the probability of the prospective job incurring a penalty when i jobs arrive in front of the serial processor while the serial processor is processing the prospective job. The total probability of the prospective job incurring a penalty, when $Y_t < B_t$, is: $= P(\circ)\left[1 - \sum_{i-B_t}^{T-1} G(B_t, i, P_S)\right] + \sum_{i-0}^{B_t - Y_t} P(i)P_{(i)}$ (4.41)

The only unknown left is $P_{(i)}$ for $i = 0$ *to* $B_t - Y_t$. $P_{(i)}$ is the probability of the serial processor processing less than B_t jobs in $T - 1$ time units, when there are initially $B_t - Y_t + 1 + i$ jobs in front of the serial processor buffer. We determine $P_{(i)}$ by modeling the arrival and processing of jobs as a discrete time, discrete state Markov Chain. The system is an *m/m/1* queue, and we are interested in its transient behavior. At a certain time instance t, Let q_t be the number of jobs in front of the serial processor, and p_t be the number of jobs that have been processed in the *m/m/1* queue up to time instance t. Let the state of the *m/m/1* queue be $X_{(t)} = (q_t, p_t)$.

The prospective job incurs a penalty when B_t jobs are not processed by $T - 1$ time units. Thus, we can set any state (q_t, B_t) as a sink. Due to the unlimited buffer size of Buffer A, the number of states can be infinite. Because we are only interested in the possibility of B_t jobs getting processed in the next $T - 1$ time instances, having more than B_t jobs inside Buffer A at any time is irrelevant; we set an artificial buffer limit of B_t at Buffer A.

Without losing generality, let us assume that we start the analysis of the *m/m/1* queue at time instance 0. The initial state (q_0, p_0) can have values q_0 from 0 *to* $Q - 1$, and $p_0 = 0$. To obtain $P_{(i)}$, set initial state to $(B_t - Y_t + 1 + i, 0)$. Since i cannot take more than $B_t + 1$ values (0 to B_t), and there can be $B_t + 1$ values for p_t, the transition probability matrix is square with dimension $(B_t + 1)^2$. The transition equations are shown in Table 4.6.

TABLE 4.6: TRANSITION EQUATIONS FOR MODIFIED M/M/1 QUEUE

Initial state	Final state	Transition probability	State conditions
$(0, p_t)$	$(0, p_t)$	$(1 - P_A)$	$P_t < B_t$
$(0, p_t)$	$(1, p_t)$	P_A	$P_t < B_t$
(q_t, p_t)	$(q_t, p_t + 1)$	$P_A P_S$	$B_t > q_t > 0$, $p_t < B_t$
(q_t, p_t)	$(q_t + 1, p_t)$	$P_A (1 - P_S)$	$B_t > q_t > 0$, $p_t < B_t$
(q_t, p_t)	$(q_t - 1, p_t + 1)$	$(1 - P_A) P_S$	$B_t > q_t > 0$, $p_t < B_t$
(q_t, p_t)	(q_t, p_t)	$(1 - P_A) (1 - P_S)$	$B_t > q_t > 0$, $p_t < B_t$
(B_t, p_t)	$(B_t, p_t + 1)$	P_S	$P_t < B_t$
(B_t, p_t)	(B_t, p_t)	$1 - P_S$	$P_t < B_t$
(q_t, B_t)	(q_t, B_t)	1	

Let the one-step transition probability matrix be M. To obtain $p_{(i)}$, set the initial state $X_{(0)} = (i + Y_t - 1, 0)$, and generate the initial state vector. Each component has value 1 if this component

corresponds to $X_{(0)}$, and 0 otherwise, leaving the vector with the form $< 0,0,0,\ldots,1,\ldots 0, 0,0 >$. This vector is multiplied with $M^{(T-1)}$ to obtain the state probability distribution after $T - 1$ time instances. The resulting vector is the state probability distribution after $T - 1$ time instances when the initial state is $X_{(0)}$. $p_{(i)}$ is the sum of the probability of being in state (q_{T-1}, p_{T-1}) where $p_{T-1} < B_t$ at time instance $T - 1$.

4.9.4 Model Four Discussion

We make the following inferences about the probability of incurring a penalty, with regard to the parameters of the problem:

- The probability of incurring a penalty is non-increasing with increasing processing time windows. Longer processing time windows provide more time for the serial processor to process enough jobs to form a full batch.
- The probability of incurring a penalty is non-decreasing with smaller job arrival probabilities. When the serial processor's current buffer level is insufficient to supply all jobs required to form a full batch at the batch processor, larger job arrival probabilities will reduce the probability that the serial processor will be starved, which would increase the waiting time of the prospective job.
- The probability of incurring a penalty is non-increasing with decreasing number of additional jobs needed by the prospective job to form a full batch. For the same time length, it is more difficult for the serial processor to process more jobs, which causes the probability of incurring a penalty to increase.
- The probability of incurring a penalty is non-increasing with increasing number of jobs queued in front of the serial processor. The more jobs at the serial processor, the smaller the probability of the serial processor getting starved. Once the serial processor buffer has enough jobs to guarantee that the prospective job can form a full batch without waiting for additional jobs to arrive, the probability of incurring a penalty becomes independent of the serial processor queue length.

- The probability of incurring a penalty is non-decreasing with smaller serial processor processing probabilities, IF there are enough jobs in front of the serial processor to guarantee the prospective job can form a full batch without waiting for future job arrivals into the serial processor. When future job arrivals are needed, this may no longer be true. A longer serial processor mean processing time increases the probability of additional jobs arriving at the serial processor buffer while the prospective job is processed.

4.9.4.1 Numerical Experimentation

We perform a set of numerical experiments to empirically estimate computational time required, as well as confirm intuition as to how the optimal policy behaves. The experimental setup is shown on Table 4.7.

TABLE 4.7: MAIN NUMERICAL EXPERIMENTAL SETUP PARAMETERS

	batch processor queue	Serial processor queue	Processing time window length	Serial processor processing probability	Job arrival rate multiplier
Min. value	0	1	6	0.2	0.2
Max. value	5	5	12	0.8	0.8
Increment	1	1	2	0.2	0.2
No. of possible values	6	6	4	4	4

The actual job arrival rate is (Serial processor processing probability) × (job arrival rate multiplier). There are a total of 6 × 5 × 4 × 4 × 4 = 1920 problem instances solved, and a naïve implementation of this algorithm on MATLAB took an mean of 3.87 seconds (averaged over 5 repetitions) to solve these 1920 problems.

The naïve implementation recreated the transition probability matrix for each problem. There are several ways to make this implementation more efficient. Firstly, M^{T-1} can be stored and reused, when the problem parameters $(P_A, P_S, \text{and } T)$ do not change. An improvement over this technique is to store the nonzero elements of M^{T-1} in terms of mathematical expressions; M^{T-1} can then be reused when P_A or P_S changes. When B_t is large, M (a square matrix of dimension $(B_t + 1)^2$) is sparse, since at most only four elements per row will have non-zero entries. Storing non-zero entries of M as a vector allows us to perform matrix multiplication in $O(B_t^4)$ time. It would then take $O((T-2) \times (B_t^4))$ time to obtain M^{T-1}. This can still be improved; for

example, let $T - 1 = 8$, then M^4 can be obtained by squaring M^2, and M^8 by squaring M^4. In this manner, the bound can be improved to approximately $O(\log(T - 1) \times (B_t^4))$.

From the bound obtained, it is expected that large values for the processing time window will adversely affect the computational time required. We take the same experimental setup in Table 4.7, and multiply all the processing time window values by ten. The 1920 problem instances take an average of 32.06 seconds, roughly ten times the computational time needed for the original 1920 problem instances. This supports the hypothesis that there is a linear relationship between T and the computational time.

For the relationship between the probability of incurring a penalty and the serial processor processing probability, when the number of job arrivals needed is low, the probability of incurring a penalty is inversely proportional to the probability of the serial processor finishing a job. This is consistent with the relationship between the two variables when no additional job arrivals are required. When the number of job arrivals required is moderately high, the relationship between the two variables is parabolic in nature: extremely high or extremely low values for the probability of the serial processor finishing a job lead to higher probability of incurring a penalty. This behavior is illustrated in Figure 4.15.

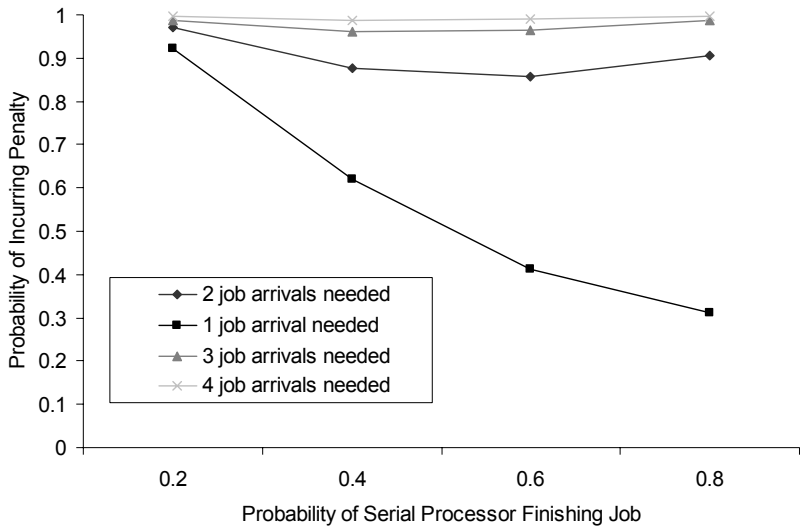


Figure 4.15: Probability of incurring penalty may not be strictly decreasing with increased serial processor processing rate.

Probability of prospective job incurring penalty when the batch processor has size five, the job arrival multiplier is set at 0.6, and the processing time window is set to 12, when the number of jobs in front of the serial processor is insufficient to provide the prospective job with all the jobs needed to form a full batch.

4.10 Characterization of Optimal Feeder Processor Policy when Batch Processor is Reliable and has Geometric Processing Time

To illustrate the effect of ignoring the probability of serial processor starvation, we plot the probability of incurring a penalty when the serial processor buffer experiences geometric arrivals and both processors have geometric processing times. We are combining Models Two and Four. In this experiment, the mean serial processing rate is 0.5 jobs per unit time, the mean batch processing rate is 0.25 jobs per unit time, the batch processor size is four, the mean job arrival rate is 0.7 jobs per unit time and the processing time window is 10 time units. The probability of incurring a penalty according to the number of jobs queued at the serial processor buffer (including the prospective job to be processed) is found in Figure 4.16.

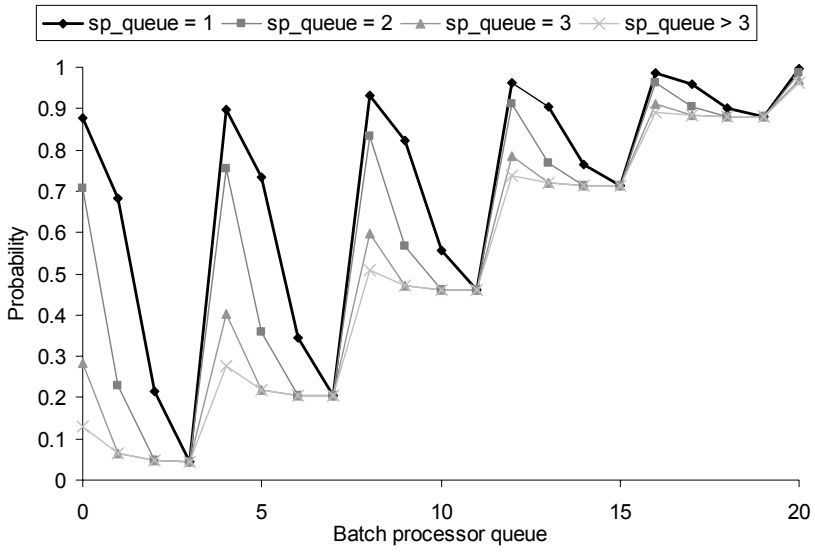


Figure 4.16: The probability of incurring a penalty, according to the queue length of the serial processor buffer, when the batch processor is busy.

The probability of incurring a penalty is non-decreasing with fewer jobs at the serial processor queue. There can be at most Q different probability plots, from sp_queue = 1 to sp_queue = Q.

Firstly, the gross characterization of the probability curves do not differ much from what is obtained when the serial processor can never be starved. Secondly, when the probability of serial processor starvation is ignored, the probability curve we assume is identical to the curve when the serial processor buffer contains at least one full batch. (This is the plot corresponding to the legend "sp_queue > 3".) Unfortunately, there can be large differences between the probabilities of incurring a penalty, according to serial processor buffer quantity, when the prospective job needs a large amount of additional jobs to form a full batch.

The number of distinct probability curves is equal to the batch processor size, since the probability curve is non-existent for serial processor queue size of zero, and since the probability curve is identical for serial processor queue sizes at least as large as the batch processor size. For each of these probability curves, there exists a threshold policy. Thus, the policy we obtain from using our methodology is a threshold policy dependent on the value of the serial processor queue, and the number of different thresholds can be as large as the batch processor size.

4.11 Extension of Results to Different Systems

The assumption that the reward R is earned for all jobs processed by the feeder processor, regardless of whether the job reaches its processing time window or not can be changed to assume that only jobs that do not reach its processing time window will earn a reward. Let R' be the reward earned for jobs that do not reach its processing time window, and let $P(X_t)$ be the probability of the prospective job incurring a penalty C, given state X_t. Then the news-vendor model suggests that we only process a job if the expected reward ($R' \times (1 - P(X_t))$ is greater than the expected penalty ($C \times P(X_t)$), where R' need not be smaller than C. The condition for processing can also be expressed as: $R'/C > P(X_t)/(1 - P(X_t))$. One desirable property is that, for any two time instances t and r, if $P(X_t) > P(X_r)$, then $P(X_t)/(1 - P(X_t)) > P(X_r)/(1 - P(X_r))$. Thus, if the optimal policy for the original problem (where a reward R is generated for each job processed by the feeder processor) is a threshold policy, then the optimal policy for the

new problem (where a reward *R'* is generated for each job that does not reach its processing time window) will also be a threshold policy.

In all four models, we assume that the system is composed of a serial processor feeding a batch processor. This manufacturing configuration assumption is not as restrictive as it appears; when the feeder processor is also a batch processor, we can first evaluate the effect of loading a full batch, by assuming that all but the last job inside the batch are already queued in front of the batch processor. If the analysis outputs that processing a full batch does not have a positive expected reward, we repeat the process by deducting one from the batch size. Furthermore, when the size of the batch processor is one, the two-stage system becomes a tandem line with two serial processors. An example of a case where there is a processing time window between two serial processors can be found inside cluster tools for wafer fabs ([Yoon and Lee 2005]), where the addition of small amounts of buffer modules can improve performance of the cluster tool ([Rostami and Hamidzadeh 2004]).

When the batch processor control policy is a generic minimum batch size policy, the equations used to determine the optimal policy when the batch processor is under no-idling policy can also be modified to suit the case where the batch processor is operated using a minimum batch size policy < *Q*.

When the jobs have different values for *R*, *C* and *T*, the first-come-first-served policy may no longer be appropriate, and job overtaking in the batch processor queue may occur. This may result in the following situation: a job *X* was processed by the feeder processor because the queue length in front of the job was initially *z*. Assume z is the largest queue length for which it was profitable to process job *X*. Subsequently, *y* additional jobs with higher priority were processed by the feeder processor, which increased the number of jobs in front of job *X* from *z* to $z + y$. If it was known that the queue length in front of job *X* will become $z + y$, instead of z, it would have been profitable to withhold processing job *X* until the queue length is reduced. This suggests that being able to predict future job arrivals into the feeder processor queue may be beneficial when jobs entering the batch processor have different *R*, *C* and *T* values.

For Models Two and Three, it is easy to modify the equations for the case where the serial processor processing time distribution is not geometric, as long as the processing time distribution is discrete (or can otherwise be adequately approximated by a discrete time distribution) and independent of the processing time distribution of the batch processor. This is because all probabilities of incurring a penalty are derived by initially conditioning on the serial processor finishing the job after x time units. Adopting Model Four to when the serial processor has a different processing time distribution is also possible, although significant state representation augmentation may be required, which could cause the matrix M to expand and increase the execution time of the algorithm.

Most wafer fabs would be processing more than one job family at a time, and jobs belonging to different job families cannot be batched together at the batch processor. In this case, the optimal control policy at the feeder processor would be highly dependent on the control policy at the batch processor. One natural heuristic, whose optimality cannot be guaranteed without knowing the control policy at the batch processor, is to limit the number of total batches in front of the batch processor, regardless of job family. Another variation is to measure the amount of work in front of a prospective job in terms of number of processing hours, rather than number of batches. This might be an improvement over the performance of the initially suggested heuristic when the job families have significantly different processing times at the batch processor.

4.12 Chapter Summary

In wafer fabs, it is common to find that jobs have to be loaded into the batch processor within a certain amount of time after it exits the processor immediately upstream of the batch processor. If a job does not get loaded into the batch processor within its processing time window, it cannot be processed without additional test or rework. Based on this problem, we develop four models, with increasing degrees of complexity, and characterize the optimal

feeder processor control policy for each of these models. The objective function is to maximize the expected reward generated by a job at each instance the feeder processor is idle.

In Model One, the upstream processor has deterministic processing time of one time unit, and the only source of variability was the batch processor processing time. In this model, we determine that the optimal policy has the following desirable characteristics:

- The optimal policy is a threshold
- The optimal threshold is an integer multiple of the batch processor size.
- Given an additional assumption for the full-batch policy, the optimal policy at the upstream processor is insensitive to whether the batch processor is under full-batch or no-idling policy.

Unfortunately, the three desirable characteristics are no longer true when the serial processor processing time assumes a geometric distribution. (Model Two). Under Model Two, the optimal policy is still a threshold policy when the batch processor is under no-idling policy. However, the optimal threshold may no longer be an integer multiple of the batch processor, due to the uncertainty in the number of batches the batch processor has to process before the prospective job is loaded into the batch processor.

When the batch processor is under full-batch policy, the probability of incurring a penalty is no longer strictly decreasing with respect to the number of jobs in queue. However, by replacing the original probability plot with an approximate probability plot, the optimal threshold policy still has an optimal threshold that is an integer multiple of the batch processor size.

In Model Three, the batch processor is assumed to suffer from operation-dependent failures. We generate expressions to determine the probability of incurring a penalty, and evaluate a hypothesis that if, at any time unit, the probability of not repairing a failed processor is less than the probability of the processor not failing and not finishing the batch, then the probability of incurring a penalty is higher when the batch processor is initially busy,

compared to when the batch processor is initially down. Experimental results support this hypothesis, regardless of batch processor policy.

In the previous models, the upstream processor is assumed to have zero probability of WIP starvation; Model Four considers the possibility of WIP starvation at the upstream processor. Model Four is relevant only when the batch processor policy allows the batch processor to wait for additional job arrivals, and can be combined with Models Two and Three, when the batch processor is under full-batch policy.

The results presented in this chapter can also be used as guides in determining the size of the batch processor buffers. In the next chapters, we discuss how the selection of job family can be done at the upstream processor to reduce the time spent by jobs in the system.

Chapter 5 Control of Upstream Serial Processor with a Downstream Batch Processor

5.1 Introduction

In Chapter 3, we have seen how varying the job arrival pattern observed by the batch processor buffers can result in jobs spending less time at the batch processor buffers. In Chapter 4, we introduce the need to control the sizes for batch processor buffers, given the presence of processing time windows. In this chapter, we explicitly model the upstream processor and its control policies.

Majority of the current literature on the control of batch processors assume that the batch processor is in isolation ([Mathirajan and Sivakumar 2006]). A reasonable conjecture is that controlling the processors upstream of the batch processor can improve the manufacturing system performance. This conjecture is supported by observations in Section 3.4.2.2.2, where an increase in positive correlation between the job families of consecutive job arrivals can cause the mean cycle time of jobs passing through the batch processor to decrease, regardless of the batch processor policy.

To determine the feasibility of the conjecture, we look at the performance of a truncated manufacturing system, comprised of a serial processor feeding the batch processor. The system produces jobs that belong to two different job families, and each job can only be batched together with jobs from the same family. The concept of dictating the production of the serial processor as a function of the anticipated immediate needs of the batch processor is evaluated.

5.2 Problem Statement for Two-stage Model

A manufacturing system is comprised of two processors and processes jobs that belong to one of m different job families. All jobs visit the processors in the same order. All jobs from the same family are identical. The first (otherwise known as upstream) Processor M_1 is a serial processor, and the second (downstream) Processor M_2 is a batch processor. The Batch Processor M_2 can process up to Q jobs, all belonging to the same family, at an instance, and processing time is independent of the number of jobs loaded into the batch processor. Both processors are assumed to be perfectly reliable, and job preemption is not allowed.

In front of each Processor M_i are m different Buffers B_{ij} of finite size C_{ij}. Each Buffer B_{ij} contains jobs belonging to Family j waiting to be processed by Processor M_i. When Buffer B_{ij} has zero jobs ($b_{ij} = 0$), Processor M_i is starved of Job Family j, and M_i cannot process a batch from Job Family j; When $b_{ij} = C_{ij}$, incoming arrivals into Buffer B_{ij} are blocked, and these arrivals are lost.

It is assumed that any control policy at M_1 ensures that Job J can be accommodated by B_{2j} before processing J. This ensures that the rate jobs enter the system is equal to the rate jobs exit the system. After being processed by M_1, Job J enters B_{2j} and waits for M_2 to process it. Job J exits the manufacturing system after being processed by M_2. Figure 5.1 illustrates the manufacturing system when there are $n = 2$ job families being processed.

Both processors are assumed to be perfectly flexible (no set-up time required when we switch processing from one job family to the next). When Buffer B_{1j} is not full, the arrival process for Job Family j into the system is a stochastic process with a time-invariant distribution.

We wish to determine the long-run mean performance of manufacturing systems of this nature under several simple two-stage control policies. Performance of the system is measured using two metrics: the total throughput rate of the system, and the over-all mean cycle time of jobs passing through the system.

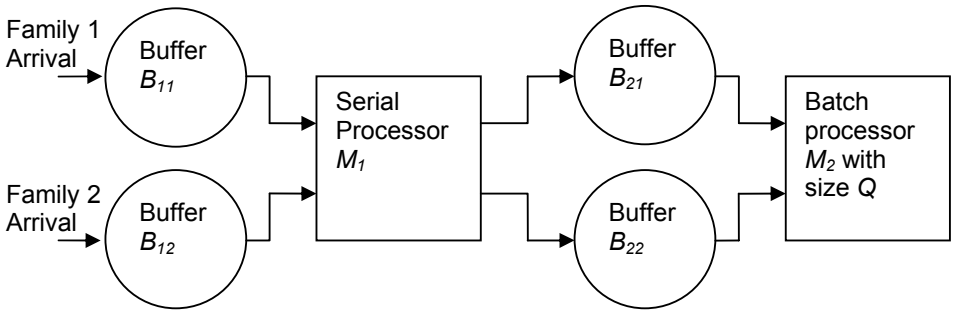


Figure 5.1: A two-stage manufacturing system with a serial-batch configuration. The system is processing two job families, and has two buffers, one for each job family, in front of every processor.

5.3 Model Development

There are several ways to model the manufacturing system described. We opt to model the manufacturing system as a Markov chain, primarily because we are interested in steady-state behavior of the manufacturing system. Modeling the manufacturing system as a Markov chain provides information on the exact proportion the manufacturing system spends on each state.

Unfortunately, the computational and storage effort required increases rapidly as the manufacturing system become more complex, making it practically infeasible to model very complex manufacturing systems as Markov Chains. The Markovian assumption of the independence of future states being independent of past states, conditioned on the current state contributes to this problem, as it sometimes becomes necessary to increase the state space size to take the Markovian assumption into account.

5.3.1 Development of Two-Stage Markovian Model

Two different control policies for the batch processor are evaluated. The full-batch policy will not load any partial batches into the batch processor; the batch processor will wait for additional arrivals. The second control policy (no-idling policy) will load batches into the batch processor as long as WIP is present in the Buffers B_{21} and B_{22}. These two control policies can be considered the extremes of every control policy at the batch processor; one policy never waits for additional arrivals, while another policy always waits until the batch processor is full. We assume that the batch processor always selects to process the job family with more jobs queueing at the batch processor.

We test several serial processor control policies, as combined with the two batch processor policies, to determine relative performance of the manufacturing system under each serial-batch control policy combination.

The two-stage manufacturing system in Section 5.3 is modeled as a continuous time- discrete state Markov chain, with stationary transition probabilities. Let $X(t)$ be the state at current Time Instance t, then, the Markovian assumption states that:

$Prob.(X(\tau) = x(\tau) \mid X(s) = x(s), for\ s < t) = Prob.(X(\tau) = x(\tau) \mid X(t) = x(t))$ for $\tau > t$.

As a consequence of the Markovian assumption, the current state $X(t)$ must carry all the information needed to determine the next state $X(t')$, assuming the next transition occurs at t'. Given m job families and k stages, there will be km buffers in the system. Let $k = 2$ and b_{ij} be the quantity of WIP remaining in buffer B_{ij}, then the status of the buffer levels at any Time Instance t is represented by the vector $v(t) = (b_{11}, b_{12},...,b_{1n}, b_{21}, b_{22},...,b_{2n})$. If there are c possible values for each b_{ij}, the number of distinct vectors $v(t)$ would be m^{2c}. This rapid growth in the state space makes it generally impractical to model manufacturing systems that process a large number of job families, and we limit our analysis to two job families.

5.3.1.1 Model Assumptions

We make the following assumptions for the manufacturing system in Section 5.3:

- The manufacturing system processes two job families.

- The time between external arrivals belonging to Job Family i is exponentially distributed, with arrival rate λ, regardless of job family. If Buffer B_{1i} is full, any Job Family i arrivals are lost.
- The time required to process a job belonging to Job Family i at the serial processor is exponentially distributed, with service rate μ_s, regardless of job family. The serial processor will not process a job belonging to Job Family i if buffer B_{2i} is full.
- The time required to process a batch at the batch processor is exponentially distributed with service rate μ_B, regardless of the job family currently being processed.

5.3.1.2 State Representation

Let the variables be defined as:

$b_{1i}(t)$ – the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the serial processor at time instance t, *including* the job currently being processed.

$b_{2i}(t)$ - the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the batch processor at time instance t, *excluding* the job/s currently being processed.

$S_s(t)$ – The status of the serial processor at Time Instance t. If $S_s(t) = 0$, the serial processor is idle. If $S_s(t) = i > 1$, the serial processor is processing a job belonging to Job Family i.

$S_b(t)$ - The status of the batch processor at Time Instance t. If $S_b(t) = 0$, the batch processor is idle. If $S_b(t) = 1$, the batch processor is processing a batch.

Then, at any Time Instance t, the state $X(t)$ is described by the vector $X(t) = (b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t), S_s(t), S_b(t))$. We frequently omit explicitly referring to t.

The primary motivation behind the state representation selected is the reduction of the state space. For example, by assuming that all batches have identical mean processing times at the batch processor, it becomes unnecessary to keep track of the job family currently being processed by the batch processor.

Assuming the Markov chain is ergodic, we obtain the steady state distribution of the model and use the steady state probabilities to obtain the performance measures (throughput rate and the mean cycle time) of the manufacturing system subjected to a specific control policy.

The amount of WIP L inside the manufacturing system, as defined by the state representation, does not include the jobs being processed by the batch processor. This is one of the consequences of not keeping track of the quantity of jobs currently loaded into the batch processor. Since the mean processing time at the batch processor is identical for both job families, we can add the mean batch processor processing time to the obtained mean cycle time to get the mean cycle time, if we wish to obtain the mean time jobs spend from entry at the serial processor buffer till exit at the batch processor.

5.3.1.3 Serial Processor Control Policies

We develop three simple serial processor control policies that we combine with batch processor control policies to form two - stage control policies. A two - stage control policy describes the control policy for both processors. We assume two common properties among all serial processor control policies considered:

- Before processing a particular job, the processor will first confirm whether there is available space in the downstream buffer for the job family to be processed. If no space is available, the processor does not process a job from that particular job family.
- At any instance the serial processor is available, if the current buffer levels allow for only one job family to be processed, then the serial processor will process that particular job family, regardless of serial processor control policy.

5.3.1.3.1 Myopic control policy

Let the serial processor be M_i. If M_i has a choice on which job family to choose, M_i processes Job Family One if $b_{i1} \geq b_{i2}$, and processes Job Family Two if $b_{i2} > b_{i1}$. This policy has been shown to stochastically minimize the amount of jobs lost, if both job families have the same processing time and the same buffer capacities ([Winston 1977]). For this reason, we expect that using myopic control policy at the serial processor will result in higher throughput rates. The two-stage control policy with the myopic control policy at the serial processor and the full-batch policy at the batch processor is the myopic-full-batch policy, while the two stage policy with myopic control policy at the serial processor and a no-idling policy at the batch

processor is called the myopic – no-idling policy. This nomenclature is used for naming all two-stage control policies.

5.3.1.3.2 Greedy-look-ahead policy

Let the serial processor be M_i and the batch processor be M_k. If M_i has a choice on which job family to choose, M_i processes Job Family One if $b_{k1} \geq b_{k2}$, and processes Job Family Two if $b_{k2} > b_{k1}$. This is based on the idea of constraining the serial processor to process what the batch processor will process, given that the batch processor will process the job family with the most jobs in front of the batch processor. If we are using the greedy-look-ahead-full-batch policy, we expect the greedy look-ahead policy to reduce the time jobs spend waiting in front of the batch processor for a full-batch to be formed. Similarly, if the greedy-look-ahead-no-idling policy is applied, we expect the average sizes of batches to increase.

5.3.1.3.3 Balanced-look-ahead policy

One possible problem with the greedy-look-ahead policy is its behavior when the number of jobs belonging to a particular Job Family i exceeds the batch processor size Q. The greedy-look-ahead policy will cause the serial processor to continue processing Job Family i, even though a full-batch is already available. This could cause the number of jobs belonging to a different Job Family j, $i \neq j$, to accumulate in front of the serial processor. The balanced-look-ahead policy seeks to correct this.

Let the batch processor be M_k. If the serial processor has a choice on which job family to choose, the processor processes Job Family One if $b_{k1} \leq b_{k2}$, and processes Job Family Two if $b_{k2} < b_{k1}$. By aiming for an equal number of jobs between two job families in front of the batch processor, we guard against prematurely blocking a buffer.

5.3.1.4 Batch Processor Control Policies

Assuming that the batch processor is M_k, the no-idling policy uses the following logic to determine the appropriate action of the batch processor at each instance it is available:

- If $b_{k1} = 0$, and if $b_{k2} = 0$, wait for a job to arrive.

- If $b_{k1} + b_{k2} > 0$, load *min (Q, b_{21})* jobs from Job Family One if $b_{k1} \geq b_{k2}$; otherwise, load *min (Q, b_{k2})* jobs from Job Family Two into M_k.

The full-batch policy is characterized by the following logic:

- If $b_{k1} < Q$ and if $b_{k2} < Q$, wait for more arrivals.
- If $b_{k1} \geq Q$ and if $b_{k2} \leq b_{k1}$, load Q jobs from Job Family One into M_k.
- If $b_{k2} \geq Q$ and if $b_{k1} < b_{k2}$, load Q jobs from Job Family Two into M_k.

For both batch processor control policies, the batch processor processes the job family with the longest queue in front of the batch processor. When the buffer sizes are identical for both job families and the batch processor processing time distribution is identical for both job families, this policy stochastically minimizes the loss of jobs, and consequently maximizes the throughput for a particular arrival rate for the batch processor in isolation ([Xia *et al.* 2002]).

All processor policies instruct the serial or batch processor to process jobs from Job Family One if the selection criterion is indifferent towards any job family. This bias towards processing jobs from Job Family One is expected to cause the mean throughput rate of Job Family One will be slightly higher than that of Job Family Two. The mean cycle time of jobs from Job Family One is also expected to be slightly lower than that of Job Family Two.

The manner in which the probability rate transition equations are obtained under either batch processor policy can be found in Appendix D.1 . The transition probability equations are obtained by generating the state space and using the flowchart in Appendix D.1 to determine which states a particular state can transition to , with a particular transition rate.

If the number of states is n, the a_{ij}^{th} entry of the n by n transition rate matrix A is obtained by: letting a_{ij} be the rate of moving from state i to state j in one transition, if $i \neq j$. If $i = j$, then a_{ij} is equal to $-\sum_{j \neq i} a_{ij}$ if $i = j$. There are several ways to obtain the steady state distribution of the system given the transition rate matrix. We opt to find the δ-skeleton of the continuous time discete state Markov Chain, by assuming a small time step-size δ. Given a transition rate matrix A, we build a discretized transition probability matrix B, with entries b_{ij} obtained through the following formula: If $i \neq j$, $b_{ij} = a_{ij} \times \delta$, and if $i = j$, $b_{ij} = 1 + a_{ij} \times \delta$. One way to

interpret the matrix B is as the discrete-time discrete-state Markov Chain of the same system, with discrete time increments of size δ, and with the limitation that, at most one transition can occur at each time increment.

Having converted the transition rate matrix A to the transition probability matrix B, we can solve for the 1 by n steady-state probability vector P (with entries p_i) that will satisfy the two equations: $P=BP$ and $\sum_{i=1}^{n} p_i = 1$.

5.3.1.5 State Space Discussion under Differing Batch Processor Policies

The size of the state space will differ according to the batch processor control policy. We determine the size of the state space by examining the possible buffer levels for each (S_s, S_b) combination, when the batch processor is under no-idling policy.

- If $(S_s = 0 \cap S_b = 0)$, $b_{21} = 0$ and $b_{22} = 0$ and $b_{11} = 0$ and $b_{12} = 0$, resulting in one state.
- If $(S_s = 1 \cap S_b = 0)$, $b_{21} = 0$ and $b_{22} = 0$ and $b_{11} \neq 0$, resulting in $C_{11} \times (C_{12} + 1)$ states.
- If $(S_s = 2 \cap S_b = 0)$, $b_{21} = 0$ & $b_{22} = 0$ & $b_{12} \neq 0$, resulting in $C_{12} \times (C_{11} + 1)$ states.
- If $(S_s = 0 \cap S_b = 1)$,$(b_{11} \neq 0$ or $b_{21} \neq C_{21})$ and $(b_{12} \neq 0$ or $b_{22} \neq C_{22})$, resulting in $C_{21} \times (C_{22} + 1) + C_{11} \times (C_{22} + 1) + C_{12} \times (C_{21} + 1) + C_{22} \times (C_{11} + 1) + 1$ states.
- If $(S_s = 1 \cap S_b = 1)$,$b_{21} \neq C_{21}$and $b_{11} \neq 0$, resulting in $C_{11} \times (C_{12} + 1) \times C_{21} \times (C_{22} + 1)$ states.
- If $(S_s = 2 \cap S_b = 1)$,$b_{22} \neq C_{22}$ and $b_{12} \neq 0$, resulting in $(C_{11} + 1) \times C_{12} \times (C_{21} + 1) \times C_{22}$ states.

The state space for a manufacturing system operating under no-idling control policy at the batch processor is the sum of all possible combinations for each combination of S_s and S_b.

We also compute for the state space when the manufacturing system is operating under full-batch control policy.

- If $(S_s = 0 \cap S_b = 0)$, $b_{21} < Q$ and $b_{22} < Q$ and $b_{11} = 0$ and $b_{12} = 0$, resulting in Q^2 states.
- If $(S_s = 1 \cap S_b = 0)$, $b_{21} < Q$ and $b_{22} < Q$ and $b_{11} \neq 0$, resulting in $C_{11} \times (C_{12} + 1) \times Q^2$ states.

- If ($S_s = 2 \cap S_b = 0$),$b_{21} < Q$ and $b_{22} < Q$ and $b_{12} \neq 0$, resulting in $C_{12} \times (C_{11} + 1) \times Q^2$ states.
- If ($S_s = 0 \cap S_b = 1$),$(b_{11} \neq 0$ or $b_{21} \neq C_{21})$ and $(b_{12} \neq 0$ or $b_{22} \neq C_{22})$, resulting in $C_{21} \times (C_{22} + 1) + C_{11} \times (C_{22} + 1) + C_{12} \times (C_{21} + 1) + C_{22} \times (C_{11} + 1) + 1$ states.
- If ($S_s = 1 \cap S_b = 1$), $b_{21} \neq C_{21}$and $b_{11} \neq 0$, resulting in $C_{11} \times (C_{12} + 1) \times C_{21} \times (C_{22} + 1)$ states.
- If ($S_s = 2 \cap S_b = 1$), $b_{22} \neq C_{22}$ and $b_{12} \neq 0$, resulting in $(C_{11} + 1) \times C_{12} \times (C_{21} + 1) \times C_{22}$ states.

The state space for a manufacturing system operating under a full-batch control policy is the sum of all possible combinations for each combination of S_s and S_b. Since $Q > 1$, the state space of a manufacturing system with batch processor under full-batch policy will be strictly larger than that of a manufacturing system with batch processor under no-idling policy.

5.4 Development of Three-Stage Markovian Model

To determine whether the inferences from a two-stage model is also true for larger manufacturing systems, a three-stage model involving two upstream serial processors with a downstream batch processor is also generated.

5.4.1 Problem formulation for Three-stage Model

A manufacturing system is comprised of three processors and processes jobs that belong to one of m different job families. All jobs visit the processors in the same order. All jobs from the same family are identical. The first two Processors M_0 and M_1 are serial processors, and the third Processor M_2 is a batch processor. The Batch Processor M_2 can process up to Q jobs, all belonging to the same family, at an instance, and processing time is independent of the number of jobs loaded into the batch processor. All processors are assumed to be perfectly reliable, and job preemption is not allowed.

In front of each Processor M_i are m different Buffers B_{ij} of finite size C_{ij}. Figure 5.2 illustrates the manufacturing system when there are $m = 2$ job families being processed.

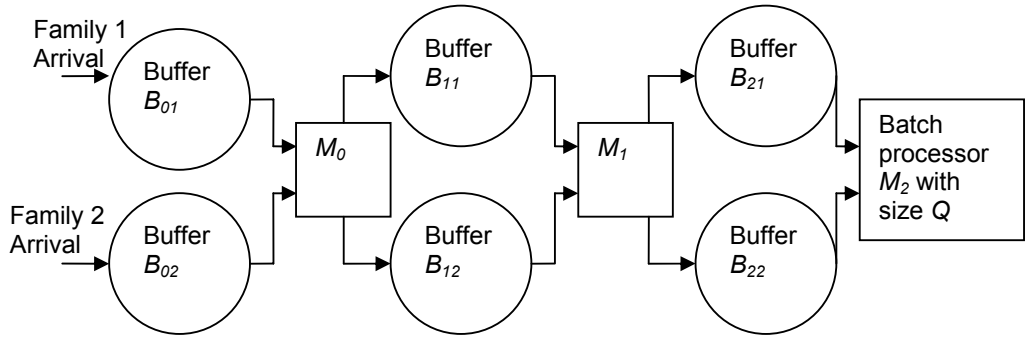


Figure 5.2: A three-stage manufacturing system with a serial-serial-batch configuration.

Processors M_0 and M_1 are serial processors. In this example, the system is processing two job families, and would have two buffers, one for each job family, in front of every processor.

All three processors are assumed to be perfectly flexible When Buffer B_{0j} is not full, the arrival process for Job Family j into the system is a stochastic process with a time-invariant distribution. We determine the long-run mean performance of manufacturing systems of this nature under simple three-stage control policies.

5.4.2 Model Assumptions

We make the following assumptions for the manufacturing system in Section 5.4.1:

- The manufacturing system processes two job families.
- The time between external arrivals belonging to Job Family i is exponentially distributed, with arrival rate λ, regardless of job family. If the Buffer B_{0i} is full, any job arrivals belonging to Job Family i are lost.
- The time required to process a job belonging to Job Family i at the serial processor M_k ($k = 0, 1$) is exponentially distributed, with service rate μ_{sk}, regardless of job family. M_k will not process a job belonging to Job Family i if buffer B_{ki} is full.
- The time required to process a batch at the batch processor is exponentially distributed with service rate μ_B, regardless of the job family currently being processed.

5.4.3 Processor Control Policies

The control policies used for the two serial processors are identical to what is described in Section 5.3.1.3. The control policies used for the batch processor are identical to what is described in Section 5.3.1.4.

5.4.4 State Representation

Let the following variables be defined as:

$b_{ki}(t)$ – the amount of jobs belonging to Job Family i (i = 1, 2) in front of the serial Processor M_k (k=0,1) at Time Instance t, *including* the jobs currently being processed.

$b_{2i}(t)$ - the amount of jobs belonging to Job Family i (i = 1, 2) stored in front of the Batch Processor at Time Instance t, *excluding* the job/s currently being processed.

$S_{sk}(t)$ – The status of the serial Processor M_k (k=0,1) at Time Instance t. If $S_{sk}(t) = 0$, M_k is idle. If $S_{sk}(t) = i>0$, M_k is processing a job belonging to job family i.

$S_b(t)$ - The status of the batch processor at Time Instance t. If $S_b(t) = 0$, the batch processor is idle. If $S_b(t) = 1$, the batch processor is processing a batch.

Then, at any instance t, the state $X(t)$ is described by the vector $X(t) = (b_{01}(t), b_{02}(t) ,b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t), S_{s0}(t), S_{s1}(t), S_b(t))$. The manner in which the transition probabilities are obtained is illustrated in Appendix D.

If m is the maximum of all buffer capacities C_{ij}, there are, at most, $m + 1$ possible values for each buffer level b_{ij}. Since there are 18 possible combinations for S_{s0}, S_{s1} and S_b, the number of states is bounded from above by $18(m + 1)^6$. We omit showing the computation for the exact state space size. Let C_{ij} be the maximum buffer size for Buffer B_{ij}, then the following rules govern the state space of the system:

- $S_{s0}(t) \neq 0$ if $(b_{01}(t) > 0$ *AND* $b_{11}(t) < C_{11})$ *OR* $(b_{02}(t) > 0$ *OR* $b_{12}(t) < C_{12})$
- $S_{s0}(t) \neq 1$ if $(b_{01}(t) = 0$ *OR* $b_{11}(t) = C_{11})$
- $S_{s0}(t) \neq 2$ if $(b_{02}(t) = 0$ *OR* $b_{12}(t) = C_{12})$
- $S_{s1}(t) \neq 0$ if $(b_{11}(t) > 0$ *AND* $b_{21}(t) < C_{21})$ *OR* $(b_{12}(t) > 0$ *OR* $b_{22}(t) < C_{22})$

- $S_{s1}(t) \neq 1$ if $(b_{11}(t) = 0$ *OR* $b_{21}(t) = C_{21})$
- $S_{s1}(t) \neq 2$ if $(b_{12}(t) = 0$ *OR* $b_{22}(t) = C_{22})$
- $S_b(t) = 0$ if and only if $(b_{21}(t) = 0 \cap b_{22}(t) = 0 \cap$ *policy is no-idling) OR* $(b_{21}(t) < Q \cap b_{22}(t) < Q \cap$ *policy is full-batch)*

The state space is larger when the batch processor is under full-batch policy.

5.5 Verification of Markovian Models-Comparison with Simulation

The Markovian models created are verified in two ways. For each two-stage control policy, the buffer size is set to three, and all possible transitions for each state are verified. This is a time-consuming method, and for this reason, a complete evaluation of all state transitions is not performed on the three-stage models.

A second method to verify the results of the Markovian model is through comparison with results from an identical simulation model. If the results from the Markov chain models are similar to the results from the simulation models, we gain confidence on the correctness of the various Markov model implementations built. Firstly, simulation models of two- or three-stage systems are relatively easy to build, debug and implement. Secondly, even if the simulation model has a logical error, it is unlikely that the flawed simulation model and the Markov model would have convergent results. We discuss the comparison in this section.

5.5.1 Comparison Methodology

The simulation models are created using the commercial software Simul8. We assume that the system is balanced (all processors have identical maximum processing rates, measured in jobs per unit time), and that the buffer sizes are identical. **Traffic intensity** is a dimensionless metric that describes the relative frequency of job arrivals compared to the processor's processing rate. The traffic intensity *TI* experienced by a processor is the mean job arrival rate

(in jobs per unit time) divided by the product of the processor's mean processing rate (in service completions per unit time) and the processor's maximum number of jobs per service completion. If λ is the mean job arrival rate, μ is the mean processing rate and c is the maximum size of the processor, then $TI = \lambda / (c \times \mu)$. When the processor is a serial processor, then the definition of TI is identical to that of the processor's utilization ρ.

Table 5.1 lists the various two-stage simulation models built. Table 5.2 lists the three-stage simulation models built. For the two-stage models, the batch processor size is held constant at two jobs. For the three-stage models, both the buffer sizes and the batch processor size are varied. The serial processor mean processing time is fixed at one time unit. Each problem instance was simulated for 11000 time units; the first 1000 time units was used as the warm-up period, and the remaining 10000 time units was used to collect data on the models. Because the simulation models are very simple at this stage, both the warm-up and results collection period are given ample allowances.

TABLE 5.1: TWO-STAGE SIMULATION MODELS COMPARED WITH MARKOV MODELS

Two-stage control policy	traffic intensity	Buffer size	Number of problem instances
Myopic-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10
Myopic-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10
Greedy look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10
Greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10
Balanced-look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10
Balanced-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2, 7	10

For each problem instance, 100 runs are performed. The throughput rate is measured directly, while mean cycle time was obtained through Little's Law by tracking the number of jobs in the system. This is the same manner in which cycle time is obtained under the Markov models. We compare the mean throughput and cycle time obtained from these 100 runs against the values obtained through the Markov model. For each set of parameters, the percentage difference between the simulation model throughput, τ_S, and the Markov model throughput, τ_M, is: $200 * |\tau_S - \tau_M| / (\tau_S + \tau_M)$. The percentage difference between the simulation model cycle time and Markov model cycle time is obtained in a similar manner.

TABLE 5.2: THREE-STAGE SIMULATION MODELS COMPARED WITH MARKOV MODELS

Three-stage control policy	traffic intensity	Batch processor size	Buffer size	Number of problem instances
Myopic-myopic-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Myopic-myopic-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Myopic-greedy look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Myopic-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Greedy-look-ahead-greedy-look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Greedy-look-ahead-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Balanced-look-ahead-greedy-look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Balanced-look-ahead-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	2	2, 3, 4	15
Myopic-myopic-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Myopic-myopic-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Myopic-greedy look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Myopic-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Greedy-look-ahead-greedy-look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Greedy-look-ahead-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Balanced-look-ahead-greedy-look-ahead-no-idling policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10
Balanced-look-ahead-greedy-look-ahead-full-batch policy	0.2, 0.4, 0.6, 0.8, 1.0	3	3, 4	10

5.5.2 Comparison Results

Table 5.3 contains the mean and maximum percentage differences in throughput and cycle time for comparisons performed for the two-stage model, while Table 5.4 contains the mean and maximum differences in throughput and cycle time for comparisons for the three-stage manufacturing system. In both tables, the discrepancies in performance estimates between the two models are low, although the larger models tend to have larger discrepancies.

The magnitude of discrepancies between the cycle time values obtained from the Markov and simulation models are generally larger than the magnitude in discrepancies between the throughput values of the Markov and simulation models. This observation may be attributed to

the fact that the cycle time is a derived value, as compared to throughput, which is directly measured.

TABLE 5.3: COMPARISON BETWEEN TWO-STAGE MARKOV MODEL AND SIMULATION MODELS

Two-stage control policy	Mean throughput difference (%)	Maximum throughput difference (%)	Mean cycle time difference (%)	Maximum cycle time difference (%)
Myopic-no-idling policy	0.129	0.206	0.353	0.816
Myopic-full-batch policy	0.126	0.237	0.196	0.396
Greedy look-ahead-no-idling policy	0.121	0.233	0.347	0.840
Greedy-look-ahead-full-batch policy	0.155	0.262	0.189	0.359
Balanced-look-ahead-no-idling policy	0.172	0.275	0.283	0.793
Balanced-look-ahead-full-batch policy	0.146	0.200	0.174	0.391

Figure 5.3 illustrates the relationship between the results obtained from the Markov model and the results obtained from the simulation model, for the three-stage model. The results of the two models agree with each other; at low traffic intensities, the results from the simulation and Markov models are virtually indistinguishable from each other.

TABLE 5.4: COMPARISON BETWEEN THREE-STAGE MARKOV MODEL AND SIMULATION MODELS

Three-stage control policy	Mean throughput difference (%)	Maximum throughput difference (%)	Mean cycle time difference (%)	Maximum cycle time difference (%)
Myopic-myopic-no-idling policy	0.118	0.255	0.448	1.212
Myopic-myopic-full-batch policy	0.128	0.272	0.216	0.431
Myopic-greedy look-ahead-no-idling policy	0.146	0.405	0.450	1.065
Myopic-greedy-look-ahead-full-batch policy	0.129	0.245	0.186	0.429
Greedy-look-ahead-greedy-look-ahead-no-idling policy	0.150	0.437	0.459	1.257
Greedy-look-ahead-greedy-look-ahead-full-batch policy	0.128	0.224	0.196	0.407
Balanced-look-ahead-greedy-look-ahead-no-idling policy	0.152	0.475	0.430	1.087
Balanced-look-ahead-greedy-look-ahead-full-batch policy	0.132	0.228	0.193	0.409

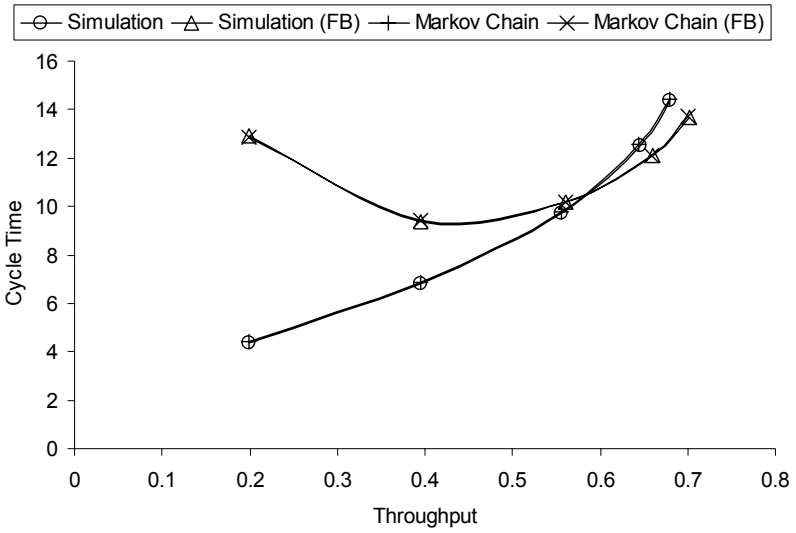


Figure 5.3: Comparing the cycle time and throughput from Markov and simulation models, when the policy is myopic-myopic-no-idling or myopic-myopic-full-batch.

The pair of convex figures corresponds to the batch processor using full-batch policy. The system parameters are buffer sizes of three and batch processor size of three. There is slight separation between the two model families, and this is more readily seen when the batch processor is under no-idling policy.

5.6 Behavior of Two-Stage System

In this section we discuss the experimental setup used, and analyze the behavior of the two-stage manufacturing system under different conditions. We initially characterize the performance of the model under various control policies. This is to confirm intuition with respect to the gross relationship between the system performance and the problem parameters. We then compare the performance of the model under specific policies against each other. This allows us to determine under which conditions a particular control policy will work better than others.

5.6.1 Experimental Design

A total of six two-stage control policies are evaluated: three control policies for the serial processor (myopic, greedy look-ahead and balanced look-ahead) and two control policies for the batch processor. We have two broad experimental set ups, summarized in Table 5.5. In the

first setup, the buffer size is kept constant, and the batch processor size and mean processing rate is varied. For each two-stage control policy, batch size and buffer capacity setting, 25 data points are collected. Each data point corresponds to a particular combination of traffic intensities on the two processors. (The traffic intensity for each processor is set to five values, giving a total of 25 possible combinations.) The second setup has fixed batch processor size and the buffer capacity is incremented.

The values selected for buffer sizes and batch processor sizes are necessarily low, due to the fast rate of growth in the state space as the buffer sizes are increased. If m is the maximum buffer size, the state space size s grows with rate $O(m^4)$ and the transition probability rate matrix has dimensions s by s. However, this matrix is sparse, since at most only five entries per row can have nonzero values. These five entries correspond to job arrivals from either family, either processor finishing a job, and the rate in which the state leaves its own state. It is then possible to store the non-zero elements of the matrix instead to save on space.

A common method of obtaining the steady-state probabilities requires inversion of the transition probability matrix. If the transition probability matrix has dimension s, matrix inversion takes $O(s^3)$ time. Thus, the number of operations required to obtain the steady state distribution is $O(m^{12})$. A more efficient way of obtaining the steady state distributions is detailed in [Gershwin 2005]. By taking advantage of the sparsity of the transition matrix of Markov chains, the steady state distribution can be obtained in only $O(m^8)$ operations. Furthermore, the storage requirements of this method is also significantly lower, as only the non-zero values of the transition probability rate matrix are stored. The number of elements of the matrix grows with rate $O(m^8)$, while the number of non-zero elements grows with rate $O(m^4)$. This method of obtaining the steady-state distributions is also used in Chapter 6.

TABLE 5.5: EXPERIMENTAL SETUP SUMMARY FOR TWO-STAGE SYSTEM

Purpose of experiment	No. of control policies	Buffer Sizes	Batch Processor Sizes	Traffic intensities for serial processor	Traffic intensities for batch processor
Evaluate effect of changing batch size	6	6	2, 3, 4, 5	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0
Evaluate effect of changing buffer size	6	2, 3, 4, 5, 6, 7	2	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0

For each experiment, the mean processing rate of the serial processor is held constant at one unit per unit time, and the nominal traffic intensities are used to determine the mean arrival rate of both job families, as well as the mean processing time at the batch processor. For all experiments, both job families have equal arrival rates. Let λ be the mean arrival rate for each Job Family and μ_B be the mean processing rate (in terms of batches per unit time) in the batch processor. Then,

λ = traffic intensity *TI* of serial processor / 2

μ_B = *TI* of serial processor / (*TI* of batch processor × batch processor size)

By fixing the mean serial processor processing rate to one job per time unit, the actual utilization of the serial processor is also the throughput rate of the system. Furthermore, the stated traffic intensity of the serial processor becomes the theoretical maximum throughput rate of the system, if no job arrivals are ever rejected entry to the system. For fixed serial processor traffic intensity and fixed batch processor size, traffic intensity at the batch processor is proportional to the mean time to process a batch at the batch processor.

The reference serial processor policy is the myopic policy; when the serial processor is in isolation and the buffer capacities and processing rates are equal for all job families, then the myopic policy maximizes the throughput rate. Since the method in which the batch processor selects the job family to process also locally maximizes the throughput rate of the batch processor when in isolation, the system throughput with the myopic policy at the serial processor is the system throughput if locally optimal policies are used at each stage.

Performance of the system is evaluated using mean throughput rate of the system and cycle time of jobs passing through the system. The performance characterization is not segregated according to job family, as it bears a very strong resemblance to the characterization of the over-all system performance. As expected through the introduction of bias towards processing jobs from Job Family One, the throughput rate for Job Family One is slightly higher than that of Job Family Two, and the cycle time of jobs belonging to Job Family One is slightly lower than that of Job Family Two.

5.6.2 Sensitivity of Model to Increasing Buffer Size

If the status of processors can be perfectly predicted, there is no need for buffers in between processors ([Gershwin 2002]). Buffers are present to partially decouple the performance of one processor from another. When the buffer sizes are infinite, any incoming jobs will be accepted by the system, and the job arrival rate will equal the throughput rate, assuming that the job arrival rate is less than the processing rate of the slowest processing rate of the system and perfect yield. Aside from the computational concerns of modeling a system with infinite buffer sizes, jobs queuing in front of the batch processor can have queuing time limits, as seen in Chapter 4. Thus, there exist reasons for limiting the buffer sizes to a finite value.

For multi-stage systems with only serial processors, increasing the buffer sizes increases both throughput and cycle time. Larger buffer sizes allow more jobs to enter the system, and consequently reduce the amount of time processors are idle. Allowing more jobs into the system also increases the probability of an incoming job seeing a long line of jobs in front of a particular processor, leading to increased cycle time. This intuition is also expected to be true for the two-stage system with a downstream batch processor.

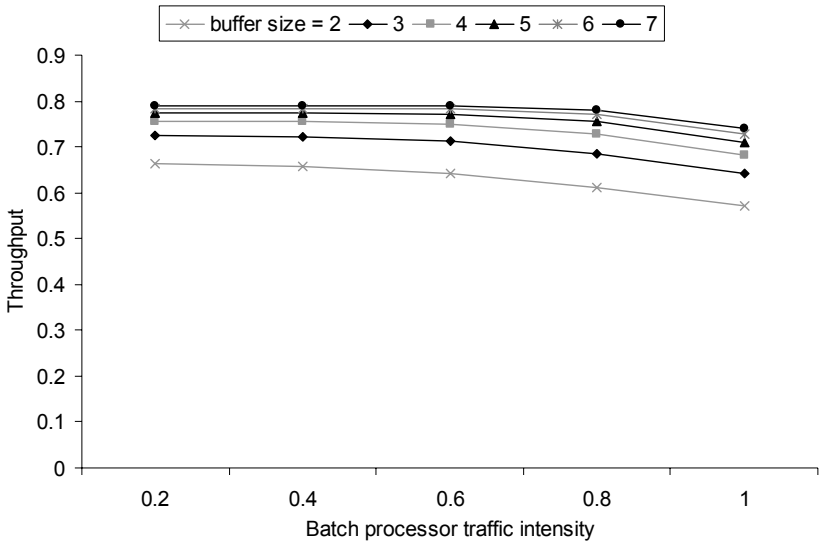


Figure 5.4: Throughput as a function of the batch processor traffic intensity.

The system parameters are: 0.8 serial processor traffic intensity, batch processor size of two and the myopic-full-batch policy. As the average processing time of the batch processor increases, the throughput of the system goes down. Increasing the size of the buffers can help mitigate this effect.

Figure 5.4 illustrates the evolution of the system's throughput as a function of the batch processor traffic intensity for several buffer sizes, when the traffic intensity at the serial processor is 0.8. By increasing the batch processor traffic intensity while keeping the serial processor traffic intensity constant, the mean processing time of the batch processor lengthens. Figure 5.4 shows the throughput of the system decreases as the batch processor traffic intensity increases. As the time required to process a batch increases, the probability of WIP accumulating in front of the batch processor increases, and this increases the probability of either batch processor buffer being full. When a batch processor buffer is full, the serial processor is blocked from processing jobs destined for the full batch processor buffer; this, in turn, leads to an increased probability of the serial processor buffer becoming full. The larger the buffer sizes, the less likely this chain of events will occur.

When the batch processor traffic intensity is low, the serial processor is the system constraint (processor with the lowest throughput rate within the system), and majority of the system WIP is in front of the serial processor. As the batch processor traffic intensity increases, the distribution of WIP shifts from being in front of the serial processor to being in front of the batch processor, which increases the probability of the batch processor buffer becoming full.

Figure 5.5 illustrates the throughput of the system as a function of the buffer size. As the buffer sizes C_{ij} for all buffers B_{ij} increases, the throughput increases. From the positive slope and the concavity of the curves, we infer that the throughput increase associated with increased buffer sizes decreases as the buffer sizes increase. This observation is consistent with what is expected for systems involving only serial processors. The effect of buffer size on throughput is also more pronounced when the batch processor takes a longer time to process a batch. When the batch processor's processing rate is high, jobs in front of the batch processor rarely have to wait for long before being processed, and this diminishes the importance of having large buffers to obtain throughput.

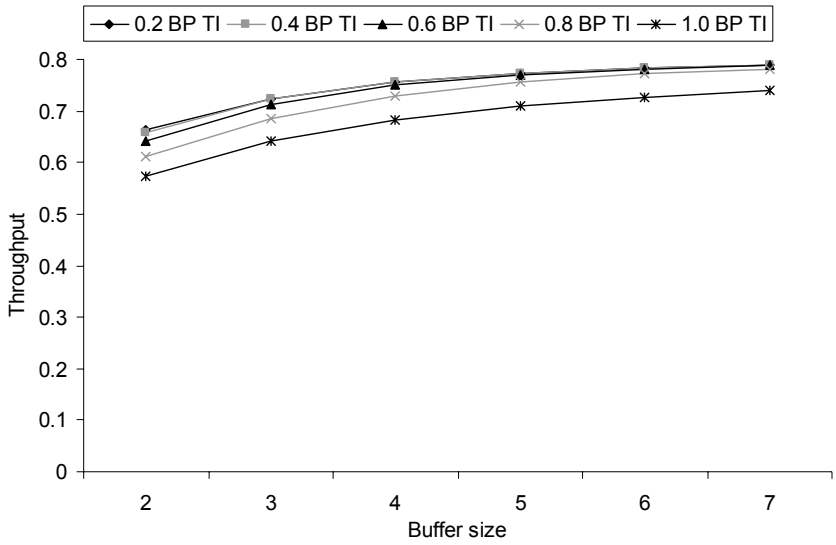


Figure 5.5: Throughput as a function of the buffer size.

The system parameters are: 0.8 serial processor traffic intensity, batch processor size of two and the myopic-full-batch policy. The incremental improvement gained from increasing the buffer size by one decreases as buffer sizes increase.

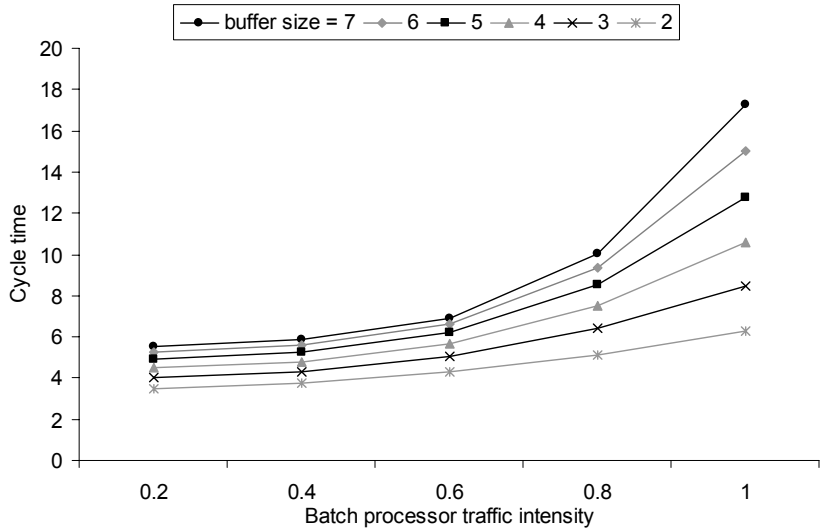


Figure 5.6: Cycle time as a function of the batch processor traffic intensity.

The system parameters are: 0.8 serial processor traffic intensity, batch processor size of two, and the system is under myopic-full-batch policy. As expected, the cycle time increases as the processing time of the batch processor increases. Larger buffer sizes allow more jobs to enter the system, which causes the cycle time to increase.

Figure 5.6 illustrates the mean cycle time of products of the system as a function of the batch processor traffic intensity while Figure 5.7 shows the mean cycle time of products inside the system as a function of the buffer size.

From Figure 5.6, the presence of larger buffer sizes allows for more dramatic increases in the mean cycle time as the batch processor traffic intensity increases. This can be explained by referring to the differences between an *m/m/1* queue and an *m/m/1/k* queue. (see, for example: [Hopp and Spearman 2000]) For the *m/m/1* queue, the number of jobs queuing in front of the processor increases rapidly as the arrival rate of the system approaches the processing rate. For the *m/m/1/k* queue, the number of jobs inside the system has a finite limit, which also limits the mean cycle time is also limited. The smaller the buffer sizes, the easier it is to find the buffers full, even at low traffic intensities. Thus, when the processing time of the batch processor is increased until traffic intensity equals 1.0, the mean cycle time does not increase at an accelerating rate when the buffer sizes are small. As the buffer sizes approach infinity, the behavior of the mean cycle time approaches that observed for an *m/m/1* queue, and the mean cycle time rapidly increases as the batch processor traffic intensity is increased.

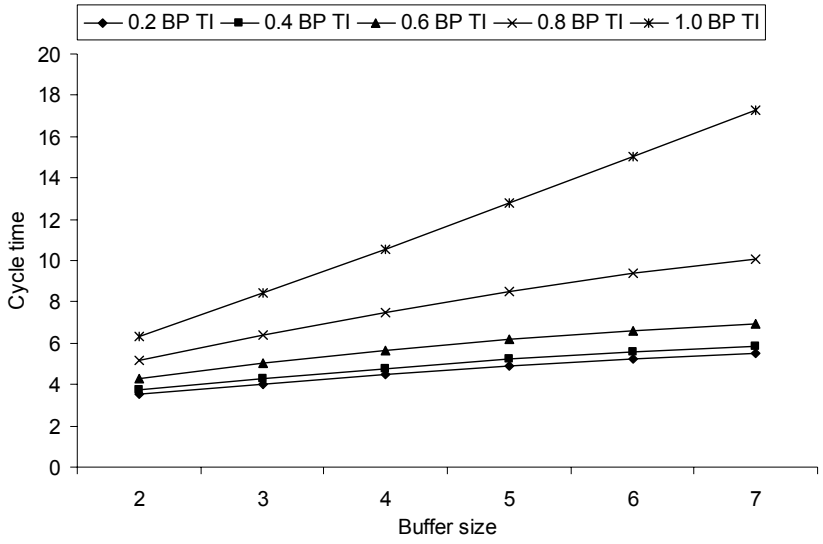


Figure 5.7: Throughput as a function of the buffer size.

The system parameters are: 0.8 serial processor traffic intensity, batch processor size of two, and the system is under myopic-full-batch policy. The incremental increase in cycle time increases as the batch processor processing rate becomes slower, relative to the serial processor processing rate.

Figure 5.7 further illustrates the relationship between buffer sizes, mean cycle time, and location of the constraint. When the batch processor traffic intensity is lower than the serial processor traffic intensity (between the values 0.2 and 0.6), the incremental increase in cycle time is low. This is because only the buffers in front of the serial processor are significantly affected by the buffer size increase. When both processors have identical maximum processing rates, the incremental increase in cycle time for each additional buffer space is noticeably higher. Finally, for a system with the downstream processor as a strong constraint, the incremental increase in cycle time is even higher, as the mean buffer levels in front of the batch processor is high, which leads to the serial processor getting blocked and causing WIP levels in front of it to also increase.

5.6.3 Sensitivity of Model to Increasing Batch Processor Size

When the size of the batch processor increases and the traffic intensities in both processors are kept constant, the processing time at the batch processor is lengthened, such that the ratio between the batch processor size and its processing time remains constant. Suppose we can choose between a serial processor and a batch processor, with both processors having the same maximum throughput rate. Replacing a serial processor with a batch processor with the same maximum throughput rate should induce a reduction in throughput rate and an increase in the cycle time. This is because we force jobs to either wait for additional job arrivals, or we process a partial batch and under-utilize the batch processor size. This intuition supports the simulation results of [Noben *et al.* 2001], which show that the mean cycle time is lower when processors with smaller sizes and faster processing times are used instead of processors with larger sizes and slower processing times.

Figure 5.8 supports the intuition that the throughput is higher when batch processor sizes are smaller but faster. When the batch processor uses no-idling policy, higher batch processor sizes make it less likely that each batch that gets processed is a full batch. This means that the

batch processor needs to process more batches than expected, which increases the probability that jobs spending significant time waiting in front of the batch processor.

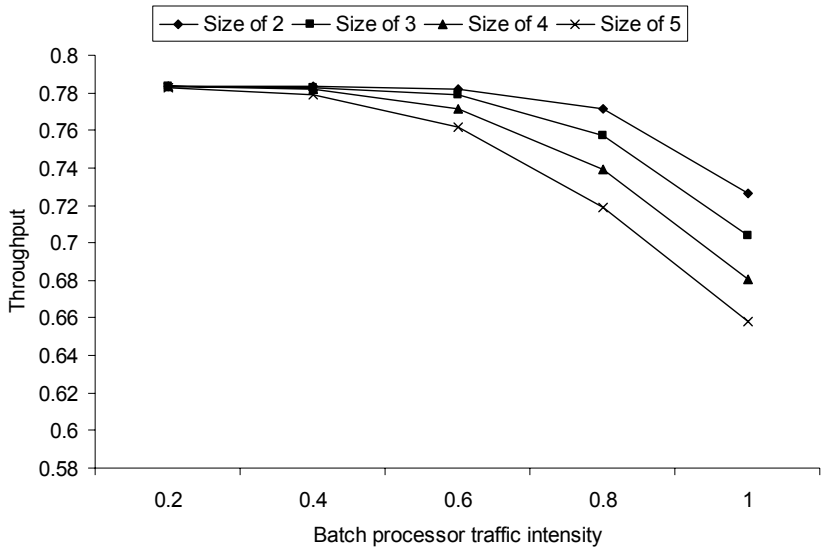


Figure 5.8: Throughput as a function of the batch processor traffic intensity.

System parameters are: buffer sizes are six, the serial processor traffic intensity is 0.8, and the policy in place is myopic-full-batch policy. Increasing the batch processor size (while maintaining constant batch processor traffic intensity) reduces the throughput.

When the batch processor uses full-batch policy, the probability of finding full batches of jobs ready whenever the batch processor becomes available is low, and jobs that arrive will have to wait longer periods of time for a full batch to be formed. The presence of jobs waiting in front of the batch processor increases the probability of the serial processor getting blocked, which in turn reduces throughput.

This reduction in throughput when batch processors become larger but slower can also be explained by the buffer sizes relative to the batch processor sizes. Assume that the buffer size is six. When the batch processor size is two, we can buffer the batch processor against starvation a maximum of three full batches. On the other hand, when the batch processor size is five, the buffering capability is limited to slightly more than one full batch. Since the buffer size has greater effect on throughput when processor traffic intensity is high, the performance deviation is larger between systems with different sizes with high batch processor traffic intensity.

As the batch processor processing time increases, the number of jobs that arrive in front of the batch processor to find the batch processor busy also increases. Jobs will also have to wait for a longer period of time before the batch processor becomes available. This causes cycle time to increase. Furthermore, it takes a longer time to form full batches (if we use full-batch policy). Figure 5.9 illustrates the evolution of the mean cycle time of jobs in the system as the batch processor traffic intensity is varied.

Figure 5.10 shows the evolution of the mean cycle time as a function of the mean arrival rate when the full-batch policy is used while Figure 5.11 shows the evolution of the mean cycle time as the mean arrival rate is increased when no-idling policy is used. From Figure 5.10, as the batch processor size increases, it becomes increasingly desirable to have higher arrival rates, in terms of reduced cycle time. When the batch processor has a size of five, for example, the cycle time when the traffic intensity is 0.2 is higher than when the traffic intensity is 1.0. The larger the batch size, the more likely it is that an incoming job needs to wait for additional jobs to form a full batch. When the mean arrival rate is low, jobs are forced to wait for long periods, and this causes cycle time to increase.

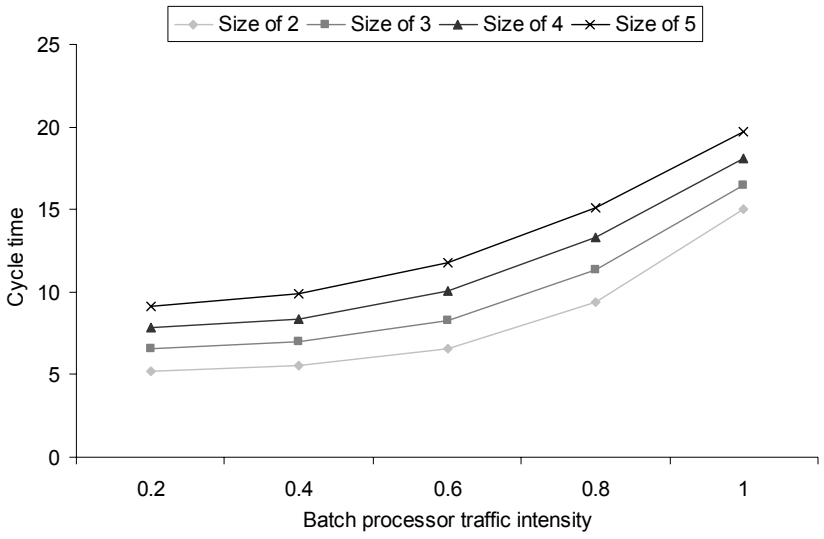


Figure 5.9: Mean cycle time of the system as a function of the batch processor traffic intensity.

The system parameters are identical to that of Figure 5.8. As the batch processor size increases, cycle time increases. Furthermore, the incremental increase in cycle time is non-decreasing with respect to the traffic intensity.

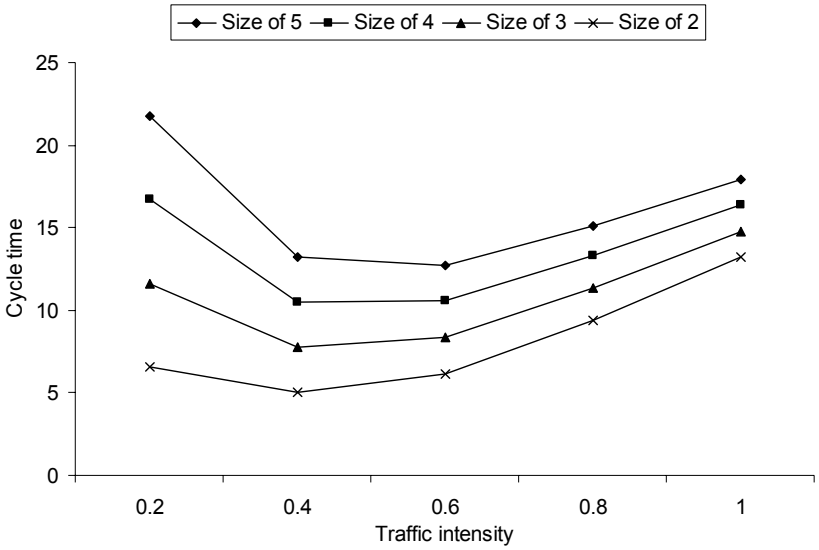


Figure 5.10: Mean cycle time of the system as a function of the traffic intensity for a balanced system under myopic-full-batch policy.

The buffer sizes are kept at six. As the batch processor size increases, the cycle time increases at a larger rate for lower traffic intensities than for higher traffic intensities.

For systems that use no-idling policy (Figure 5.11), the mean cycle time monotonically increases as the batch processor size increases. This is consistent with what is observed with systems involving only serial processors.

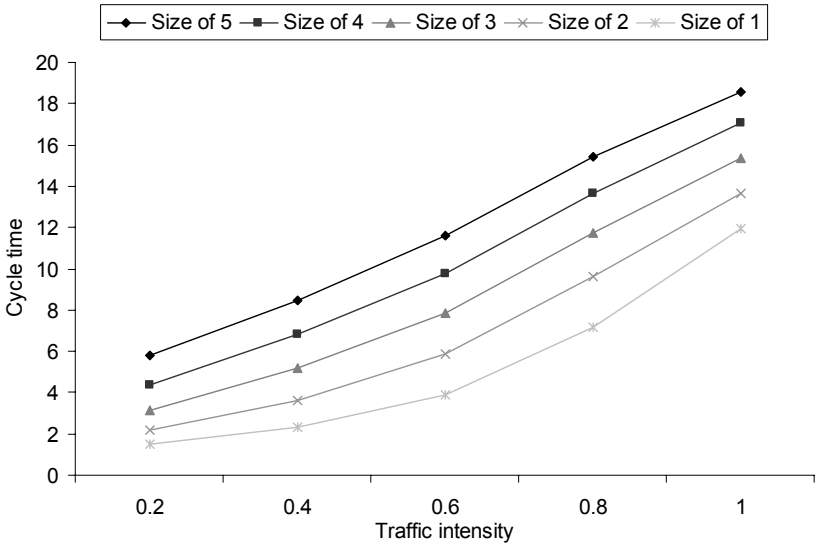


Figure 5.11: Mean cycle time of the system as a function of the traffic intensity for a balanced system under myopic-no-idling policy.

The cycle time for the case where the size is one is obtained via simulation, for comparison purposes. The buffer sizes are six. Mean cycle time is strictly increasing with respect to traffic intensity, similar to what we would expect if all processors in the system are serial processors.

We included the plot of cycle time versus traffic intensity for a system with two serial processors (obtained via simulation) in Figure 5.11 for comparison. We do not see the rapid rise of cycle time expected in systems with only serial processors. We believe the reason for this is that the larger sizes of the batch processor allow the batch processor to compensate for an increase in the job arrival rate by processing larger batches. When the traffic intensity is low, the batch processor will generally process only a few jobs at a time, but will still have high utilization. As the traffic intensity increases, the utilization of the batch processor can only slightly increase, as it is already quite high. The batch processor compensates by increasing the mean number of jobs processed per batch. Thus, the higher the batch processor size, the less dramatic the change in cycle time will be with respect to changes in the arrival rate.

5.6.4 Sensitivity of Model to Different Batch Processor Policies

Figure 5.12 illustrates the behavior of two different systems when different batch processor policies are used. In all systems, the maximum throughput rates are the same for both processors. Thus, an increase in traffic intensity when the processing rates are constant means that the arrival rates increase. As the arrival rate is increased, the throughput of the system increases, regardless of batch processor policy. The system under full-batch policy has greater throughput when the arrival rates are high. When the batch processor is under no-idling policy, the amount of time the batch processor is busy is high, and consequently, jobs arriving in front of the batch processor will tend to form a queue. This queue can propagate and block the production of the serial processor, which leads to incoming job arrivals getting rejected. This scenario is less likely to happen when the batch processor is under full-batch policy. When the traffic intensity is very low, the reverse is true: the system under no-idling policy can have greater throughput. This is because the full-batch policy encourages the build-up of queues in front of the batch processor. This reduces the effective buffering capability of the system, and an increase in the number of job arrivals rejected by the system is expected.

When the batch processor is under no-idling policy, the mean cycle time is strictly increasing with respect to traffic intensity (and throughput rate). Furthermore, the rate in which cycle time increases is accelerated when the traffic intensities are high. When the batch processor is under full-batch policy and the traffic intensity is low, jobs spend a large amount of time waiting for a full batch to be formed, causing jobs to incur a large amount of cycle time. As the traffic intensity is increased, the queues in front of the serial processor increase, which causes increased waiting time for the jobs. This increased waiting time is offset by the reduction in the mean time it takes to form a full batch. As the traffic intensity is further increased, the reduction in the time spent forming a full batch is no longer sufficient to compensate for the increase in waiting time due to the queues at the serial processor, and the mean cycle time starts increasing with respect to the traffic intensity (and throughput rate). Thus the cycle time of the system under full-batch policy is convex, and this behavior is more noticeable when the batch processor size is large.

The intuition of using no-idling policy when traffic intensity is low and using full-batch policy when traffic intensity is high is also supported by Figure 5.12, since the performance plot for the no-idling policy is below that of the full-batch policy when the traffic intensity is low, while the reverse is true when the traffic intensity is high. These observations are magnified when the batch processor size is increased.

When the maximum throughput rate of the serial processor is much lower than that of the batch processor (serial processor is strong system constraint), the throughput of the system is slightly higher and the mean cycle time slightly lower when the batch processor is under no-idling policy. Under the full-batch policy, jobs will be forced to wait in front of the batch processor. Since the serial processor is the constraint of the system, the arrival rate of jobs into the batch processor is low, and causes the mean cycle time of jobs to increase. The presence of jobs queuing in front of the batch processor also increases the probability of the serial processor getting blocked, which causes a small throughput loss. As the maximum throughput rate of the batch processor approaches that of the serial processor, the full-batch policy generates higher throughput and lower cycle time than the no-idling policy. When the batch

processor is under no-idling policy and the batch processor is the constraint of the system, the arrival rate experienced by the batch processor is high compared to its processing rate. Because of the tendency of the batch processor to process partial batches, jobs arriving in front of the batch processor frequently see the batch processor is busy, and are forced to wait for the processing of at least one batch. This causes the cycle time to increase, and also increases the queue lengths seen in front of the batch processor, which increases the probability of the serial processor getting blocked, and, ultimately, the job arrivals getting rejected by the system.

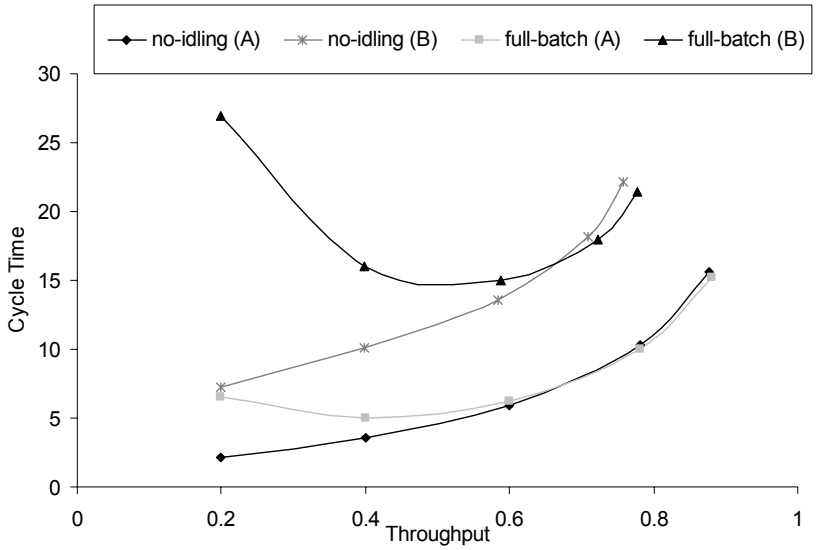


Figure 5.12: Throughput versus cycle time for various two-stage systems with the serial processor under myopic policy.

In both systems, the maximum throughput rate is the same for both processors. The legend indicates the batch processor policy and the parameter set for the system. System A has buffer capacity of seven and batch processor size of two. System B has buffer capacity of seven and batch processor size of six. When the system is under no-idling policy, the cycle time increases as the traffic intensity (and hence the throughput) increases. When the system is under full-batch policy, the cycle time starts at a high value when the traffic intensity is low, as jobs spend a significant amount of time waiting for additional jobs to arrive, to form a full batch. As traffic intensity is increased, the cycle time is initially reduced before it starts increasing.

5.6.5 Sensitivity of Model to Different Serial Processor Policies

We wish to determine the performance of the system when the serial processor policies are varied. Figure 5.13 illustrates the performance of three serial processor control policies when the batch processor is under no-idling policy, while Figure 5.14 illustrates the performance of three serial processor control policies when the batch processor is under full-batch policy. We

assume that the two processors have identical maximum throughput rates, and keep the buffer capacities fixed at seven. In each figure, the lower set of graphs refer to the performance of the system when the batch processor size is two, while the upper set of graphs refer to the system performance when the batch processor size is six.

When the traffic intensities are low, there is little differentiation between the performances of the different control policies. When job arrivals are rare, the system does not have any choice as to which job family to process. The larger the batch processor size, the lower the traffic intensity has to be before there is significant differentiation in performance between the different policies.

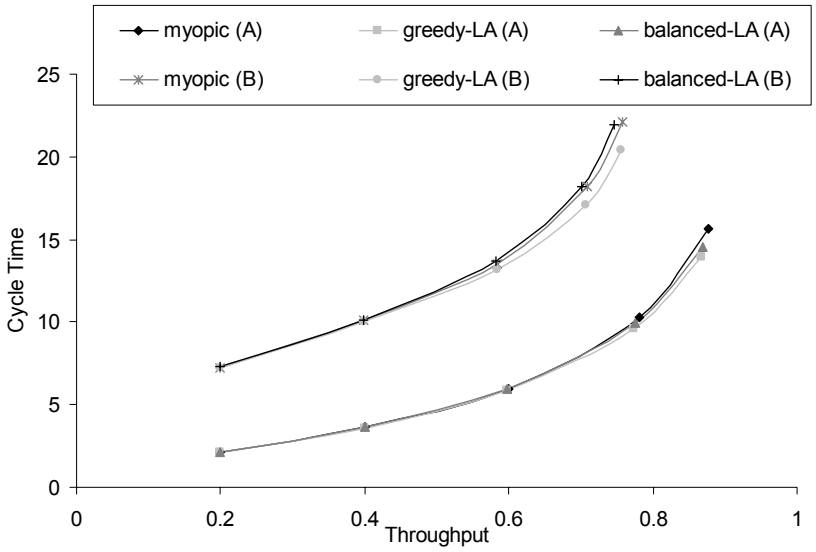


Figure 5.13: Throughput versus cycle time performance of systems with different serial processor policies when the batch processor is under no-idling.

The buffer capacities are held at seven, and the maximum throughput rates of both processors are equal. System A has batch processor size of two, and System B has batch processor size of six. In both systems, the greedy-look-ahead policy results in lower mean cycle time than the myopic policy, at the expense of slightly lower throughput. The combined-look-ahead policy exhibits similar behavior.

Although the balanced-look-ahead policy generates higher throughput rate than the greedy-look-ahead policy when the batch processor has small size, this is no longer true when the batch processor has larger size. Furthermore, the balanced-look-ahead policy has consistently higher cycle times than the greedy-look-ahead policy. This is especially evident when the

batch processor sizes are large compared to the buffer capacities. This suggests that the performance of the balanced look-ahead policy degrades as the batch processor size increases. The balanced-look-ahead policy seeks to keep an equal number of jobs in front of each buffer of the batch processor. This is beneficial when the queue sizes in front of the batch processor are large compared to the batch processor size (Case One), as the blockage of the serial processor gets delayed, which reduces the amount of job arrivals that get rejected. When the queue sizes are low compared to the batch processor size (Case Two), the balanced-look-ahead policy will either cause batches to have few jobs (if batch processor is under no-idling policy), or it can increase the probability of job arrivals getting rejected, due to the significant amount of jobs waiting for a full batch in front of the batch processor (if batch processor is under full-batch policy). When the batch processor size is low, the system spends a significant proportion of time under Case One, and spends a significant amount of time under Case Two when the batch processor size is high.

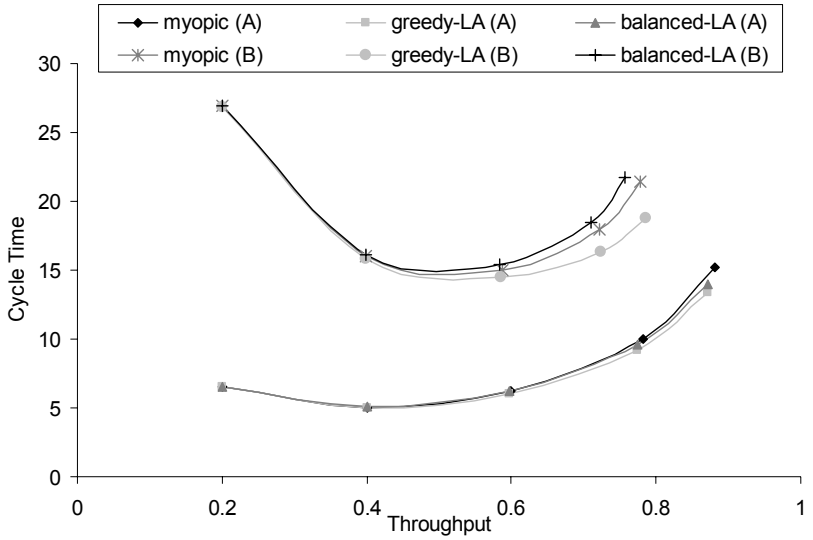


Figure 5.14: Throughput versus cycle time performance of systems with different serial processor policies when the batch processor is under full-batch.

The buffer capacities are held at seven, and the maximum throughput rates of both processors are equal. System A has batch processor size of two, and System B has batch processor size of six. In both systems, the greedy-look-ahead policy results in lower mean cycle time than the myopic policy, at the expense of slightly lower throughput. The cycle time reduction is slightly greater than that observed when the batch processor is under no-idling policy, and the throughput reduction is slightly smaller.

In both Figure 5.13 and Figure 5.14, the plot denoting the system performance for the myopic control policy is above the graph for the greedy-look-ahead policy. This suggests that for a specific throughput rate, the mean cycle time for the myopic control policy is greater than the mean cycle time for the greedy-look-ahead policy. For the same traffic intensity, the two policies will not have the same throughput rate; since the myopic control policy reduces the maximum length of the queue in front of the serial processor, the probability of an incoming job being rejected by the system is generally smaller under the myopic control policy. This results in the myopic control policy typically having larger throughput rate than the greedy-look-ahead policy for a specific arrival rate. From Table 5.7, when the traffic intensity is 1.0 for both processors, using the greedy-look-ahead policy results in 11.86% cycle time reduction, at the expense of 1.04% reduction in the throughput rate when the batch processor is under full-batch policy, and results in 10.8% cycle time reduction, at the expense of 1.1% throughput rate reduction if the batch processor is under no-idling policy.

The buffer sizes in front of the serial processor includes the job being processed by the serial processor, while the buffer sizes in front of the batch processor do not include the jobs being processed by the batch processor. This implies that the maximum number of jobs waiting to be processed by the batch processor can be larger than the maximum number of jobs waiting to be processed by the serial processor. To obtain support that the performance difference between the two policies is not due to this slight buffer size difference, we simulate a two-stage system, System A, with maximum buffer sizes of seven and batch processor size of two. The difference between this two-stage system and the two-stage system in Section 5.2 is that the queue sizes in front of the serial processor of System A do not include the job currently being processed. When System A has traffic intensity of 1.0, the greedy-look-ahead policy has 1.12% less throughput rate and 12.47% less cycle time when the batch processor is under no-idling policy, and has 1.10% less throughput rate and 13.26% less cycle time when the batch processor is under full-batch policy. These results suggest that the observed performance difference cannot be attributed to the slight difference in buffer sizes between the stages.

We also discount the possibility that the cycle time reduction is due to the corresponding reduction in throughput rate by building a simulation model of the two-stage system with infinite buffer sizes. The batch processor size was set at two. As the buffer sizes are unbounded, no job arrivals are rejected, and the throughput rate will be identical under the two serial processor policies. The arrival rate was varied from 0.5 to 0.9, and it was confirmed that the system under greedy-look-ahead policy still has reduced mean cycle time of jobs, albeit with a smaller rate of reduction. When the traffic intensity of the system is 0.9, the magnitude of the cycle time reductions are 1.0% when the batch processor is under no-idling policy, and 1.4% when the batch processor is under full-batch policy. Infinite buffer sizes cause a large number of jobs to enter the system, which increases the mean cycle time of jobs in the system regardless of control policy. This reduces the relative difference in the cycle time generated by the two policies.

Before the myopic and greedy-look-ahead policies are compared in greater detail, a combination of the greedy-look-ahead and balanced-look-ahead policy was also evaluated. This combined look-ahead policy uses the greedy look-ahead policy when the buffers in front of the batch processor are all less than the batch processor size, but switches to using the balanced-look-ahead policy when at least one of the buffers has queue size greater than or equal to the batch processor size. A direct comparison with the greedy-look-ahead policy revealed that behavior of the combined look-ahead policy is very similar to that of the greedy-look-ahead policy, with the combined look-ahead policy having slightly higher mean cycle time and throughput rate. Figure 5.15 plots the performance of the two control policies. Systems A and B have the batch processor under full-batch policy, while Systems C and D have the batch processor under no-idling policy. Systems A and C have the batch processor size at two, while Systems B and D have batch processor size at six. In an attempt to keep things simple, we opt to use the greedy-look-ahead policy for further comparisons.

In the next sets of experiments, we wish to further investigate the difference between the performance of the myopic and greedy-look-ahead policies. We keep the batch processor size at two, batch processor mean processing rate at 0.5 batches per unit time, and the buffer sizes

at seven. The serial processor processing rate is varied from 0.5 jobs per unit time to 1.5 jobs per unit time; when the serial processor processing rate is 0.5, the serial processor is the strong system constraint, and when the serial processor processing rate is 1.5, the batch processor is the strong system constraint. Table 5.6 contains the exact values assumed by the serial processor processing rate and the job arrival rate.

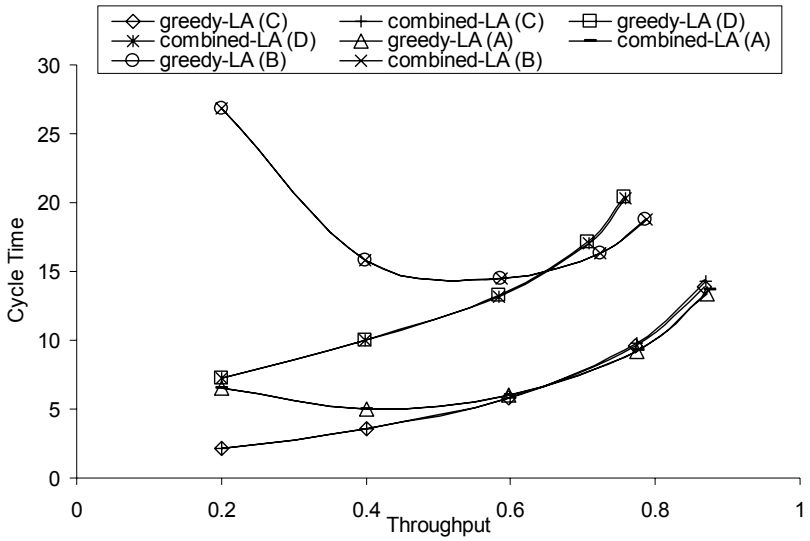


Figure 5.15: Performance comparison between the combined and the greedy-look-ahead policy.

Systems A and B have the batch processor under full-batch policy, while Systems C and D have the batch processor under no-idling policy. Systems A and C have the batch processor size at two, while Systems B and D have batch processor size at six. It is difficult to distinguish the performance of the two policies from one another.

When the serial processor processing rate is less than one job per unit time, the arrival rate can be strictly larger than the serial processor processing rate; while this situation is rarely found in manufacturing systems, an example where the mean processing rate is temporarily lower than the arrival rate is when several processors are in parallel, and some of these processors are down for a particular time interval.

TABLE 5.6: SUPPLEMENTARY EXPERIMENTAL SETUP

Batch processor policy	Serial processor processing rate (jobs per unit time)	Job arrival rate (jobs per unit time)
No-idling	0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5	0.2, 0.4, 0.6, 0.8, 1.0
Full-batch	0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5	0.2, 0.4, 0.6, 0.8, 1.0

TABLE 5.7: PERFORMANCE OF THE GREEDY LOOK-AHEAD POLICY, RELATIVE TO MYOPIC POLICY

		Cycle time reduction (%)					Throughput reduction (%)				
Serial processor processing rate (jobs / unit time)		0.6	0.8	1.0	1.2	1.4	0.6	0.8	1.0	1.2	1.4
Batch processor Policy	Job arrival rate (jobs / unit time)	% reduction in performance calculated as: 100% × (performance of myopic-performance of greedy look-ahead)/performance of myopic policy									
Full-batch	0.2	0.63	0.28	0.14	0.08	0.05	0.00	0.00	0.00	0.00	0.00
Full-batch	0.4	4.49	1.67	0.87	0.51	0.32	0.40	0.04	0.01	0.00	0.00
Full-batch	0.6	17.81	6.74	2.62	1.31	0.77	2.06	0.81	0.18	0.04	0.01
Full-batch	0.8	20.45	15.64	7.76	3.51	1.75	0.84	1.76	0.99	0.35	0.13
Full-batch	1.0	16.99	16.96	11.86	6.98	3.86	0.15	0.77	1.04	0.65	0.34
No-idling	0.2	0.25	0.16	0.11	0.08	0.06	0.00	0.00	0.00	0.00	0.00
No-idling	0.4	3.45	0.88	0.46	0.30	0.21	0.43	0.05	0.01	0.00	0.00
No-idling	0.6	16.86	5.65	1.75	0.75	0.42	2.11	0.84	0.19	0.05	0.01
No-idling	0.8	18.99	14.27	6.72	2.80	1.31	0.86	1.80	1.03	0.38	0.15
No-idling	1.0	15.20	15.37	10.80	6.21	3.50	0.17	0.82	1.10	0.71	0.38

Table 5.7 presents the percentage cycle time and throughput reduction for a system with the serial processor under greedy-look-ahead policy, compared to when the serial processor under myopic policy. Figure 5.16 and Figure 5.17 present the same set of data in graphical manner.

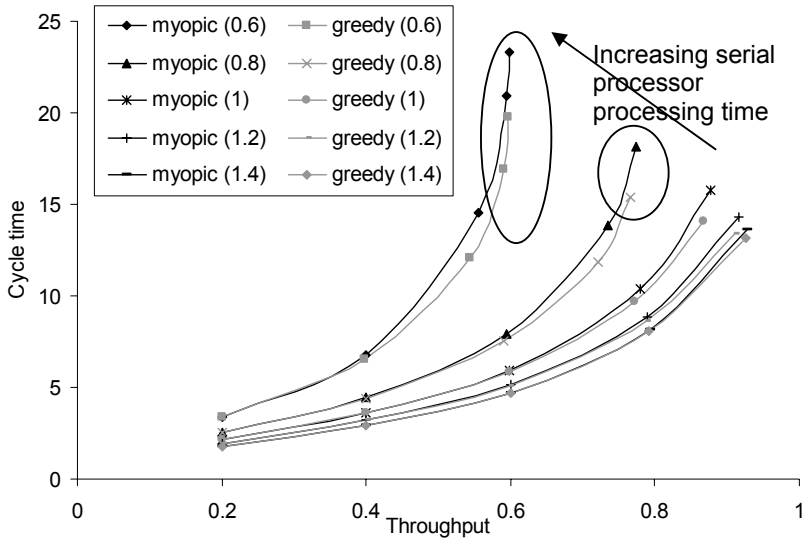


Figure 5.16: Sensitivity of the relative performance of two serial processor control policies to the mean processing rate of the serial processor, when the batch processor is under no-idling policy.

The batch processor size is two and the buffer levels are seven. The number beside the control policy is the mean processing rate in jobs/unit time. When the arrival rates are low, there is little difference between the two policies. The faster the serial processor, the higher the arrival rate needs to be for queues to form in front of the serial processor. This is why separation between the two policies is noticed only at high arrival rates. The encircled points correspond to cases where the arrival rate is strictly larger than the processing rate of the serial processor.

Regardless of the serial processor processing rate and job arrival rate, the cycle time reduction is larger and the throughput reduction is smaller when the batch processor is under full-batch policy, compared to when the batch processor is under no-idling policy. Let the job being processed by the serial processor (under the greedy look-ahead policy) be Job A, who belongs to Job Family *i*. The greedy look-ahead policy assumes that Job A will be processed in the next batch the batch processor will process. If the batch processor is currently idle or if the batch processor finishes after the serial processor, then Job A will join the next batch regardless of batch processor policy. If the batch processor finishes before the serial processor, then Job A will not be able to join the next batch if the batch processor is under no-idling policy. In contrast, if the batch processor is under full-batch policy, then Job A will join the next batch as long as there is less than one full batch of Job Family *i* in front of the batch processor.

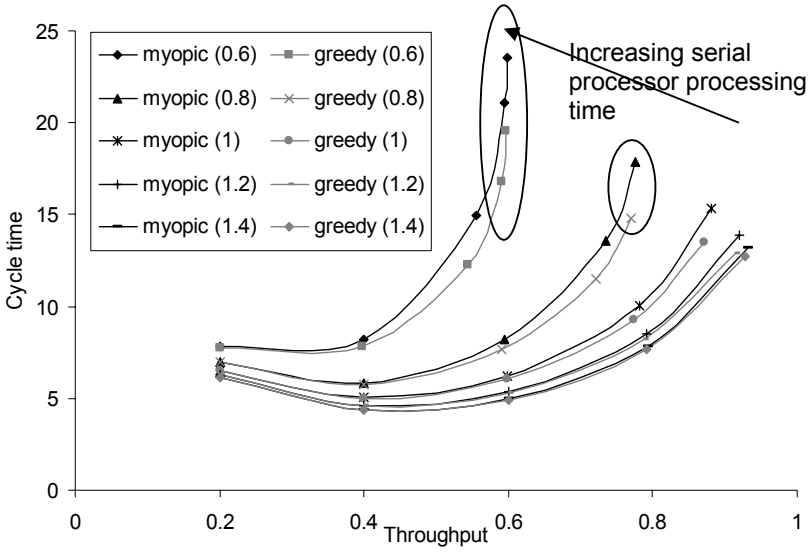


Figure 5.17: Sensitivity of the relative performance of two serial processor control policies to the mean processing rate of the serial processor, when the batch processor is under full-batch policy.

The batch processor size is two and the buffer levels are seven. The number beside the control policy is the mean processing rate in jobs/unit time. The encircled points correspond to cases where the arrival rate is strictly larger than the processing rate of the serial processor. The faster the serial processor processes jobs, the greater the cycle time reduction resulting from using the greedy look-ahead instead of the myopic policy. This is normally accompanied with greater reduction in throughput. However, when the arrival rate is strictly higher than the serial processor processing rate, the gap in throughput is narrowed down, as jobs continually arrive to replace any jobs that may have been rejected due to the greedy look-ahead policy.

When the arrival rate is low compared to the serial processor processing rate, it is difficult to distinguish the performance of the two control policies. Thus, the higher the serial processor processing rate, the higher the job arrival rate has to be before the performance of the greedy look-ahead and the myopic policy can be easily distinguished. When the serial processor processing rate is high compared to the arrival rate, queues rarely form in front of the serial processor, and the serial processor rarely has a choice as to what job family is to be processed. As the job arrival rate increases, queues in front of the serial processor form, and the two serial processor policies diverge in the job families they choose to process. Since the greedy look-ahead policy ignores the serial processor buffer levels, the reduction in throughput rate also generally increases with increasing job arrival rate. This pattern holds true if the arrival rate is not higher than the serial processor processing rate. However, when jobs arrive at a faster rate than the serial processor can remove them (when the serial processor utilization is greater than one), the buffer levels at the serial processor tend to be full. This reduces the impact on throughput of the greedy look-ahead policy ignoring the serial processor buffer level, as starvation is unlikely to occur. Secondly, because the probability of having a large number of jobs in front of the serial processor is high, the system cycle time increases, regardless of serial processor policy. This reduces the relative difference in cycle time between the two policies.

5.7 Behavior of Three-Stage System

In this section we discuss the experimental setup used, and analyze the behavior of the three-stage manufacturing system. The goal is to determine whether constraining the production of another processor further upstream of the batch processor will still result in reducing the mean cycle time for jobs inside the system, without excessive throughput loss.

5.7.1 Experimental Design

Due to the large rate of state space expansion with respect to the buffer sizes, only a limited number of experiments can be performed, using the Markov models. We set all buffer sizes to four and the batch processor size to three. Furthermore, we only consider systems where all processors have identical maximum throughput rates. The mean processing time of the serial processor is held constant at 1 job / time unit, and the other parameters are derived from:

λ = traffic intensity *TI* of serial processor / 2

μ_B = *TI* of serial processor / (*TI* of batch processor × batch processor size)

For each possible batch processor control policy, the base case has the two upstream processors under myopic control policy. Table 5.8 contains the other control policies evaluated.

TABLE 5.8: EXPERIMENTAL SETUP FOR THREE-STAGE SYSTEM

Purpose of experiment	three-stage control policies considered	Batch processor size	Buffer size	Job arrival rates
Compare performance measures of system under different control policies	Myopic-myopic-no-idling Myopic-greedy look-ahead-no-idling Greedy look-ahead-greedy look-ahead-no-idling Balanced look-ahead-greedy look-ahead-no-idling Myopic-myopic- full-batch Myopic-greedy look-ahead- full-batch Greedy look-ahead-greedy look-ahead- full-batch Balanced look-ahead-greedy look-ahead-full-batch	2, 3	3, 4	0.2, 0.4, 0.6, 0.8, 1.0

5.7.2 Sensitivity of Model to Different Serial Processor Policies

Figure 5.18 plots the performance of a three-stage system with the batch processor under no-idling policy. The buffer sizes are fixed at four, and the batch processor size is set at two. The graphs for the same systems with different values for buffer sizes and batch processor sizes are similar.

Table 5.9 contains the relative performance of the three policies for which the second processor is under greedy look-ahead policy, as compared against the myopic-myopic policy. The table reveals that the balanced-greedy look-ahead policy has the greatest cycle time reduction, and also has the least throughput reduction among all three. This is followed by the

greedy-greedy look-ahead, and by the myopic-greedy look-ahead. All three policies have lower mean cycle time than the myopic-myopic policy.

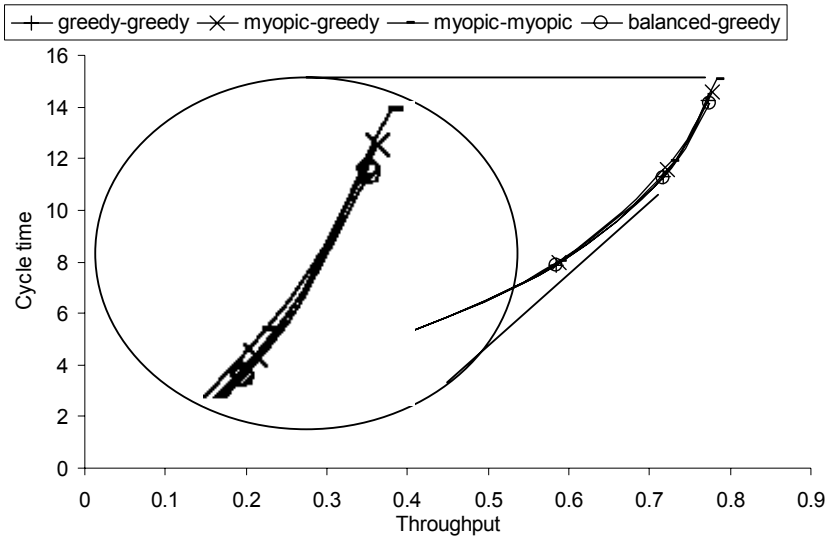


Figure 5.18: Performance of a three-stage system with two upstream processors and a downstream batch processor, with the batch processor is under no-idling policy.

Because the buffer sizes are small, there is little separation between the performances of the various upstream processor policies evaluated. However, each of the three policies where the serial processor immediately upstream of the batch processor uses a greedy look-ahead policy can cause a significant reduction in cycle time over the benchmark myopic-myopic policy.

TABLE 5.9:RELATIVE PERFORMANCE OF POLICY COMBINATIONS VERSUS MYOPIC-MYOPIC FOR THREE STAGE SYSTEM WITH BATCH PROCESSOR UNDER NO-IDLING POLICY.

Traffic Intensity	Myopic-greedy look-ahead		Greedy-greedy look-ahead		Balanced-greedy look-ahead	
	Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction
0.2	0	0.068%	0	0.097%	0	0.097%
0.4	0.005%	0.304%	0.083%	0.603%	0.078%	0.592%
0.6	0.173%	0.987%	0.609%	2.281%	0.578%	2.358%
0.8	0.633%	2.851%	1.364%	5.191%	1.282%	5.569%
1.0	0.724%	3.196%	1.353%	5.475%	1.166%	5.953%

We observe that the throughput loss increases at a higher rate than the cycle time reduction, as we move from a myopic-greedy look-ahead policy to a greedy-greedy look-ahead policy. Furthermore, the incremental cycle time reduction obtained in "looking ahead" an additional upstream processor shrinks. The superiority of the balanced-greedy look-ahead policy for the set of parameters used in the experimental setup can be understood in this manner: the job that the first serial Processor (M_0) is currently processing will have to pass through the second

serial Processor (M_1) before reaching the batch processor. Depending on the policy at M_1, the job may also have to queue in front of M_1. Thus, the total time it takes for the job currently at M_0 to reach the batch processor is at least the sum of the processing times of the two serial processors. This increases the probability that the batch processor has either formed a full batch, or has already loaded a new batch for processing, before the job currently at M_0 can even reach the queue for the batch processor. This implies that it would be better for M_0 to process jobs, not with the batch processor's next batch in mind, but the batch following the batch processor's next batch. The balanced look-ahead policy at M_0 captures the essence of this idea, and results in larger cycle time reduction, with smaller throughput loss. At higher batch processor sizes, supplementary simulation experiments show that the performance of the balanced-greedy look-ahead policy combination, relative to the myopic-myopic policy, will degrade. When the batch processor's size is increased, the mean time to process a batch also increases, if the maximum processing rates of each processor is identical. By ignoring the buffer levels at the first serial processor when it is under the balanced-look-ahead policy, the extended processing times at the downstream batch processor increases the probability that one of the buffers in front of the first serial processor will become full, which increases the probability that external job arrivals get rejected by the system. This is also true when the first serial processor is under greedy look-ahead policy. However, the adverse effects are partially mitigated by the similarity of job families being processed by the first and second serial processor (under greedy-greedy look-ahead policy combination), which reduces the probability of external job arrivals getting blocked.

The same observations made when the batch processor is under no-idling policy also hold true when the batch processor is under full-batch policy. The differences between the benchmark myopic-myopic policy and the three look-ahead policies are greater when the batch processor is under full-batch policy. Figure 5.19 illustrates the performance of the system under four policies, and the graph for the system under myopic-myopic policy is clearly above the graphs for the other three policies, while Table 5.10 contains the relative performance of the three policies versus the myopic-myopic policy.

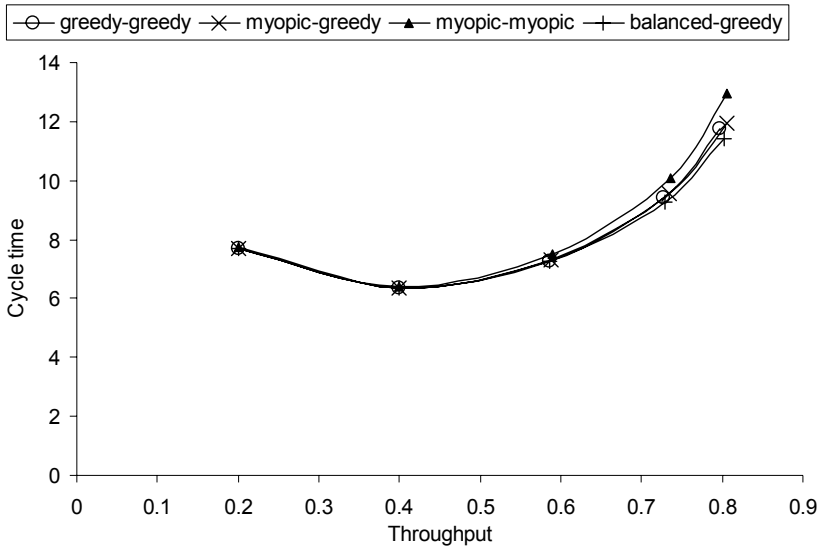


Figure 5.19: Performance of a three-stage system with two upstream processors and a downstream batch processor.

The buffer sizes are four, the batch processor size is two, and the batch processor is under full-batch policy. There is considerable separation in the performance of systems with different upstream processor policies. The relative rankings of the policy combinations remain the same.

TABLE 5.10:RELATIVE PERFORMANCE OF POLICY COMBINATIONS VERSUS MYOPIC-MYOPIC FOR THREE STAGE SYSTEM WITH BATCH PROCESSOR UNDER FULL-BATCH POLICY.

Traffic Intensity	Myopic-greedy look-ahead		Greedy-greedy look-ahead		Balanced-greedy look-ahead	
	Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction
0.2	0	0.141%	0	0.150%	0	0.137%
0.4	0.005%	0.793%	0.073%	0.914%	0.073%	0.848%
0.6	0.121%	2.266%	0.537%	2.946%	0.513%	3.180%
0.8	0.332%	5.440%	1.212%	6.841%	1.041%	8.212%
1.0	-0.027%	7.986%	0.982%	9.289%	0.470%	11.818%

5.8 Chapter Summary

In this chapter, we investigate the feasibility of controlling an upstream processor based on the anticipated needs of the batch processor. In doing so, we also characterize the performance of a system that has a downstream batch processor and at least one upstream serial processor under differing sets of parameters.

We compare the steady-state performance of three different serial processor policies, under two different batch processor policies. We consider a continuous time, discrete state Markov

chain model of a two-stage system, with an upstream serial processor. The relationship between the performance of the system and the parameters varied is generally invariant of the control policy used on either processor. Furthermore, the system behavior can often be inferred from the behavior of systems with only serial processors (see: [Gershwin 2002]).

From the numerical experiments, we also observe the following:

- The generalization that increased throughput leads to increased cycle time is not always true for the two-stage system. When the batch processor is under full-batch policy, it can take a long time to form full batches when job arrivals are infrequent. Increasing the job arrival rate can simultaneously increase system throughput and decrease mean cycle time. In comparison, when the batch processor uses no- idling policy, the cycle time increases monotonically as the arrival rate is increased.
- The system performance deteriorates when batch processors are larger but slower. Since there is no guarantee that a full batch will always be waiting for the batch processor whenever it becomes available, performance deterioration occurs when the batch processor size is increased. Thus, larger batch processor sizes make it less likely the system will be well-approximated by a system with only serial processors.
- In comparing the no-idling and full-batch policy, we confirm the intuition that the no-idling policy would be preferred if the mean WIP level in front of the batch processor is low, while the full-batch policy would be preferred if the mean WIP level in front of the batch processor is high.
- Because WIP accumulation in front of the batch processor has to propagate to the serial processor buffer before jobs get rejected, it generally is more important to manage the WIP levels in front of the serial processor if the sole objective is to maximize throughput. This is why the myopic policy generally has greater throughput than either look-ahead policy. This difference is magnified when the serial processor become busier and when the buffer sizes are larger. As the batch processor traffic intensity is increased, the difference is reduced.

- The increase in throughput from the myopic policy comes with a price: increased cycle time. The myopic policy generally has the largest cycle time among the control policies evaluated. The greedy-look-ahead policy has lower cycle time because it hastens the formation of larger batches for the job type that will be produced next. When the batch processor uses full-batch policy, full batches are formed faster. When no-idling policy is in effect at the batch processor, more jobs are batched together.
- The higher the batch processor size, the more important it is to form large batches quickly. This is where the greedy-look-ahead policy works best. Since the balanced look ahead policy is designed for the cases where a buffer in front of the batch processor has already exceeded the batch processor size Q, the efficacy of the balanced look-ahead strategy in reducing cycle time and increasing throughput decreases as the batch processor size increases.

Although there is no policy that consistently outperforms the other two in both throughput and cycle time, we show that the concept of controlling the serial processor as a function of the anticipated needs of the batch processor can reduce the mean cycle time of jobs in the system. also determine under which conditions a particular concept embodied by a control policy is more effective, allowing us to combine aspects of these policies to form better policies.

A three-stage model with an additional upstream serial processor is also created, to determine whether constraining the control of the upstream processor with respect to batch processor can be extended to processors further upstream. When an additional processor upstream is constrained to process jobs according to the anticipated needs of the batch processor, the incremental cycle time reduction obtained is smaller than the initial cycle time reduction obtained when only the processor immediately upstream of the batch processor is constrained. This implies that we can concentrate on the one or two stages immediately preceding the batch processor, while preserving a large proportion of the potential cycle time reduction.

In wafer fabs, it is also possible that ovens have batch processors (typically with smaller sizes) as their upstream processors. We extend the analysis of these types of systems in the next chapter.

Chapter 6 Control of Upstream Batch Processor with a Downstream Batch Processor

6.1 Introduction

In semiconductor wafer fabs, the processor upstream of the diffusion furnace can also be a batch processor. A diffusion oven can also be fed by the clean station, which have size two. We wish to determine whether the conclusions derived when the feeder processor is a serial processor also hold true when the upstream processor is also a batch processor. We only consider the case where the upstream batch processor has size two, and the downstream batch processor has size greater than two, as this is what has been observed in the wafer fab. A more comprehensive characterization of a two-stage system with two batch processors processing a single job family has been initiated in [Chang and Gershwin 2005].

6.2 Problem Statement

A manufacturing system is comprised of two processors and processes jobs that belong to one of m different job families. All jobs visit the processors in the same order. All jobs from the

same family are identical. Both processors are perfectly flexible and perfectly reliable batch processors. The i^{th} processor can process up to Q_i jobs, all belonging to the same family, at an instance, and processing time is independent of the number of jobs loaded into the batch processor. Job preemption is not allowed.

In front of each Processor M_i are m different Buffers B_{ij} of finite size C_{ij}; each Buffer B_{ij} holds jobs belonging to Family j waiting to be processed by Processor M_i. It is assumed that any control policy at M_1 ensures that any batch J from Job Family j currently being processed at M_1 can be accommodated by B_{2j} before processing commences. Figure 6.1 illustrates the manufacturing system when there are $m = 2$ job families being processed, and with $Q_1 = 2$ jobs.

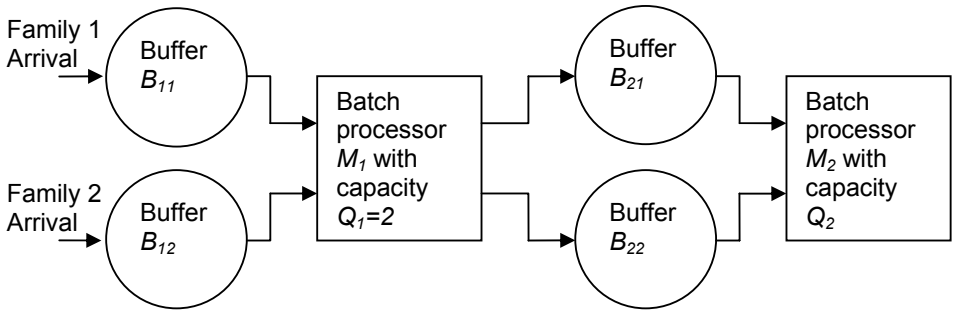


Figure 6.1: A two-stage manufacturing system with a batch-batch configuration. The system is processing two job families, and has two buffers, one for each job family, in front of every processor.

When Buffer B_{1j} is not full, the arrival process for Job Family j into the system is a stochastic process with a time-invariant distribution. We wish to determine the long-run mean performance of manufacturing systems of this nature under several simple two-stage control policies

6.3 Two-stage Model Development

We modify the Markovian models in Chapter 5 to evaluate the performance of several simple control policies at the upstream batch processor for two- and three-stage systems. Throughout this chapter, the term 'batch processor' shall be used to refer to the downstream batch

processor, while the term 'feeder processor' shall be used to refer to the upstream batch processor with size two, unless specified otherwise.

6.3.1 Development of Two-Stage Markovian Model

We assume that the manufacturing system only processes two job families. The batch processor uses the same control policies as those used in Chapter 5. The full-batch policy will only process a full batch while the no-idling policy will process a batch, regardless of how many jobs belong to the batch. The feeder processor control policies we use are analogous to the serial processor control policies used in Chapter 5, modified to account for the feeder processor being a batch processor.

6.3.1.1 Model Assumptions

We make the following assumptions for the manufacturing system described in Section 6.2:

- The manufacturing system processes two job families.
- The time between external arrivals of jobs belonging to Family i is exponentially distributed, with arrival rate λ. If the Buffer B_{1i} is full, any arrivals of Job Family i are lost.
- The time required to process a job belonging to Job Family i at the feeder processor is exponentially distributed, with service rate μ_f. The feeder processor will not process a batch belonging to Job Family i if buffer B_{2i} does not have enough space for all jobs in a batch. The minimum number of jobs inside a batch is dependent on the feeder processor control policy.
- The time required to process a batch at the batch processor is exponentially distributed, with service rate μ_B, regardless of the job family currently being processed.

6.3.1.2 Feeder Processor Control Policies

When the upstream processor is a batch processor, each control policy needs to address two questions: what is the minimum number of jobs required to form a batch, and which job family

needs to be processed. To determine the minimum number of jobs required to form a batch, we assume that the feeder processors uses either no-idling policy, or full-batch policy.

To determine which job family needs to be processed, we create policies analogous to those discussed in Chapter 5. However, if we do not take into account the number of jobs that can be processed by the feeder processor, the mean batch size might be small and throughput loss may be excessive. Thus, we assume two common properties among all feeder processor control policies considered:

- The feeder processor will not process a batch if not all jobs can fit into the appropriate batch processor buffer.
- Each feeder processor policy first determines the maximum number of jobs that can be processed if a batch of Job Family i is processed. Let Ω_i be the maximum number of jobs belonging to Job Family i the feeder processor can batch together. If $\Omega_1 > \Omega_2$, the feeder processor prefers to process Job Family One, and if $\Omega_1 < \Omega_2$, the feeder processor prefers to process Job Family Two. It is only when $\Omega_1 = \Omega_2$ that the different control policies would choose different job families to process.

To avoid confusion, a parenthesis shall be used to differentiate the feeder processor control policy from that of a two-stage system with a serial processor and a batch processor. For example, (myopic-no-idling) policy refers to the feeder processor policy, while myopic-no-idling policy refers to the control policy of a two-stage system. The first term dictates what batch is to be processed and the second term dictates when batches are formed.

6.3.1.2.1 Myopic control policy variant

Let M_i be the feeder processor. If $\Omega_1 = \Omega_2$, then M_i prefers to process Job Family One if $b_{i1} \geq b_{i2}$, and prefers to process Job Family Two if $b_{i2} > b_{i1}$.

6.3.1.2.2 Greedy look-ahead policy

Let M_i be the feeder processor and let M_k be the batch processor. If $\Omega_1 = \Omega_2$, then M_i processes Job Family One if $b_{k1} \geq b_{k2}$, and processes Job Family Two if $b_{k2} > b_{k1}$.

6.3.1.2.3 Balanced look-ahead policy

Let M_i be the feeder processor and let M_k be the batch processor. If $\Omega_1 = \Omega_2$, then M_i processes Job Family One if $b_{k1} \leq b_{k2}$, and processes Job Family Two if $b_{k2} < b_{k1}$.

6.3.1.3 Batch Processor Control Policies

The batch processor control policies are identical to what is described in Section 5.3.1.4.

6.3.1.4 State Representation

Let the variables be defined as:

$b_{1i}(t)$ – the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the feeder Processor M_1 at Time Instance t, *including* the job currently being processed.

$b_{2i}(t)$ - the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the batch processor at Time Instance t, *excluding* the job/s currently being processed.

$S_f(t)$ – The status of the feeder Processor M_1 at Time Instance t. If $S_f(t) = 0$, M_1 is idle. If $S_f(t) = i>0$, M_1 is processing a batch belonging to Job Family i.

$S_b(t)$ - The status of the batch processor at Time Instance t. If $S_b(t) = 0$, the batch processor is idle. If $S_b(t) = 1$, the batch processor is processing a batch.

When the feeder processor is under a full-batch policy variant, the number of jobs being processed by the feeder processor is constant. When the feeder processor is under a no-idling policy variant, there exists a need to keep track of the number of jobs. Let:

$N(t)$ – the number of jobs currently being processed by the feeder processor at Time Instance t.

Then, at any instance t, the state $X(t)$ is described by the vector $X(t) = (b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t), S_f(t), S_b(t), N(t))$, if the feeder processor is under a no-idling policy variant and $X(t) = (b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t), S_f(t), S_b(t))$, if the feeder processor is under a full-batch policy variant. We can also augment $X(t)$ to include $N(t)$ when the feeder processor is under a full-batch policy variant, with the condition that $N(t) = 0$ if $S_f(t) = 0$, and $N(t) = Q_1$ if $S_f(t) > 0$.

Let Q be the batch processor size and let C_{ij} be the maximum buffer size of Buffer B_{ij}, then the following rules are used in generating the state space when the feeder processor is under full-batch policy:

- $S_f(t) \neq 0$ if $(b_{11}(t) \geq Q$ *AND* $b_{21}(t) \leq C_{21} - Q)$ *OR* $(b_{12}(t) \geq Q$ *OR* $b_{22}(t) \leq C_{22} - Q)$
- $S_f(t) \neq 1$ if $(b_{11}(t) < Q$ *AND* $b_{21}(t) > C_{21} - Q)$
- $S_f(t) \neq 2$ if $(b_{12}(t) < Q$ *OR* $b_{22}(t) > C_{22} - Q)$
- $S_b(t) = 0$ if and only if $(b_{21}(t) = 0$ *AND* $b_{22}(t) = 0$ *AND batch processor policy is no-idling)* *OR* $(b_{21}(t) < Q$ *AND* $b_{22}(t) < Q$ *AND batch processor policy is full-batch)*

The following rules are used when the feeder processor is under no-idling policy:

- $N(t) = 0$ if and only if $S_f(t) = 0$
- $N(t) \leq max\ \{Q, b_{11}(t), C_{21} - b_{21}(t))$ *if* $S_f(t) = 1$
- $N(t) \leq max\ \{Q, b_{12}(t), C_{22} - b_{22}(t))$ *if* $S_f(t) = 2$
- $S_f(t) \neq 0$ if $(b_{11}(t) > 1$ *AND* $b_{21}(t) < C_{21})$ *OR* $(b_{12}(t) > 1$ *OR* $b_{22}(t) < C_{22})$
- $S_f(t) \neq 1$ if $(b_{11}(t) = 0$ *AND* $b_{21}(t) = C_{21})$
- $S_f(t) \neq 2$ if $(b_{12}(t) = 0$ *OR* $b_{22}(t) = C_{22})$
- $S_b(t) = 0$ if and only if $(b_{21}(t) = 0$ *AND* $b_{22}(t) = 0$ *AND batch processor policy is no-idling)* *OR* $(b_{21}(t) < Q$ *AND* $b_{22}(t) < Q$ *AND batch processor policy is full-batch)*

The manner in which the transition probabilities are obtained is found in Appendix D.2 .

6.3.2 Development of Three-Stage Markovian Model

To determine whether the inferences from a two-stage model is also true for larger manufacturing systems, a three-stage model involving a serial processor followed by a batch processor with size two (still called the feeder processor) with a downstream batch processor was also generated.

6.3.2.1 Problem formulation for three-stage model

A manufacturing system is comprised of three perfectly flexible processors and processes jobs that belong to one of m different job families. All jobs visit the processors in the same order. All jobs from the same family are identical. The first Processor M_0 is a serial processor, the second Processor M_1 is a batch processor with size two. The third Processor M_2 is a batch

processor with size $Q > 2$ jobs. M_1 and M_2 can only process jobs belonging to the same family as a batch, and processing time is independent of the number of jobs loaded. All processors are assumed to be perfectly reliable, and job preemption is not allowed.

In front of each Processor M_i are m different Buffers B_{ij} of finite size C_{ij}; each Buffer B_{ij} holds jobs belonging to Family j waiting to be processed by processor M_i. Figure 6.2 illustrates the manufacturing system when there are $m = 2$ job families being processed.

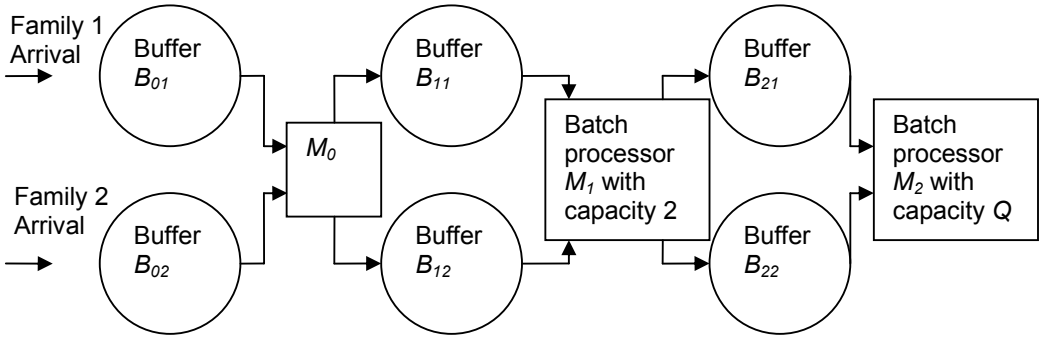


Figure 6.2: A three-stage manufacturing system with a serial-batch-batch configuration.

Processor M_0 is a serial processor. In this example, the system is processing two job families, and would have two buffers, one for each job family, in front of every processor.

When Buffer B_{0j} is not full, the arrival process for Job Family j into the system is a stochastic process with a time-invariant distribution. We determine the long-run mean performance of manufacturing systems of this nature under simple three-stage control policies.

6.3.2.2 Model Assumptions

We make the following assumptions for the manufacturing system in Section 6.3.2:

- The manufacturing system processes two job families.
- The amount of time between external arrivals of Job Family i is exponentially distributed with mean $1/\lambda$. If Buffer B_{0i} is full, any Job Family i arrivals are lost.
- The amount of time required to process a job belonging to Job Family i at the serial processor is exponentially distributed with mean $1/\mu_s$. The serial processor will not process a job belonging to Job Family i if Buffer B_{1i} is full.

- The amount of time required to process a batch belonging to Job Family *i* at the feeder processor is exponentially distributed with mean $1/\mu_f$. The feeder processor will only process a batch of X jobs belonging to Job Family *i* if Buffer B_{2i} can fit X jobs.
- The amount of time required to process a batch at the batch processor is exponentially distributed with mean $1/\mu_B$, regardless of the job family currently being processed.

6.3.2.3 Processor Control Policies

The serial processor control policies used are identical to those described in Section 5.3.1.3. The feeder processor control policies used are identical to those described in Section 6.3.1.2. The batch processor control policies used are identical to those described in Section 5.3.1.4.

6.3.2.4 State Representation

Let the variables be defined as:

$b_{ki}(t)$ – the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the processor M_k (k=0, 1) at Time Instance t, *including* the job/s currently being processed.

$b_{2i}(t)$ - the amount of jobs belonging to Job Family i (i = 1,2) stored in front of the batch processor at Time Instance t, *excluding* the job/s currently being processed.

$S_s(t)$ – The status of the serial processor M_0 at time instance t. If $S_s(t) = 0$, M_0 is idle. If $S_s(t) = i>0$, M_0 is processing a job belonging to Job Family i.

$S_f(t)$ – The status of the feeder processor M_1 at Time Instance t. If $S_f(t) = 0$, M_1 is idle. If $S_f(t) = i>0$, M_1 is processing a job belonging to Job Family i.

$S_b(t)$ - The status of the batch processor at Time Instance t. If $S_b(t) = 0$, the batch processor is idle. If $S_b(t) = 1$, the batch processor is processing a batch.

$N(t)$ – the number of jobs currently being processed by the feeder processor at Time Instance t.

Then, at any Time Instance t, the state $X(t)$ is described by the vector

$X(t) = (b_{01}(t), b_{02}(t), b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t) , S_s(t), S_f(t), S_b(t), N(t))$, if the feeder processor is under a no-idling policy variant and, $X(t) = (b_{01}(t), b_{02}(t), b_{11}(t), b_{12}(t), b_{21}(t), b_{22}(t) , S_s(t), S_f(t), S_b(t))$, if the feeder processor is under a full-batch policy variant. A state augmentation

procedure similar to that in Section 6.3.1.4 can also be performed such that the state representation is identical under all feeder processor policies.

Let Q be the batch processor size and let C_{ij} be the maximum buffer size of Buffer B_{ij}, then the following rules are used in generating the state space regardless of feeder processor policy:

- $S_s(t) \neq 0$ if $(b_{01}(t) > 0$ *AND* $b_{11}(t) < C_{11})$ *OR* $(b_{02}(t) > 0$ *OR* $b_{12}(t) < C_{12})$
- $S_s(t) \neq 1$ if $(b_{01}(t) = 0$ *OR* $b_{11}(t) = C_{11})$
- $S_s(t) \neq 2$ if $(b_{02}(t) = 0$ *OR* $b_{12}(t) = C_{12})$
- $S_b(t) = 0$ if and only if $(b_{21}(t) = 0$ *AND* $b_{22}(t) = 0$ *AND batch processor policy is no-idling) OR* $(b_{21}(t) < Q$ *AND* $b_{22}(t) < Q$ *AND batch processor policy is full-batch)*

When the feeder processor is under full-batch policy, the following additional rules apply:

- $S_f(t) \neq 0$ if $(b_{11}(t) \geq Q$ *AND* $b_{21}(t) \leq C_{21} - Q)$ *OR* $(b_{12}(t) \geq Q$ *OR* $b_{22}(t) \leq C_{22} - Q)$
- $S_f(t) \neq 1$ if $(b_{11}(t) < Q$ *AND* $b_{21}(t) > C_{21} - Q)$
- $S_f(t) \neq 2$ if $(b_{12}(t) < Q$ *OR* $b_{22}(t) > C_{22} - Q)$

When the feeder processor is under no-idling policy, the following additional rules apply:

- $N(t) = 0$ if and only if $S_s(t) = 0$
- $N(t) \leq max\ \{Q, b_{11}(t), C_{21} - b_{21}(t))$ *if* $S_s(t) = 1$
- $N(t) \leq max\ \{Q, b_{12}(t), C_{22} - b_{22}(t))$ *if* $S_s(t) = 2$
- $S_f(t) \neq 0$ if $(b_{11}(t) > 1$ *AND* $b_{21}(t) < C_{21})$ *OR* $(b_{12}(t) > 1$ *OR* $b_{22}(t) < C_{22})$
- $S_f(t) \neq 1$ if $(b_{11}(t) = 0$ *AND* $b_{21}(t) = C_{21})$
- $S_f(t) \neq 2$ if $(b_{12}(t) = 0$ *OR* $b_{22}(t) = C_{22})$

The manner in which the transition probabilities are obtained is illustrated in Appendix E.2

6.3.3 Verification of Markovian Models-Comparison with Simulation

The methodology and results of verifying the Markov models are discussed in this section.

6.3.3.1 Comparison Methodology

The comparison methodology is similar to that discussed in Section 5.5.1. Table 6.1 lists the various two-stage simulation models built. The feeder processor size is held constant at two jobs. The feeder processor mean processing time is kept at one time unit. Thus, the theoretical maximum throughput rate is two jobs per unit time. Each problem instance was simulated for 11000 time units; the first 1000 time units was used as the warm-up period, and the remaining 10000 time units was used to collect data on the models. All models found in Table 6.1 and Table 6.2 are started empty; all processors are idle and all buffers are empty at the start of the simulation. This is important, since some of the Markov models do not exhibit ergodic behavior, and the steady state distribution is dependent on the initial state of the system.

TABLE 6.1: TWO-STAGE SIMULATION MODELS COMPARED WITH MARKOV MODELS

Two-stage control policy	traffic intensity	Other parameters (batch processor size Q and maximum buffer size B)	No. of problem instances
(Myopic-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Myopic-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Greedy-look-ahead-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Greedy-look-ahead-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Balanced-look-ahead-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Balanced-look-ahead-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 4$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Myopic-no-idling)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Myopic- no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Greedy-look-ahead-no-idling) -no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Greedy-look-ahead-no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Balanced-look-ahead-no-idling) -no-idling	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15
(Balanced-look-ahead-no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	$Q = 3$ and $B = 3$; $Q = 3$ and $B = 7$; $Q = 6$ and $B = 7$	15

Table 6.2 lists the various three-stage simulation models built. The batch processor size is set at three, the mean processing time of the serial processor is set to one; the maximum throughput rate is one job / time unit.

For each problem instance, 100 runs are performed. The throughput is measured directly, while cycle time was obtained through Little's Law by tracking the number of jobs in the system. This is the same manner in which cycle time is obtained under the Markov models. We compare the mean throughput and cycle time obtained from these 100 runs against the values obtained through the Markov model. For each set of parameters, the percentage difference between the simulation model throughput, τ_S, and the Markov model throughput, τ_M, is: $200 * |\tau_S - \tau_M| / (\tau_s + \tau_M)$. The percentage difference between the simulation model cycle time and Markov model cycle time is obtained in a similar manner.

TABLE 6.2: THREE-STAGE SIMULATION MODELS COMPARED WITH MARKOV MODELS

Three-stage control policy	traffic intensity	Number of instances
myopic-(myopic-full-batch)-full- batch	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(greedy-look-ahead-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
Greedy-look-ahead-(greedy-look-ahead-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
Balanced-look-ahead-(greedy-look-ahead-full-batch)-full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(myopic-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(greedy-look-ahead-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
greedy-(greedy-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
Balanced-look-ahead-(greedy-full-batch)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(myopic-no-idling) -full- batch	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(greedy-look-ahead- no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
Greedy-look-ahead-(greedy-look-ahead- no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
Balanced-look-ahead-(greedy-look-ahead- no-idling) -full-batch	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(myopic- no-idling) -no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
myopic-(greedy-look-ahead- no-idling)-no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
Greedy-look-ahead-(greedy-look-ahead- no-idling) -no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5
Balanced-look-ahead-(greedy-look-ahead- no-idling) -no-idling	0.2, 0.4, 0.6, 0.8, 1.0	5

6.3.3.2 Comparison Results

Table 6.3 lists the mean and maximum throughput and cycle time differences between the Markov and simulation models. As also observed in Section 5.5.2, the difference between the two families of models is small. The differences in the obtained cycle time values are also larger than the throughput differences.

TABLE 6.3: COMPARISON BETWEEN MARKOV MODEL AND SIMULATION MODELS

Two-stage control policy	Mean % throughput difference	Max. % throughput difference	Mean % cycle time difference	Max. % cycle time difference
(Myopic-full-batch)-no-idling	0.121	0.183	0.290	0.364
(Myopic-full-batch)-full-batch	0.145	0.230	0.222	0.466

(Greedy-look-ahead-full-batch)-no-idling	0.096	0.157	0.346	0.666
(Greedy-look-ahead-full-batch)-full-batch	0.148	0.283	0.274	0.590
(Balanced-look-ahead-full-batch)-no-idling	0.177	0.276	0.230	0.477
(Balanced-look-ahead-full-batch)-full-batch	0.168	0.248	0.211	0.443
(Myopic-no-idling)-no-idling	0.120	0.177	0.277	0.549
(Myopic-no-idling)-full-batch	0.147	0.248	0.183	0.426
(Greedy-look-ahead-no-idling)-no-idling	0.105	0.186	0.301	0.468
(Greedy-look-ahead-no-idling)-full-batch	0.130	0.244	0.220	0.654
(Balanced-look-ahead-no-idling)-no-idling	0.166	0.242	0.218	0.435
(Balanced-look-ahead-no-idling)-full-batch	0.174	0.274	0.229	0.523

Table 6.4 contains the mean and maximum differences in throughput and cycle time for comparisons performed on specific control policies of the three-stage manufacturing system, while Figure 6.3 illustrates the closeness of the throughput and cycle time values obtained from a Markov model and its corresponding simulation model.

TABLE 6.4: COMPARISON BETWEEN RESULTS OF MARKOV MODELS AND SIMULATION MODELS

Three-stage control policy	Mean % throughput difference	Max. % throughput difference	Mean % cycle time difference	Max. % cycle time difference
myopic-(myopic-full-batch)-full- batch	0.145	0.225	0.327	0.506
myopic-(greedy-look-ahead-full-batch)-full-batch	0.104	0.229	0.263	0.470
Greedy-look-ahead-(greedy-look-ahead)-full-batch-full-batch	0.169	0.255	0.245	0.366
Balanced-look-ahead-(greedy-look-ahead)-full-batch-full- batch	0.186	0.265	0.259	0.500
myopic-(myopic-full-batch)-no-idling	0.149	0.328	0.562	1.347
myopic-(greedy-look-ahead-full-batch)-no-idling	0.161	0.253	0.528	1.057
greedy-(greedy-full-batch)-no-idling	0.161	0.303	0.533	1.007
Balanced-look-ahead-(greedy-look-ahead-full-batch)-no-idling	0.167	0.317	0.479	1.012
myopic-(myopic-no-idling) -full- batch	0.118	0.166	0.192	0.359
myopic-(greedy-look-ahead- no-idling) -full-batch	0.160	0.342	0.255	0.556
Greedy-look-ahead-(greedy-look-ahead- no-idling) -full-batch	0.112	0.175	0.201	0.292
Balanced-look-ahead-(greedy-look-ahead-no-idling) -full- batch	0.159	0.273	0.297	0.520
myopic-(myopic- no-idling) -no-idling	0.129	0.272	0.525	0.829
myopic-(greedy-look-ahead- no-idling) -no-idling	0.106	0.182	0.518	0.868
Greedy-look-ahead-(greedy-look-ahead- no-idling) -no-idling	0.162	0.247	0.458	0.717
Balanced-look-ahead-(greedy-look-ahead-no-idling)-no-idling	0.236	0.462	0.631	1.300

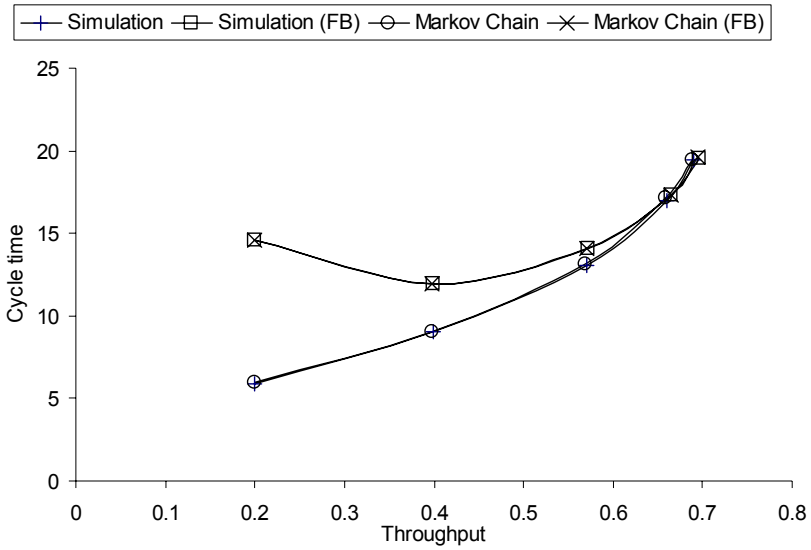


Figure 6.3: Comparing the cycle time and throughput from Markov and simulation models, when the policy is myopic-(myopic-no-idling)-no-idling or myopic-(myopic-no-idling)-full-batch.

The pair of convex figures corresponds to the batch processor using full-batch policy. The system parameters are buffer sizes of four and batch processor size of three. The results of the two model families (Markov and simulation) are very similar.

The small percentage differences in throughput and cycle time indicate good agreement between the two model families.

6.4 Behavior of Two-Stage System

In this section we discuss the experimental setup used, and analyze the behavior of the two-stage manufacturing system under different conditions. Similar to the work done in Section 5.6, we initially characterize the performance of the model under various control policies, before comparing the performance of the system under differing control policies.

6.4.1 Experimental Design

A total of twelve two-stage control policies are evaluated: there are two control policies for the batch processor, and six control policies for the feeder processor. We have two broad

experimental set ups that are quite similar to what was done in Section 5.6.1. Table 6.5 summarizes the settings for which the experiments performed.

TABLE 6.5: EXPERIMENTAL SETUP SUMMARY FOR TWO-STAGE SYSTEM

Purpose of experiment	No. of two-stage control policies	Buffer Sizes	Batch Processor Sizes	Traffic intensities for feeder processor	Traffic intensities for batch processor
Evaluate effect of increasing batch processor size	12	6	3, 4, 5	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0
Evaluate effect of increasing buffer sizes	12	4, 5, 6, 7	3	0.2, 0.4, 0.6, 0.8, 1.0	0.2, 0.4, 0.6, 0.8, 1.0

A smaller number of different buffer sizes and batch processor sizes were used, compared to the experimental setup used when the feeder processor is a serial processor (Table 5.5). This is to avoid unwanted situations that could happen when the buffer sizes are low, such as deadlock. These special scenarios are discussed under each two-stage policy.

For each experiment, the mean processing rate of the feeder processor is held constant at one batch per unit time, and the nominal traffic intensities are used to determine the mean arrival rate of both job families, as well as the mean processing time at the batch processor. This implies that the maximum throughput rate for the two-stage system is two jobs per unit time. For all experiments, both job families have equal arrival rates. Let $1/\lambda$ be the mean arrival rate for either job family, μ_f be the mean time to finish processing a batch in the feeder processor and μ_B be the mean time to finish processing a batch in the batch processor. Then,

μ_f = 1 time unit / batch

$1/\lambda$ = traffic intensity TI of feeder processor × feeder processor size / 2

μ_B = (batch processor TI × batch processor size) / (feeder processor TI × feeder processor size)

The process of generating the state space, creating the probability rate transition matrix, and solving for the steady state distribution is programmed in MATLAB. The primary performance parameters considered are throughput and cycle time.

6.4.2 Behavior of System when Batch Processor is under No-Idling Policy

We divide the discussion, first according to the policy at the batch processor, then according to the policy at the feeder processor.

6.4.2.1 Feeder Processor is under No-Idling Policy

When both processors are under no-idling policy, neither processor incurs idle time waiting for larger batches to be produced. Consequently, the system behaves similarly to a two-stage system with an upstream serial processor and a downstream batch processor under no-idling policy (discussed in Section 5.6.4). Figure 6.4 illustrates the behavior of three different system configurations. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, while also increasing its processing time such that the maximum processing rate is constant.

Cycle time is non-decreasing with respect to throughput rate. As the buffer capacity is increased, both throughput and cycle time increases. As the batch processor's size is increased (with a corresponding increase in processing time), the throughput decreases and the cycle time increases. At very low arrival rates, the buffers are frequently empty, and the capacity of the buffers do not have a large effect on the system performance.

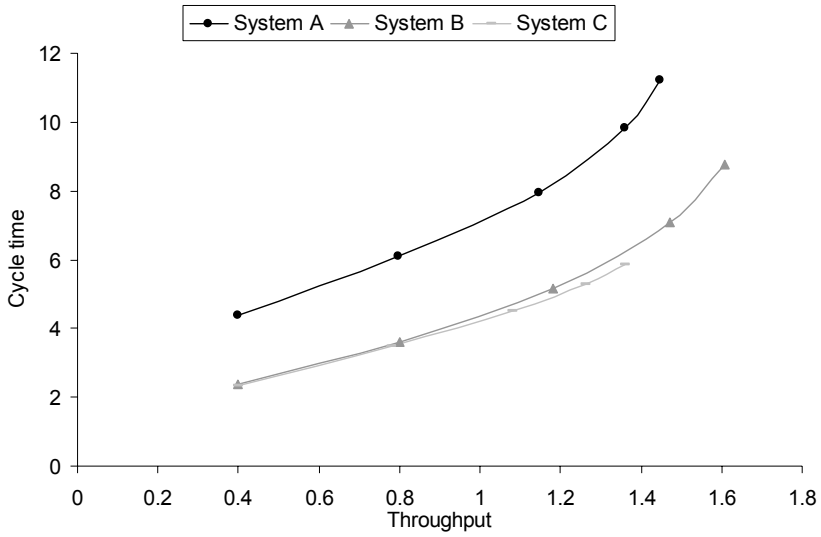


Figure 6.4 : The plot of cycle time versus throughput for a two-stage system with two batch processors.

The feeder processor has size two, and operates under (myopic-no-idling) policy. The downstream batch processor is under no-idling policy. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, while also increasing its processing time such that the maximum processing rate is constant. For all three system configurations, as the arrival rate is increased, the throughput and the average cycle time also increases.

6.4.2.2 Feeder Processor is under Full-Batch Policy

When the feeder processor is under full-batch policy and the batch processor is under no-idling policy, there exist situations where the Markov Chain model of the manufacturing system has transient states. The steady-state probability of the Markov chain residing in any transient state is zero. When the buffers in front of the batch processor are too small to hold at least two full batches of the feeder processor and the Batch Processor size Q_2 > Feeder Processor size Q_1, the only recurrent states are the states whose buffers in front of the batch processor have values 0 or Q_1.

Theorem 1: Let B_{2j} be a batch processor buffer for Job Family j. Let the capacity of B_{2j} be C_{2j}, and let the number of jobs inside B_{2j} be b_{2j}. If C_{2j} < the feeder processor size Q_1 and if the batch processor size $Q_2 > Q_1$, then the steady-state probability of any state where $0 < b_{2j} < Q_1$ is zero, when the feeder processor is under no-idling policy and the batch processor is under full-batch policy.

Proof: Assume that the buffer capacity is $Q_1 + \delta$, where $Q_1 > \delta \geq 0$. When the initial buffer level $b_{2j}(0) > Q_1$, the only possible change in the buffer level is from $b_{2j}(0)$ to $max\ (0,\ b_{2j}(0) - Q_2)$, since the feeder processor is blocked. If $max\ (0,\ b_{2j}(0) - Q_2) = 0$, then the only possible change in the buffer level is from 0 to Q_1, and the only possible change from Q_1 is from Q_1 to 0. If $0 < max\ (0,\ b_{2j}(0) - Q_2) < Q_1$, then the only possible changes in the buffer level are from $max\ (0,\ b_{2j}(0) - Q_2)$ to 0 or from $max\ (0,\ b_{2j}(0) - Q_2)$ to $Q_1 + max\ (0,\ b_{2j}(0) - Q_2)$. Since $Q_2 > Q_1$, $max\ (0,\ b_{2j}(0) - Q_2) < b_{2j}(0)$. Repeating this analysis with $b_{2j}(1) = max\ (0,\ b_{2j}(0) - Q_2)$, we eventually wind up with $b_{2j}(n) = max\ (0,\ b_{2j}(n-1) - Q_2) = 0$, in which case the only possible change in the buffer level is from 0

to Q_1. If the initial buffer level $b_{2j}(0) < Q_1$, the only possible changes in buffer level are from $b_{2j}(0)$ to 0, or from $b_{2j}(0)$ to $b_{2j}(0) + Q_1$, if $b_{2j}(0) + Q_1 < Q_1 + \delta$, in which case the analysis for when $b_{2j}(0) > Q_1$ is repeated. □

Theorem one identifies parameter settings that would allow us to artificially reduce the state space, since certain states would be transient. This reduces the storage space and computational time requirements in obtaining the steady state probability distributions.

There is another set of conditions that could cause the Markov chain to have transient states. When Q_1 and Q_2 are not relatively prime, there exist transient states, corresponding to the states where the contents of the buffers in front of the batch processor have quantities not relatively prime with Q_1 and Q_2.

Theorem 2: Let B_{2j} be a batch processor buffer for Job Family j. Let the capacity of B_{2j} be C_{2j}, and let the number of jobs inside B_{2j} be b_{2j}. If Q_1 can be expressed as $n \times \delta$ and Q_2 can be expressed as $m \times \delta$, where m and n are positive integers and δ is an integer greater than 1, then the steady-state probability of any state where b_{2j} cannot be expressed as $d \times \delta$, where d is a nonnegative integer, is zero, when the feeder processor is under no-idling policy and the batch processor is under full-batch policy.

Proof: Let $Q_1 = n \times \delta$ and $Q_2 = m \times \delta$, where $m > n$ and m, n and δ are positive integers and $\delta > 1$. When the initial buffer level is $b_{2j}(0) = p \times \delta$, p positive integer, the feeder processor finishing a batch will result in a new buffer level $b_{2j}(1) = (p + n) \times \delta$. If the batch processor starts a new batch, the new buffer level is $b_{2j}(1) = max((p - m) \times \delta, 0)$. Thus, if the initial buffer level is not relatively prime with respect to Q_1 and Q_2, then the buffer level will never be at a value that is not a multiple of δ, or 0.

If the initial buffer level is $b_{2j}(0) = p \times \delta + \psi$, ψ from 1 to $\delta - 1$, there exists a sequence of events such that the buffer level will be $b_{2j}(n) = q \times \delta + \psi < Q_2$. (The most straightforward sequence of events is when the batch processor keeps on processing full batches without the feeder processor finishing a batch.) From there, the buffer level can either go back to a value that falls under our assumption $b_{2j}(n+1) = (q + n) \times \delta + \psi$ (if the feeder processor finishes), or it can go to 0 (when the batch processor

loads a batch). Once the buffer level reaches *0*, any future transitions would limit the buffer level to values that are multiples of δ, or *0*. □

The proof does not rely on the feeder processor size being smaller than the batch processor size. Aside from reducing the state space, this result also means that it is possible to reduce the batch processor buffer capacity without affecting the long-term performance of the system. When the batch processor buffer capacity is $\chi = p \times \delta + \psi$, ψ from *1* to δ - *1* and $\delta > 1$, then reducing the buffer capacity to $\chi = p \times \delta$ will not affect system performance.

Figure 6.5 plots the mean cycle time versus throughput rate of three system configurations. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, with its processing rate decreased so that the maximum processing rates of all three systems are identical.

As seen in Section 5.6.4, forcing a batch processor to idle when buffers are nonempty results in long cycle times at low traffic intensities. For Systems B and C, the batch processor size (three) is not much greater than the feeder processor size (two). As the maximum processing rates of both processors are identical, the processing time of the batch processor is slightly longer than that of the feeder processor. When the arrival rates are very low, the waiting time to form a full batch at the feeder processor is large. As the arrival rates are increased, this waiting time is reduced, but the waiting time experienced by jobs in front of the batch processor increases, as the batch processor's utilization increases with increasing arrival rate. The net effect on the mean cycle time is a reduction of the mean cycle time, up to a certain arrival rate.

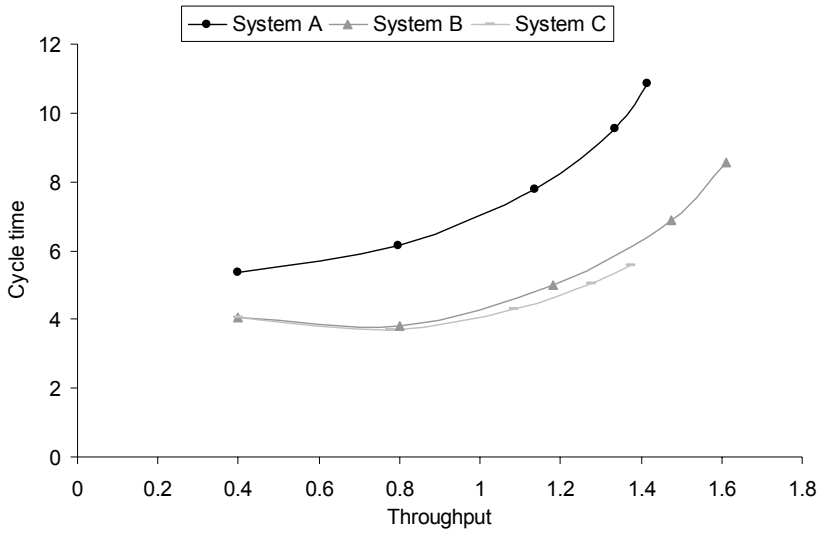


Figure 6.5: The plot of cycle time versus throughput for a two-stage batch-batch system.

The feeder processor has size two, and operates under (myopic-full-batch) policy. The downstream batch processor is under no-idling policy. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, while also increasing its processing time such that the maximum processing rate is constant. When the downstream batch processor size is not much larger than that of the feeder processor, the performance of the system assumes a convex form.

System A has the batch processor with three times the size of the feeder processor (six versus two). The processing time at the batch processor is then also three times that of the feeder processor. When the arrival rates are increased, the utilization of the batch processor increases (since it is under no-idling policy), and consequently jobs incur a large amount of waiting time in front of the batch processor. This is because the probability of the job reaching the batch processor while it is busy goes up, and the expected waiting time for the batch processor to become idle also goes up as the batch processor processing time goes up. This shifts the arrival rate with the minimum cycle time downward. In Figure 6.5, the plot for the performance of System A does not suggest a cycle time minimum although subsequent simulation experiments on the same system (shown in Figure 6.7) show that the cycle time versus throughput plot is still parabolic, with the minimum cycle time occurring in the vicinity of 0.2 arrival rate.

Figure 6.6 compares the performance of the system under different feeder processor policies (no-idling or full-batch) when the batch processor is under no-idling policy. When the batch

processor size is not much larger than that of the feeder processor (Systems A2 and B2), we obtain a pair of plots that are quite similar to a two-stage system with an upstream serial processor, with the downstream batch processor under either no-idling or full-batch policy. When the arrival rate is low, lower cycle time is achieved when the feeder processor is under no-idling policy. When the arrival rate is high, lower cycle time is achieved when the feeder processor is under full-batch policy.

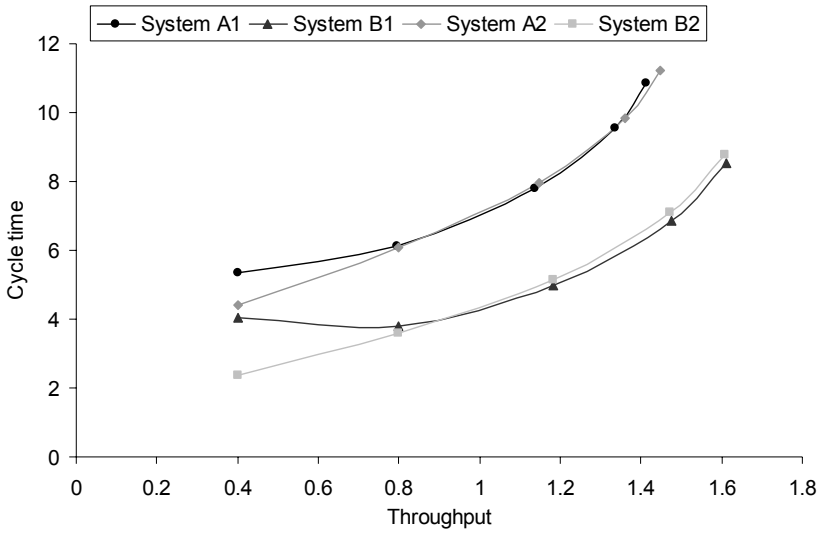


Figure 6.6: Performance plot for four different system configurations.

All four configurations have the buffer sizes fixed at seven, the feeder processor size fixed at two, and the batch processor under no-idling policy. Systems A1 and B1 have the feeder processor under (myopic-full-batch) policy, while Systems A2 and B2 have the feeder processor under (myopic-no-idling) policy. The batch processor size is at six for A1 and A2, and three for B1 and B2.

We look closer into the system performance when the batch processor size (six) is much larger than that (two) of the feeder processor (Systems A1 and B1). We use the simulation models built to validate the Markov models to obtain more detailed performance plots for Systems A1 and B1, which are shown in Figure 6.7.

Figure 6.7 suggests that when the arrival rates are low, using no-idling policy at the feeder processor reduces the mean cycle time. As the arrival rates increase to a moderate level, switching to the full-batch policy at the feeder processor will result in reduced mean cycle time. When the arrival rates are high, switching back to the no-idling policy at the feeder processor will result in significantly higher throughput. Interpolation of the points where experiments

were performed suggests that the no-idling policy also results in lower mean cycle time at high arrival rates.

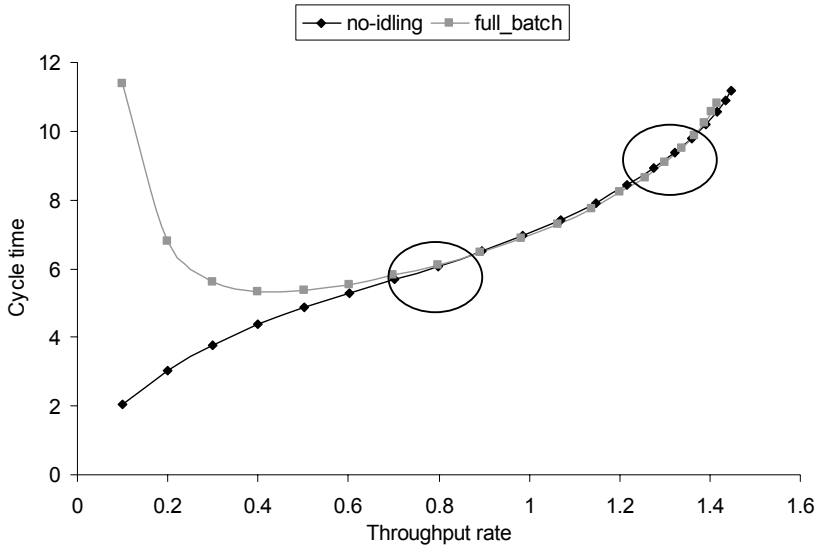


Figure 6.7: Detailed plot of systems A1 and A2 in Figure 6.6, obtained through simulation.

The plots for A1 and A2 intersect twice. This suggests that using the no-idling policy at the feeder processor results in lower cycle times when the arrival rates are low or when the arrival rates are very high. For this example, the first intersection occurs around arrival rate of 0.4, and the second intersection occurs at an arrival rate of approximately 0.85.

When the feeder processor is under full-batch policy, forcing jobs to wait at the feeder processor can cause starvation at the downstream batch processor. Furthermore, using the full-batch policy at the feeder processor can ironically cause smaller batches to be formed at the batch processor. For example, when there are four jobs in front of the batch processor and only one job at the feeder processor, the feeder processor will opt to wait for another job arrival before processing. Consequently, the batch quantity at the batch processor will be four, instead of five. While the batch processor is busy, the two jobs being processed by the feeder processor arrive in front of the batch processor, increasing the probability that the feeder processor will eventually become blocked, due to WIP build-up at the batch processor.

Using the full-batch policy at the feeder processor also reduces the effectiveness of buffers in between the feeder and batch processor, as the feeder processor becomes blocked when there is not enough space to hold a full batch (of the feeder processor) at the buffer, when the feeder

processor is under full-batch policy. On the other hand, when the feeder processor is under no-idling policy, the feeder processor only becomes blocked if the batch processor buffers are full. This maximizes the use of the batch processor buffer, and increases the throughput of the system.

6.4.3 Behavior of System when Batch Processor is under Full-Batch Policy

In this section we characterize the behavior of the system when the batch processor is under full-batch policy.

6.4.3.1 Feeder Processor is under No-Idling Policy

When the feeder processor is under no-idling policy and the batch processor is under full-batch policy, the gross system performance characterization is similar to that of a two-stage system with an upstream serial processor and a downstream batch processor under full-batch policy.

Figure 6.8 shows the performance plot of three system configurations. In System C, the downstream batch processor has capacity three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor capacity is increased to six, with its processing rate decreased so that the maximum processing rates of all three systems are identical. The cycle time versus throughput plot assumes a parabolic form for each of the three system configurations, due to the waiting time incurred by jobs in front of the batch processor, while waiting for full batches to be formed. The effect is more pronounced when the downstream batch processor size is much larger than that of the feeder processor.

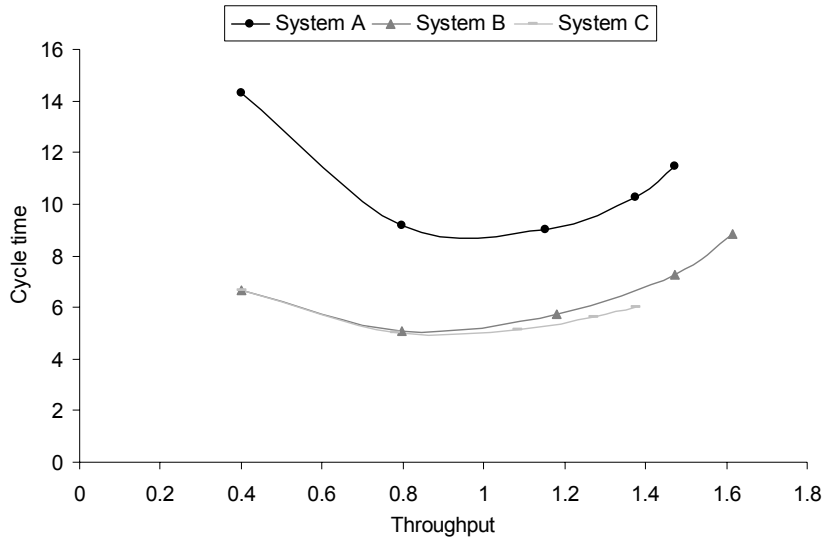


Figure 6.8: Cycle time versus throughput for a two-stage batch-batch system.

The feeder processor has size two, and operates under (myopic-no-idling) policy. The downstream batch processor is under full-batch policy. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, while also increasing its processing time such that the maximum processing rate is constant. The cycle time versus throughput plot assumes a parabolic form for each of the three system configurations, due to the waiting time incurred by jobs in front of the batch processor.

6.4.3.2 Feeder Processor is under Full-Batch Policy

When both processors are under full-batch policy, system deadlock can occur if $Q_1 + Q_2 - 1 >$ the smallest of the batch processor buffer capacities ([Chang and Gershwin 2005]). [Chang and Gershwin 2005] also proved that the system is not ergodic if the batch processor sizes are not relatively prime to each other. As a standard procedure, we assume that the initial state for all systems discussed in this chapter starts with the system empty.

Figure 6.9 shows the performance plot of three system configurations. In System C, the downstream batch processor has capacity three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor capacity is increased to six, with its processing rate decreased so that the maximum processing rates of all three systems are identical. The figure looks similar to Figure 6.8, although the parabolic curve is less prominent in Figure 6.8, as there is a greater tendency for a processor to wait in this case. (By the parabolic curve being less prominent, we mean that if

the curves were approximated by parabolas, the approximation of the less prominent curve would have greater distance between its focus and the directrix.)

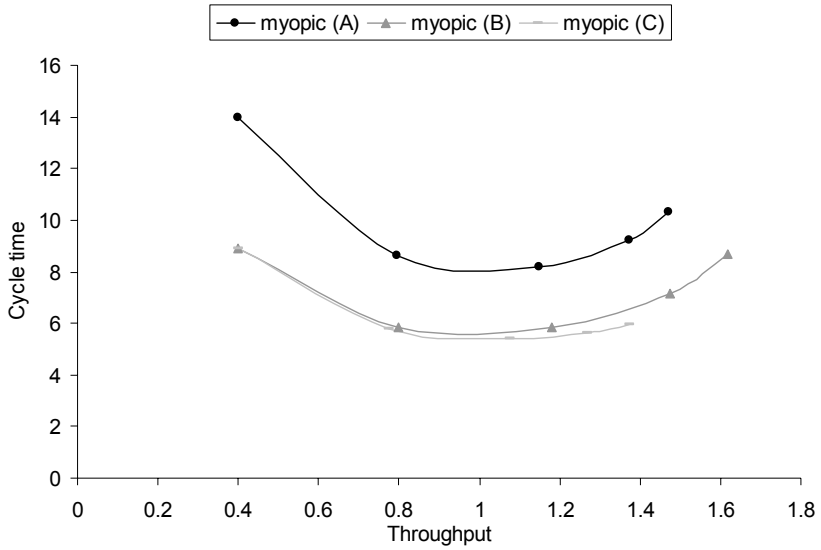


Figure 6.9: Cycle time versus throughput for a two-stage batch-batch system.

The feeder processor has size two, and operates under (myopic-full-batch) policy. The downstream batch processor is under full-batch policy. In System C, the downstream batch processor has size three and the maximum buffer sizes are all equal to four. In System B, the maximum buffer sizes are all increased to seven. In System A, the batch processor size is increased to six, while also increasing its processing time such that the maximum processing rate is constant.

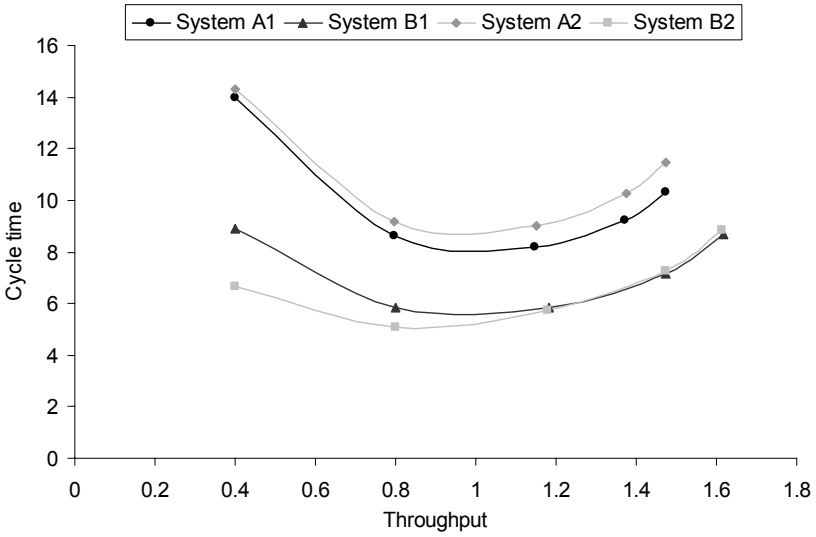


Figure 6.10: Performance plot for four different system configurations.

All four configurations have the buffer sizes fixed at seven, the feeder processor size fixed at two, and the batch processor under full-batch policy. Systems A1 and B1 have the feeder processor under (myopic-full-batch) policy, while Systems A2 and B2 have the feeder processor under (myopic-no-idling) policy. The batch processor size is at six for A1 and A2, and three for B1 and B2.

Figure 6.10 compares the performance of the system under different feeder processor policies (no-idling or full-batch) when the batch processor is under full-batch policy. When the batch processor size is not an integer multiple of the size of the feeder processor (Systems A2 and B2), the optimal policy at the feeder processor is dependent on the arrival rate. When the batch processor size is an integer multiple of the size of the feeder processor (Systems A1 and B1), the full-batch policy at the feeder processor provides better performance at any traffic intensity than the no-idling policy at the feeder processor. When a full batch is unavailable at the batch processor, it will require an integer number of full batches from the feeder processor before it can start processing. When the feeder processor is under no-idling policy, it would take at least as long a time interval for the feeder processor to process the required number of jobs, as it would take if the feeder processor is under full-batch policy.

6.4.4 Sensitivity of Model to Different Feeder Processor Policies

We discuss the relative merits of having the feeder processor under no-idling or under full-batch policy in Sections 6.4.2 and 6.4.3. Thus, in this section, we only examine the sensitivity of the model to the portion of the serial processor processing policy that deals with selection of the particular job family to be processed. We divide the discussion into four cases, corresponding to each possible combination of no-idling or full-batch policy for the feeder and the batch processor.

Figure 6.11 compares the performance of the system under two different feeder processor policies, when only the batch processor is under no-idling policy. Figure 6.12 does the same, except that both processors are under no-idling policy. In both figures, the feeder processor has size two, the batch processor has size three and under no-idling policy, and the buffer capacities at seven. Similar to when the feeder processor is a serial processor, the greedy look-ahead policy curve is below that of the no-idling policy curve for both figures, with the maximum throughput rate slightly higher for the myopic control policy.

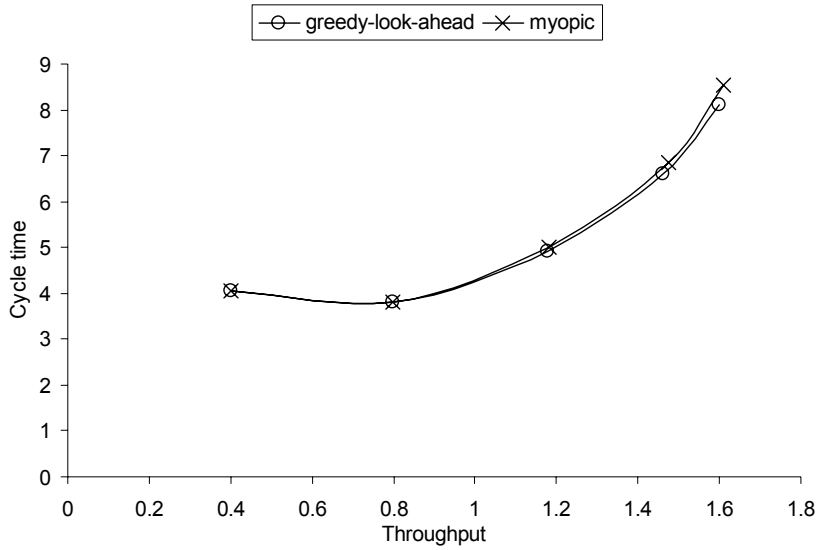


Figure 6.11: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three, and buffer capacities are seven.

Only the batch processor is under no-idling policy. For either system configuration, the curve for the greedy look-ahead policy is below that of its myopic policy counterpart.

When the batch processor size is small, the percentage reduction in both throughput and cycle time for the greedy look-ahead policy over the myopic policy is generally smaller when the feeder processor is under full-batch policy (Configuration A) than when the feeder processor is under no-idling policy (Configuration B). As the batch processor size is increased, the relationship reverses. (When the batch processor size is doubled from three to six, the greedy look-ahead policy can experience greater throughput rate than the myopic policy at high traffic intensity.) As the batch processor size is increased, it becomes more beneficial to have the feeder processor deliver more jobs (when the feeder processor is under full-batch policy). Unfortunately, having the feeder processor under a full-batch policy also encourages the build-up of WIP in front of the feeder processor, which increases the probability that incoming jobs see a full buffer in front of the feeder processor, and consequently get rejected by the system.

Table 6.6 contains the percentage reduction in throughput and cycle time of the greedy look-ahead policy over its counterpart myopic policy for a number of different parameter settings. Thus, we will find the percentage reduction of throughput and cycle time of the (greedy-look-ahead-full-batch) policy over the (myopic-full-batch) policy under Configuration A, and the

percentage reduction of throughput and cycle time of the (greedy-look-ahead-no-idling) policy over the (myopic-no-idling) policy under Configuration B.

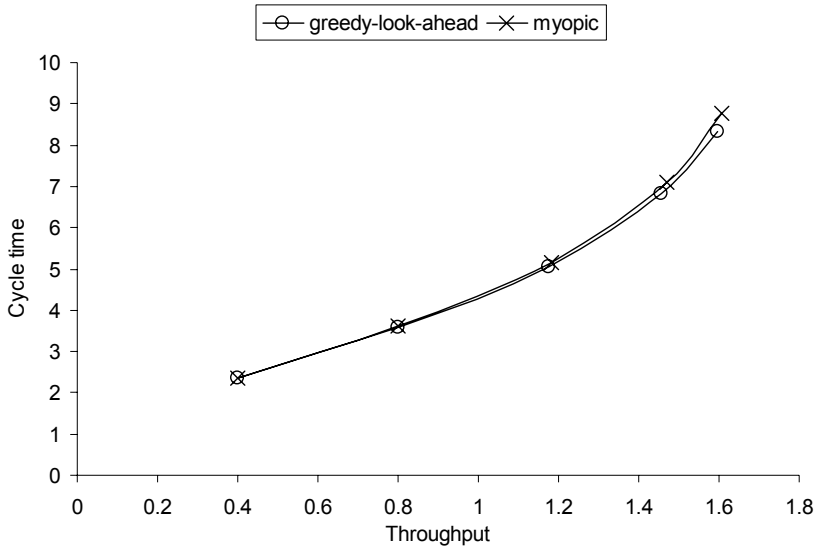


Figure 6.12: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three, and buffer capacities are seven.

Both processors are under no-idling policy. The curve for the greedy look-ahead policy is below that of its myopic policy counterpart, indicating that the greedy look-ahead policy can cause jobs to have reduced cycle time.

TABLE 6.6: RELATIVE PERFORMANCE OF GREEDY LOOK-AHEAD POLICY OVER MYOPIC POLICY FOR BALANCED TWO-STAGE SYSTEM WITH TWO BATCH PROCESSORS.

Parameters Varied			Throughput Reduction (%)		Cycle Time Reduction (%)	
Policy	Size	Traffic Intensity	Config. A	Config. B	Config. A	Config. B
No-idling	3	0.2	0	0	0.01%	0.14%
No-idling	3	0.4	0.04%	0.05%	0.23%	0.51%
No-idling	3	0.6	0.39%	0.47%	1.54%	1.81%
No-idling	3	0.8	0.92%	1.00%	3.88%	4.00%
No-idling	3	1.0	0.62%	0.85%	5.12%	5.06%
No-idling	6	0.2	0	0	0.04%	0.24%
No-idling	6	0.4	0.05%	0.05%	0.54%	0.82%
No-idling	6	0.6	0.26%	0.34%	2.31%	2.28%
No-idling	6	0.8	0.08%	0.35%	4.57%	4.09%
No-idling	6	1.0	-0.57%	-0.12%	5.86%	5.04%
Full-batch	3	0.2	0	0	0.03%	0.38%
Full-batch	3	0.4	0.03%	0.04%	0.37%	1.32%
Full-batch	3	0.6	0.30%	0.36%	1.71%	2.77%
Full-batch	3	0.8	0.64%	0.77%	4.16%	4.88%
Full-batch	3	1.0	0.40%	0.58%	5.71%	5.99%
Full-batch	6	0.2	0	0	0.04%	0.22%
Full-batch	6	0.4	0.03%	0.05%	0.66%	1.02%
Full-batch	6	0.6	0.18%	0.20%	3.02%	2.56%
Full-batch	6	0.8	-0.08%	0.05%	6.37%	4.50%
Full-batch	6	1.0	-0.86%	-0.41%	8.54%	6.04%

Figure 6.13 compares the system performance under two different policies when both processors are under full-batch policy, while Figure 6.14 compares the system performance under two different policies when only the batch processor is under full-batch policy.

The observations are consistent with what is seen when the batch processor is under no-idling policy. Furthermore, as seen in Chapter 5 where the feeder processor is a serial processor, the benefit of using the greedy look-ahead policy is larger when the batch processor is under full-batch policy. This difference is magnified when the batch processor size is large.

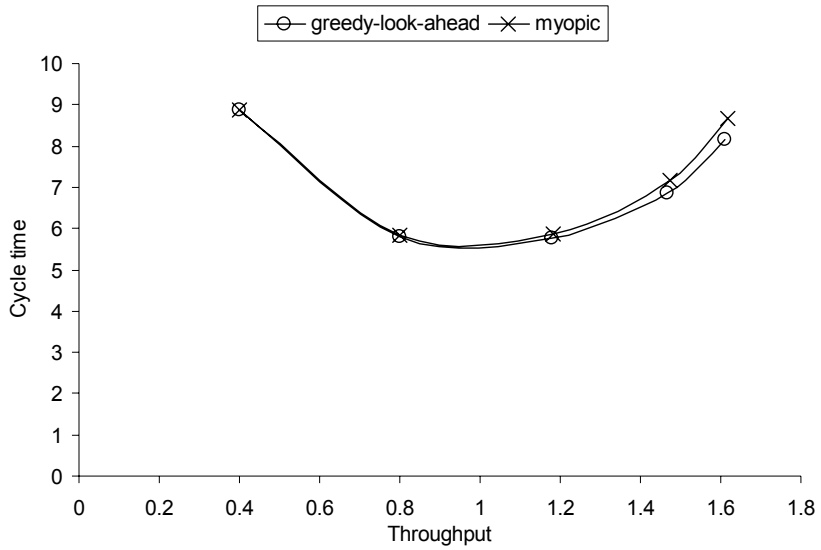


Figure 6.13: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three and buffer capacities are seven.

Both processors are under full-batch policy. The curve for the greedy look-ahead policy is below that of its myopic policy counterpart, indicating that the greedy look-ahead policy can have the same throughput rate, but with reduced mean cycle time than the myopic policy.

When the feeder processor is a serial processor instead of being a batch processor, the relative reduction in cycle time due to the greedy look-ahead policy is larger. As the feeder processor's size is increased relative to the batch processor's size, the relative importance of anticipating the needs of the batch processor diminishes. Furthermore, the number of instances for which the decisions made by the greedy look-ahead policy and the myopic policy potentially differ is reduced when the feeder processor size is increased. When the feeder processor is a serial

processor, the two policies may result in different decisions if there is at least one job from each family in front of the feeder processor, and there is at least one free slot for each job family buffer in front of the batch processor. When the feeder processor size is increased, these conditions become more stringent. For example, when the feeder processor is under full-batch policy, with size Q_1, then the two policies will only have different decisions if each job family has at least Q_1 jobs in front of the feeder processor and at least Q_1 free slots in front of the batch processor.

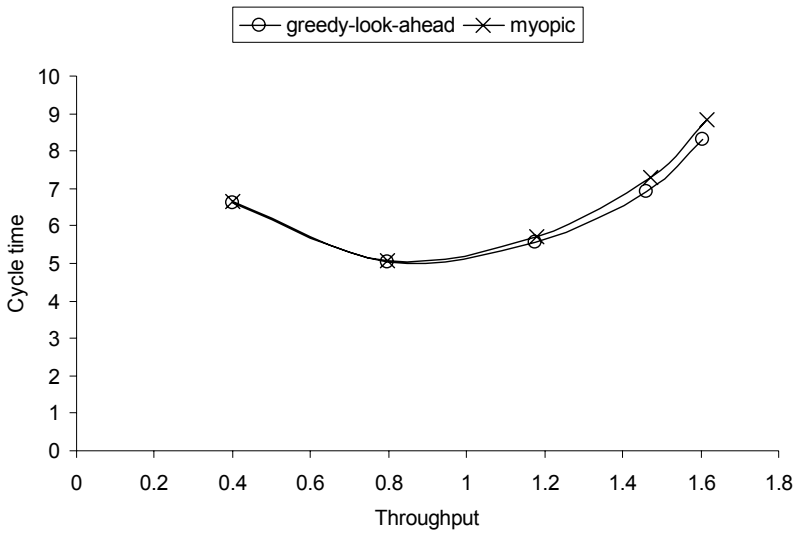


Figure 6.14: Performance of a balanced two-stage system with feeder processor size = two, batch processor size = three and buffer capacities = seven.

Only the batch processor is under full-batch policy. The curve for the greedy look-ahead policy is below that of its myopic policy counterpart. However, the myopic policy can generate a slightly higher maximum throughput rate, given a particular buffer size limit.

6.5 Behavior of Three-Stage System

In this section we discuss the experimental setup used, and analyze the behavior of the three-stage manufacturing system. Similar to Section 5.7, we are primarily interested in determining whether the strategy of controlling the production of additional processors further upstream of the batch processor as a function of the anticipated needs of the batch processor can result in a larger amount of cycle time reduction.

6.5.1 Experimental Design

We set all buffer sizes to four, the feeder processor size to two, and the batch processor size to three. We only look into the case where all processors have identical maximum throughput rates. The mean processing time of the serial processor is held constant at one job / time unit, and the other parameters are derived from:

λ = traffic intensity *TI* of serial processor / 2

μ_f = *TI* of feeder processor / (*TI* of feeder processor × feeder processor size)

μ_B = *TI* of batch processor / (*TI* of batch processor × batch processor size)

Because the feeder processor is also a batch processor, the decision to process can be split into two: the decision to form a batch, and the decision to process which job family. This results in a relatively large number of possible three-stage control policies. We select 16 three-stage control policies. Table 6.7 contains the combination of control policies evaluated.

TABLE 6.7: EXPERIMENTAL SETUP FOR THREE-STAGE SYSTEM

Purpose of experiment	Batch processor control policies	Serial processor - feeder processor control policies considered	Job arrival rates
Compare performance measures of system under different control policies	No-idling Full-batch	Myopic-(Myopic-no-idling) Myopic-(Myopic-full-batch) Myopic-(Greedy look-ahead-no-idling) Myopic-(Greedy look-ahead- full-batch) Greedy-look-ahead - (Greedy look-ahead-no-idling) Greedy-look-ahead - (Greedy look-ahead- full-batch) Balanced-look-ahead - (Greedy look-ahead-no-idling) Balanced-look-ahead - (Greedy look-ahead- full-batch)	0.2, 0.4, 0.6, 0.8, 1.0

6.5.2 Sensitivity of Model to Different Serial Processor Policies

Due to the small buffer sizes used in the experiment, the performance differences between the policies can be difficult to observe when plotted. Figure 6.15 to Figure 6.18 plot the throughput rate versus the mean cycle time for jobs passing through the system described in Section 6.5.1, under different policies for the first two processors. Figure 6.15 assumes both the feeder and the batch processor to be under full-batch policy, Figure 6.16 assumes only the batch processor is under full-batch policy, Figure 6.17 assumes only the feeder processor is

under full-batch policy, and Figure 6.18 assumes both the feeder and the batch processor is under no-idling policy.

The figures are arranged in descending order of expected variations between sets of policies. This expectation is based on the observations made for a smaller two-stage system, and visual inspection confirms that this is indeed the case. Furthermore, the relative ordering of the myopic-myopic, myopic-greedy-look-ahead and greedy-look-ahead-greedy-look-ahead policy combinations for the serial and feeder processors is identical to that seen when the feeder processor is also a serial processor (Section 5.7.2).

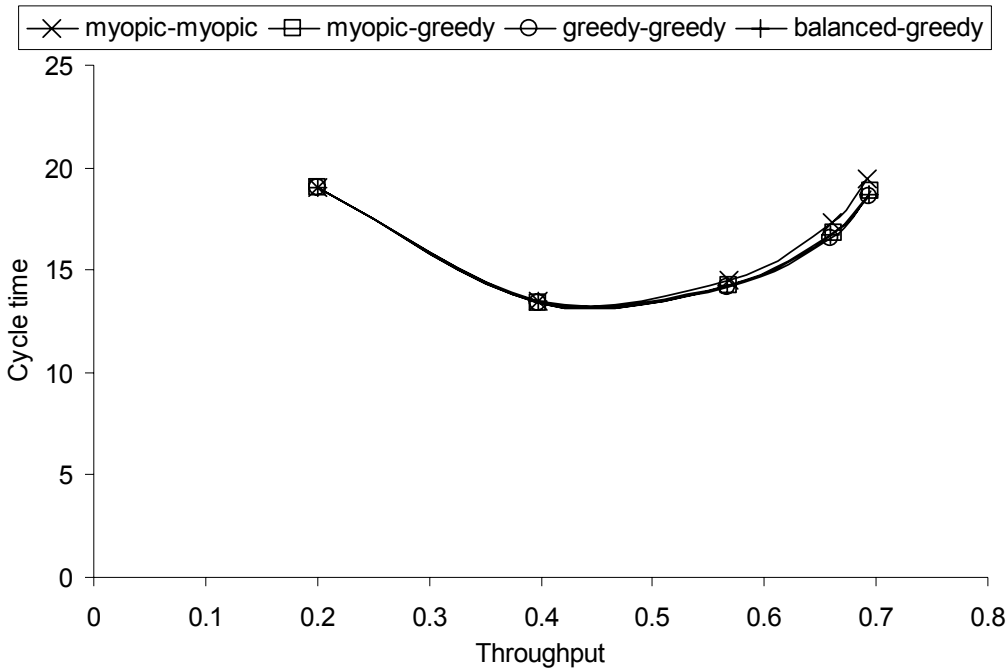


Figure 6.15: Performance of several policy combinations for a three stage system when both the feeder and batch processors are under full-batch policy.

The myopic-myopic policy combination performance plot is clearly above that of the other three policy combinations.

When the feeder processor's size is increased, the feeder processor's processing rate is reduced, and the mean time to process a batch increases. Similar to what is seen in Section 5.7.2, this causes the balanced-look-ahead-greedy-look-ahead policy combination to experience deteriorating performance. Thus, for the set of parameters evaluated, the performance plot of the balanced-look-ahead-greedy look-ahead policy is no longer below that of the greedy-

greedy look-ahead policy. When the feeder processor is under full-batch policy, the time it takes for the feeder processor to process a batch (including the time spent waiting for a full batch) is increased, and this causes the performance of the balanced-look-ahead-greedy look-ahead policy to further deteriorate.

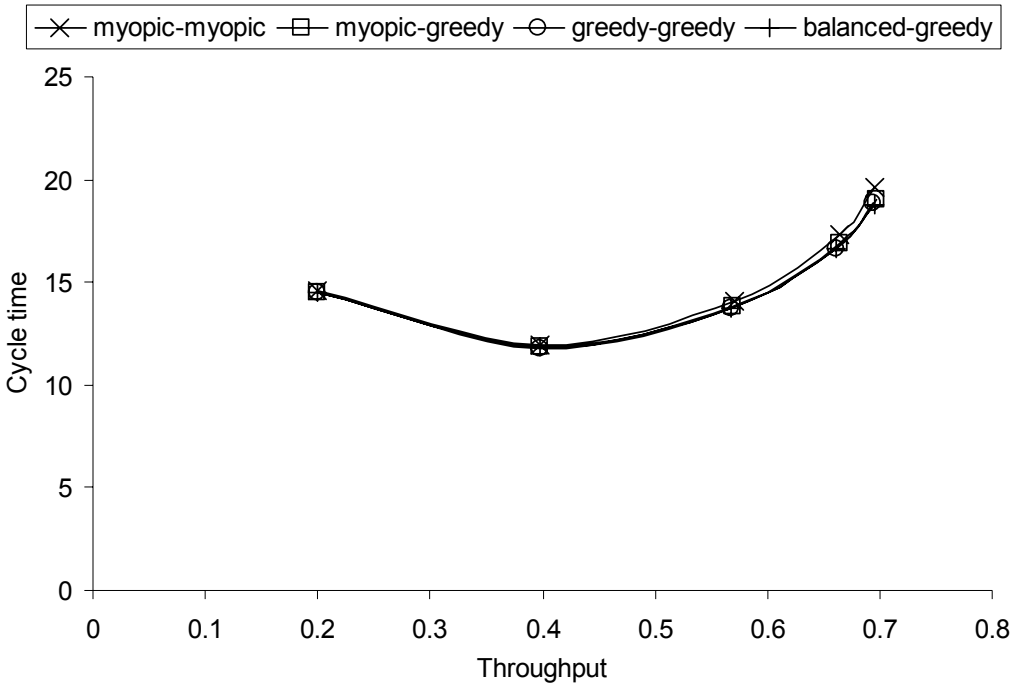


Figure 6.16: Performance of several policy combinations for a three stage system when only the feeder processor is under no-idling policy.

The myopic-myopic policy combination performance plot is still clearly above that of the other three policy combinations, albeit with a smaller gap between the myopic-myopic and the look-ahead policies.

Table 6.8 contains the percentage reduction from the base myopic-myopic policy combination for the other policy combinations evaluated. In System One, both feeder and batch processor are under full-batch policy. System Two has the only the batch processor under full-batch policy, System Three has only the feeder processor under full-batch policy. In System Four, neither processor is under the full-batch policy. The additional reduction in cycle time due to the greedy-look-ahead-greedy look-ahead policy combination, compared with the myopic-greedy-look-ahead policy combination, is indicative of the additional benefit we can hope to achieve if we constrain an additional processor further upstream.

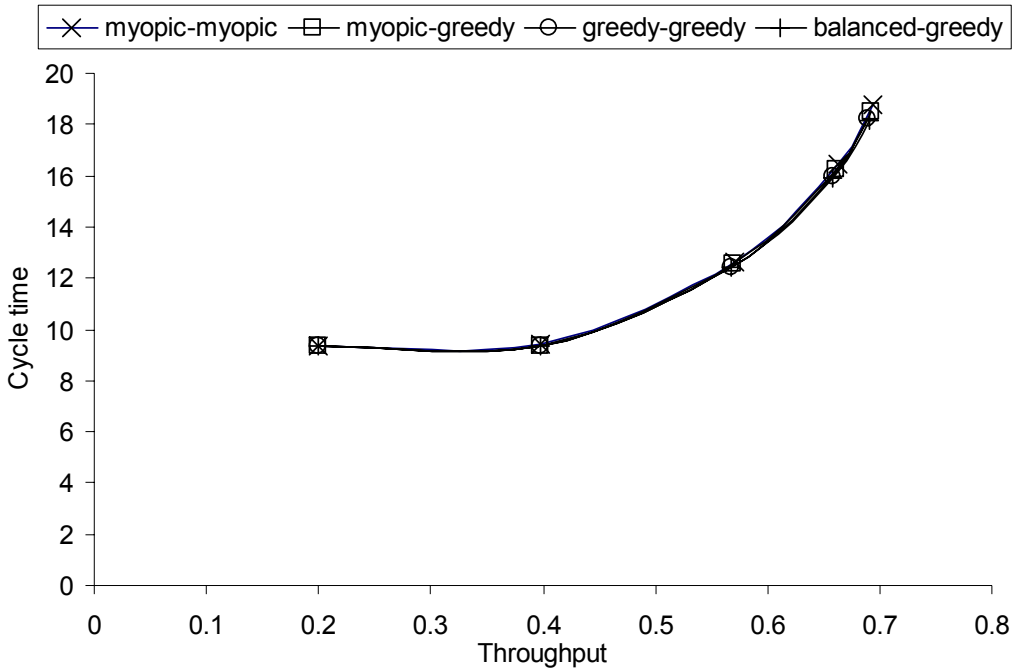


Figure 6.17: Performance of several policy combinations for a three stage system when only the feeder processor is under full-batch policy.

Several of the individual performance plots are difficult to discern from each other. However, the graph for the myopic-myopic policy is still noticeably above that of the other policies.

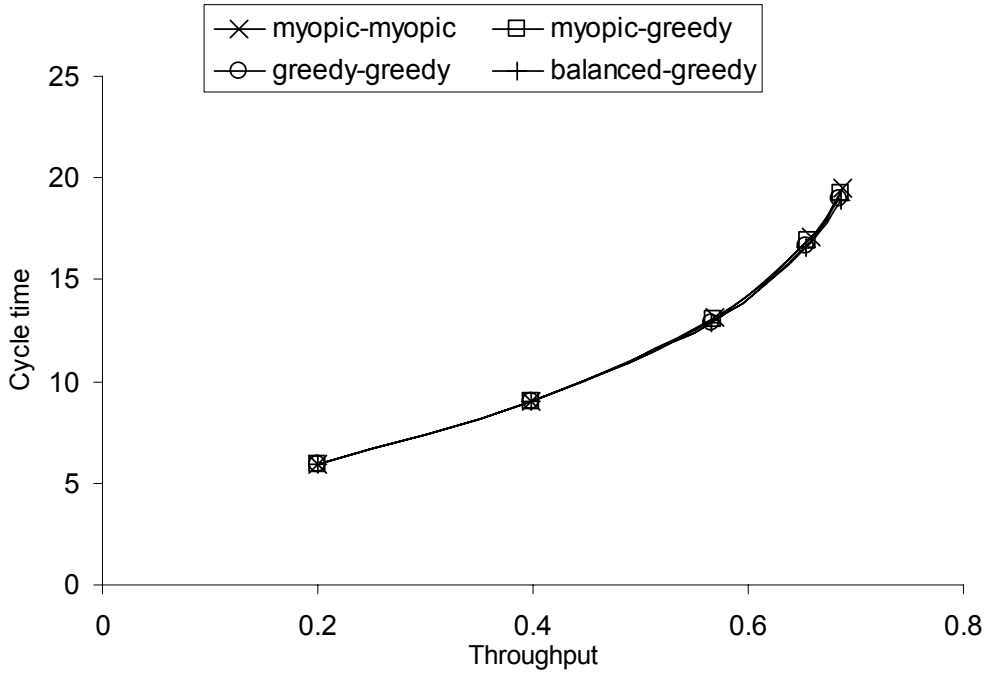


Figure 6.18: Performance of several policy combinations for a three stage system, when both the feeder and batch processor are under no-idling policy.

The individual performance plots are almost coincident to each other. Separation can barely be detected even at high traffic intensity.

When the feeder processor is a serial processor (Section 5.7.2), the incremental benefit of looking ahead an additional upstream processor is rapidly reduced. When the feeder processor is also a batch processor, the results suggest that the incremental cycle time reduction diminishes at a slower rate, for three-stage systems. This is because the first (serial) processor and the feeder processor will attempt to process jobs from the same family most of the time under the greedy-look-ahead-greedy-look-ahead policy combination. This implies that the serial processor is processing jobs in accordance to the anticipated needs of the feeder processor. Thus, the first two processors of the system act like the serial processor-batch processor two-stage system in Section 5.6. For this reason, the greedy-look-ahead-greedy look-ahead policy combination still manages to substantially improve on the initial cycle time reduction by the myopic-greedy-look-ahead policy combination without a large increase in the maximum throughput loss, in all four systems.

TABLE 6.8: PERFORMANCE OF THREE POLICY COMBINATIONS RELATIVE TO THAT OF MYOPIC-MYOPIC POLICY COMBINATION

System	Traffic intensity	Myopic-greedy look-ahead		Greedy-greedy look-ahead		Balanced-greedy look-ahead	
		Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction	Throughput reduction	Cycle time reduction
1	0.2	0	0.06%	0	0.08%	0	-0.06%
1	0.4	0.07%	0.34%	0.08%	0.73%	0.07%	-0.30%
1	0.6	0.33%	0.92%	0.36%	2.53%	0.37%	0.01%
1	0.8	0.43%	1.45%	0.24%	4.22%	0.48%	0.82%
1	1.0	0.34%	1.53%	-0.16%	4.68%	0.31%	1.15%
2	0.2	0	0.34%	0	0.39%	0	0.28%
2	0.4	0.02%	1.10%	0.09%	1.32%	0.07%	0.97%
2	0.6	0.11%	1.96%	0.43%	2.66%	0.43%	2.34%
2	0.8	0.02%	2.65%	0.46%	3.84%	0.42%	3.79%
2	1.0	-0.23%	2.80%	0.12%	4.06%	-0.06%	4.37%
3	0.2	0	0	0	0.01%	0	0
3	0.4	0.08%	0.16%	0.11%	0.26%	0.08%	0.14%
3	0.6	0.38%	0.79%	0.59%	1.50%	0.38%	0.99%
3	0.8	0.52%	1.36%	0.76%	2.96%	0.41%	1.84%
3	1.0	0.43%	1.43%	0.59%	2.85%	0.19%	1.95%
4	0.2	0	0.10%	0	0.14%	0	0.14%
4	0.4	0.03%	0.29%	0.10%	0.50%	0.10%	0.55%
4	0.6	0.28%	0.80%	0.61%	1.60%	0.60%	1.91%
4	0.8	0.41%	1.32%	0.94%	2.60%	0.73%	3.09%
4	1.0	0.26%	1.36%	0.59%	2.73%	0.35%	3.19%

6.6 Chapter Summary

The previous chapter looks at the performance of a small system with an upstream serial processor and a downstream batch processor, under several simple control policies. We determine that using a serial processor policy that processes jobs according to the anticipated needs of the batch processor has the potential to reduce the mean cycle time of jobs passing through the system. In this chapter, we extend the analysis to the case where the batch processor's feeder processor is also a batch processor with smaller size. When a processor's size is greater than one, two decisions have to be made: when to process a batch, and what job family should be processed. We consider two different feeder processor policies on determining when to process a batch, and two different batch processor policies, and characterize the performance of the system. We also determine some interesting properties of the system, particularly when the buffer sizes are small.

In determining which job family to process, we adapt the serial processor control policies used in Chapter 5, and evaluate their relative performances under the four different possible feeder processor-batch processor policy combinations (in determining when to process a batch). We determine that the conclusions derived when the feeder processor is a serial processor also hold true when the feeder processor is a batch processor of smaller size.

In extending the analysis to a three-stage system, experimental results suggest that the incremental cycle time reduction of "looking ahead" an additional upstream processor does not deteriorate significantly. This is primarily due to the feeder processor being also a batch processor; since the first processor and the feeder processor will attempt to process the same job families most of the time, the first two processors of the system perform like the two-stage system evaluated in Section 5.6.

Chapter 7 Discussion of Conclusions and Recommendations

Batch processors can process more than one job at a time; typically, the processing time of the batch processor is not explicitly dependent on the quantity of the batch being processed, but can be dependent on the composition of the batch. In semiconductor wafer fabrication, the oxidation and diffusion ovens (or furnaces) belong to a certain sub-class of batch processors, which we generalize as the wafer fab batch processor model. In the wafer fab batch processor model, jobs belong to a job family. Although the batch processor can process up to Q jobs at a time, all jobs that comprise the batch have to belong to the same job family. The processing time of a batch is dependent solely on the job family it is currently processing. We concentrate on the wafer fab batch processor model.

In Chapter 2, we classify the current literature on the control of systems involving batch processors, and determine that there is a gap in the literature between the control of a single batch processing stage and the control of multi-stage systems involving at least one batch processing stage. Majority of current literature assume that the batch processing stage exists in isolation, and treat the job arrivals as a constraint. Although there has been some work done on small multi-stage systems involving batch processors, few have considered the possibility of the system processing multiple job families. Furthermore, majority of the work done on large multi-stage systems (such as wafer fabs) either adopt a simplistic rule for the batch processor, or consider controlling the batch processor as an isolated problem. We believe that there is

significant potential in improving manufacturing system performance by constraining the control of processors upstream of the batch processor as a function of the batch processor. In particular, the hypothesis of controlling the upstream processors of the batch processor in accordance to the batch processor's anticipated needs is the subject of this investigation.

7.1 Summary of Conclusions

In this section, we summarize the work done, and discuss the significance of the conclusions obtained. The discussion is segmented into four sections: controlling a batch processing stage in isolation, controlling the quantity of jobs arriving into the batch processor buffer, controlling an upstream serial processor and controlling an upstream batch processor.

7.1.1 Control of the batch processor in isolation

We consider the problem of controlling a single batch processor that processes jobs that belong to different job families, when the jobs arrive at future time instances. When both the number of jobs to be processed and the arrival times of all jobs are known, the problem is NP-Hard. We propose both an integer linear programming and a Dynamic Programming (DP) algorithm that will minimize the mean cycle time. The proposed DP algorithm is not efficient, but runs in polynomial time if the number of job families is fixed, albeit with high polynomiality. Although we identify several conditions that can reduce the amount of time required to solve the problem, empirical testing of the algorithm show that only small problem instances can be solved within a reasonable amount of time.

For larger problem instances where the number of jobs to be processed may not be necessarily known, we propose a heuristic based on Model Predictive Control (MPC). We evaluate the performance of the MPC-based heuristic against that of a popular look-ahead method, NACHM. Statistical analysis of simulation results support the hypothesis that the MPC-based heuristic (with properly selected parameters) has significantly lower mean cycle time than

NACHM. This improvement still holds true even if NACHM has a longer horizon, and is due to the MPC-based heuristic optimizing the sequence of batches needed to process all jobs within its horizon. In contrast, NACHM only optimizes the cycle time incurred in processing the first batch

If the hypothesis that controlling the processing of the upstream processors can improve the performance of the batch processor is true, then increasing the correlation in the job families of future job arrivals should yield an improvement in the mean cycle time of jobs, regardless of the batch processor control policy. Statistical analysis of simulation results supports this hypothesis, and the predicted improvement is more significant when the number of job families is large. Furthermore, as the degree of correlation is increased, the relative performance of NACHM, compared to that of the MPC-based heuristic, improves to a point where NACHM can outperform the MPC-based heuristic for specific parameter settings. Lastly, the magnitude of improvement obtained from inducing positive correlation is generally larger than that obtained from switching from a simpler heuristic (NACHM) to a more complicated heuristic.

We have shown that there exists two different ways to reduce the mean cycle time of jobs passing through the batch processor. We can make better use of the information we have (MPC-based heuristic versus NACHM), at the cost of increased algorithmic complexity and computational time, or we can control the upstream processors such that the future arrivals to the batch processor have highly correlated job families. Furthermore, results suggest that inducing positive correlation can be more rewarding than using a complicated control policy for the batch processor. These generalizations also hold true when the batch processing stage contains more than one batch processor (Appendix A).

7.1.2 Control of job arrivals into the batch processor buffer

Another advantage of controlling the upstream processor with the batch processor in mind occurs when jobs have processing time windows. In wafer fabrication, jobs exiting the

upstream processor need to be processed by the batch processor before T time units (T is the processing time window) have elapsed since the job exited the upstream processor. Failing this, the job has to undergo reprocessing or validation before continuing. Since the upstream processors of the batch processor also supply other processors, jobs that reach their processing time windows also represent lost production for the other processors being supplied by the feeder processor.

We assume a two-stage system, where a serial processor feeds a buffer that supplies the batch processor with WIP. All jobs belong to a single job family. We initially look at three different models, with each model having more sources of system stochasticity than the previous model. In each of the three models, we assume that the serial processor is never starved and the batch processor is never blocked. Each job processed by the serial processor earns a reward R, and each job that exceeds its processing time window incurs a penalty C. At each instance that the serial processor can process a job, we determine the expected reward of processing this job, and process a job if and only if the expected reward of processing the job is positive. This policy maximizes the expected reward obtained, at each instance the processor is free.

In the first model, the serial processor has deterministic processing time of one time unit, and the batch processor processing time is geometrically distributed. We determine that the probability of incurring a penalty is strictly increasing with respect to the number of jobs in front of the batch processor. This means that the optimal policy is a threshold policy. Furthermore, the threshold value is determined to be an integer multiple of the batch processor size, and this threshold value is identical, regardless of whether the batch processor is under no-idling or full-batch policy.

In the second model, the serial processor processing time is also geometrically distributed. When the batch processor is under no-idling policy, the optimal policy is still a threshold policy. Unfortunately, this threshold value is no longer guaranteed to be an integer multiple of the batch processor size. This discrepancy occurs because the number of batches processed before the prospective job is uncertain; the batch it is supposed to join may get loaded into the batch processor before the prospective job exits the serial processor. Because of this possibility,

the threshold value can either be an integer multiple of the batch processor size, or one + an integer multiple of the batch processor size.

When the batch processor is under full-batch policy, the probability of incurring a penalty is no longer strictly increasing. This is because there are two independent events that can cause the prospective job to incur a penalty: (1) the batch processor cannot finish all the batches ahead of the prospective job in time, or (2) the serial processor cannot process enough jobs to form a full batch for the prospective job. While the probability of (1) is strictly increasing with respect to the number of jobs in front of the batch processor, the probability of (2) is periodic with respect to the number of jobs in front of the batch processor (with period equal to the batch processor size). However, we can approximate the probability of incurring a penalty with a step function where the steps occur at integer multiples of the batch processor size. Using this step function as the approximate probability function results in a threshold policy where the threshold is an integer multiple of the batch processor size.

For Model Three, we assume that the batch processor also suffers from operation dependent failures, where the mean time to fail and the mean time to repair are both geometrically distributed. For both batch processor policies, the probabilities of incurring a cost may differ according to the current state of the batch processor (up or down).

In Model Four, we assume that the serial processor can get starved of WIP, and look at the probability of incurring a penalty solely due to the lack of additional jobs to form a full batch. The analysis can then be coupled with the batch processor assumptions in Models Two and Three, when the batch processor is under full-batch policy. We determine that the probability of incurring a penalty is non-decreasing with smaller serial processor processing probabilities, IF there are enough jobs in front of the serial processor to guarantee the prospective job will be able to form a full batch without waiting for future job arrivals into the serial processor. When future job arrivals to the serial processor buffer are needed, this may no longer be true. This is because longer serial processor mean processing times (which corresponds to lower serial processor processing probabilities) increase the probability of additional jobs arriving in front of the serial processor while the serial processor is processing the prospective job.

Aside from developing the expressions for the probabilities of incurring a penalty, and characterizing the policies obtained from the obtained probabilities, we also provide the state space and the one-step transition equations for the Markov Chain equivalents of the two-stage systems in Models One to Three, when the serial processor is under a threshold policy. These equations can be used to determine the steady-state performance of the two-stage system, given the threshold adopted. We also discuss several ways of adopting the results into more complex systems. In particular, since the probability of incurring a penalty is primarily determined by the number of batches in front of the prospective job, it is natural to interpret the buffer size limit in terms of number of batches that can be formed, rather than number of jobs. Thus, when the batch processor is processing jobs that belong to one of many job families, to maximize the use of the buffer space, the serial processor would prefer to process jobs for which a partial batch at the batch processor buffer already exists. This ties in neatly with the concept of introducing positive correlation between consecutive job arrivals to reduce mean cycle time, in Chapter 3.

7.1.3 Performance of a serial - batch processor system under simple control policies

From Chapter 3, the mean cycle time for a system composed of a single batch processing stage is affected when the upstream processor (which were outside of the system being considered) processes jobs such that the job flow into the batch processor buffers are modified. We wish to determine whether this is also true when the system is enlarged to include the upstream processor and its buffers. We initially characterize the gross long-term performance of a two – stage system with a serial processor feeding a batch processor. The system experiences stochastic arrival and processing times, and processes jobs belonging to two job families. The batch processor is assumed to always process the job family with the highest number of jobs queued at the batch processor, and can either process partial batches (no-idling policy) or always wait for a full batch (full-batch policy).

The general behavior of the system mimics that of a two-stage system composed of only serial processors, in terms of its sensitivity to changes in buffer sizes and mean processing rate. The smaller the batch processor size, the closer the system performance is to that of a system with serial processors. Unfortunately, the possibility of the batch processor processing less than a full batch, or waiting for additional jobs to arrive means that the performance of a two-stage system with serial processors serves as a strict upper bound to the performance of a system with a downstream batch processor, with the bound becoming less tight as the batch processor becomes larger but slower.

When the batch processor is under no-idling policy, the throughput and the cycle time both increase monotonically with respect to arrival rate. However, the rate of increase in cycle time as arrival rates are increased is lower than that of a serial – serial system. This is due to the batch processor's ability to partially compensate for increased job arrivals by processing more jobs per batch.

When the batch processor is under-full batch policy, the throughput still increases monotonically with respect to arrival rate. However, the cycle time is no longer monotonically increasing with respect to arrival rate. When job arrivals are rare, jobs spend considerable time waiting in front of the batch processor for enough jobs to form a full batch. This causes the mean cycle time of jobs to be high when the job arrival rate is low, particularly when the batch processor sizes are large. As the job arrival frequency increases, the ease in forming full batches causes the mean cycle time of jobs to decrease, up to a certain point. Beyond this traffic arrival rate, system congestion causes the cycle time to start increasing with increasing job arrival frequency. This relationship between cycle time and throughput implies that it is sometimes possible to increase throughput and reduce cycle time for systems involving batch processors, if batch processors are forced to wait for future job arrivals.

When the arrival frequency is low, the system has better performance when it is under no-idling policy (System A), since the system incurs high cycle time if a full-batch policy is used (System B). As the arrival frequency increases, it becomes faster to form full batches under

System B, and System A becomes more and more congested, due to its partial use of the entire batch processor size.

To model the serial processor taking the batch processor's anticipated needs into consideration, we develop a naively constructed serial processor policy that prefers to process the job family that has the most WIP in front of the batch processor. This idea is based on the assumption that the batch processor prefers to locally maximize its throughput, which occurs when it processes jobs that have the longest queue at the batch processor. This greedy-look-ahead policy is compared with a myopic policy that processes the job with the longest queue at the serial processor; this myopic policy locally maximizes the throughput of the serial processor. When the traffic intensities are low, there is little differentiation between the performances of the different control policies, since the system rarely has any choice as to which job family to process. The smaller the batch processor size, the larger the traffic intensity has to be before there is significant differentiation in performance between the different policies. At higher traffic intensities, the mean cycle time for the myopic control policy becomes noticeably higher than the mean cycle time for the greedy-look-ahead policy.

We also consider what happens when the two processors do not have the same maximum processing rates. The faster the serial processor processing rate, the higher the job arrival rate has to be before the performance of the greedy look-ahead and the myopic policy can be easily distinguished. When the serial processor processing rate is high compared to the arrival rate, queues rarely form in front of the serial processor, and the serial processor rarely has a choice as to what job family is to be processed. As the job arrival rate increases, queues form in front of the serial processor, allowing the policies to diverge in the job families selected for processing. Since the greedy look-ahead policy ignores the serial processor buffer levels, the reduction in throughput rate also generally increases with increasing job arrival rate, up until the arrival rate equals the serial processor processing rate.

The concept of constraining additional processors further upstream to process jobs according to the batch processor's anticipated needs is also evaluated. Unfortunately, the throughput loss increases at a higher rate than the cycle time reduction, as we move from a myopic-greedy

look-ahead policy to a greedy-greedy look-ahead policy. Furthermore, the incremental cycle time reduction obtained in "looking ahead" an additional upstream processor shrinks. This implies that the incremental benefit of constraining upstream processors to process jobs the batch processor prefers diminishes, and this concept can be used for only a small number of upstream stages before performance deterioration occur.

7.1.4 Performance of a batch - batch processor system under simple control policies

In semiconductor wafer fabs, the ovens can also be supplied by another batch processor, albeit with smaller size. Similar to what is done in Chapter 5, we use a Markov chain model of a two-stage system processing two job families. The main objective is to determine whether controlling the feeder processor as a function of the batch processor's anticipated needs will also result in lower mean cycle times. Four cases are considered: when both processors are under no-idling policy, when both processors are under full-batch policy, when only the feeder processor is under full-batch policy, and when only the feeder processor is under no-idling policy.

When the feeder processor is under no-idling policy, the characterization of the two-stage system is similar to the behavior of a two-stage system with a serial processor feeding a batch processor. This is consistent with observations from Chapter 5 that a system where the batch processor operates on no-idling policy mimics a system where the batch processor is replaced by a serial processor.

When the feeder processor is under full-batch policy, the behavior of the Markov chain models of the two-stage system can exhibit special properties. [Chang and Gershwin 2005] proved the existence of deadlock conditions and the lack of ergodicity for a two-stage system comprised of batch processors under full-batch policy. When the batch processor is under no-idling policy, we identify two different conditions that would cause certain states to be transient.

These conditions can be used to reduce the time required to obtain the steady-state performance measures of the system, as well as serve as a guide for determining buffer sizes.

In Chapter 5, the no-idling policy has better performance than the full-batch policy when the arrival rate is low, while the full-batch policy outperforms the no-idling policy when the arrival rate is high. When the system is composed of two batch processors, this generalization is no longer always true. When the batch processor is under no-idling policy and its size is much higher than the feeder processor size, it is optimal to use no-idling policy at the feeder processor when the arrival rates are low or very high, and to use the full-batch policy in between.

Another exception to the general rule occurs when the batch processor size is an integer multiple of the feeder processor size, and the batch processor size is under full-batch policy. Under this scenario, using full-batch policy at the feeder processor is always better than using no-idling policy. When a full batch is unavailable at the batch processor, it will require an integer number of full batches from the feeder processor before it can start processing. When the feeder processor is under no-idling policy, it would take at least as long a time interval for the feeder processor to process the required number of jobs, as it would take if the feeder processor is under full-batch policy.

Using the greedy-look-ahead policy results in lower mean cycle time than using the myopic policy, with minimal loss in the maximum throughput rate. The larger the proclivity of the system to wait for additional jobs, the larger the disparity in the performance of the greedy-look-ahead and the myopic policies. Thus, the disparity is most visible when both processors are under full-batch policy. Furthermore, extending the strategy of constraining the control of a processor further upstream to the batch processor's anticipated needs does not result in immediate excessive throughput loss when the feeder processor is also a batch processor.

7.2 Contributions

We claim the following contributions:

- We provide a dynamic programming algorithm that minimizes the mean cycle time of jobs to be processed by a batch processor, where jobs belong to different job families and arrive at future intervals. We also provide an optimal DP algorithm where there are multiple batch processors at the batch processing stage.
- We propose a MPC-based heuristic when the problem has an infinite horizon. This heuristic is shown to significantly improve upon the performance of a popular look-ahead method. We also propose a heuristic when the number of batch processors at the stage is greater than one; this heuristic has better performance than an existing look-ahead method when no job families are ignored.
- We show that inducing positive correlation between the job families of future job arrivals can result in significant reduction in mean cycle time, regardless of batch processor policy. Furthermore, the magnitude of the improvement exceeds the improvement observed when a switch is made from a simple to a more computationally expensive batch processor policy. This is also generally true when the number of batch processors is increased.
- We look at a previously ignored problem of limiting the number of jobs that reach a predetermined queue time limit at the batch processor buffer. We consider four different models of varying complexity, and develop expressions to numerically calculate the probability of a prospective job exceeding the queue time limit, given the current system state. These probabilities are used to characterize the optimal threshold policies. We also provide the state space and the transition probability equations for the Markov Chain equivalents of three of the systems considered, when the feeder processor is under a threshold policy. These equations make it possible to determine the long-run performance of the systems under a particular threshold value.
- We use a Markov Chain model to characterize the long-run gross performance of a two- stage system processing two different job families with an upstream serial

processor and a downstream batch processor. We contrast the performance of systems with batch processors against systems composed of only serial processors.

- We determine that controlling an upstream serial processor such that it prefers to process jobs that the batch processor also wants to process can reduce the mean cycle time of jobs passing through the system, albeit with a small reduction in the maximum throughput rate. The reduction in mean cycle time is higher when the batch processor size is high, or when the batch processor policy allows the processor to idle. However, the incremental cycle time reduction in controlling another serial processor further upstream diminishes rapidly, and the accompanying throughput loss increases.
- We create Markov Chain models to characterize the long-run gross performance of a two- stage system processing two different job families with an upstream batch processor and a downstream batch processor with larger size. We prove the existence of transient states under certain conditions when only the downstream batch processor is under no-idling policy.
- We confirm that that controlling an upstream processor such that it prefers to process jobs that the batch processor yields the same conclusion when the upstream processor is a batch processor with smaller size. Furthermore, the incremental cycle time reduction associated with constraining another processor further upstream to the anticipated needs of the batch processor does not exhibit rapid decay.

In summary, we illustrate the feasibility of controlling processors upstream of the batch processor as a function of the batch processor, and its anticipated needs. We initially show how changes in the arrival stream the batch processor experiences can improve the performance of the batch processor in isolation. We also provide additional incentive to control the upstream processor by analyzing the problem of preventing processing time window violations at the batch processor buffer. Finally, we evaluate the idea of the upstream processor processing jobs according to the anticipated needs of the batch processor, and

determined that the mean cycle time of jobs passing through the system is reduced, whether the upstream processor is a serial processor, or another batch processor with smaller size.

7.3 Limitations

The most significant limitation of the work presented in this dissertation is the disjointedness in discussing the control of the batch processing stage, the sizing of the batch processor buffer and the control of the upstream processor. While the conclusions and insights from analyzing each subsection is still valid for larger systems, we have yet to show how the framework for an integrated control policy of the batch processor and its upstream processor can be transformed into a policy suitable for real-life control problems. A case study using models of systems that are sufficiently detailed to simulate real-life production can also provide a better estimate of the potential benefits of an integrated upstream processor-batch processor control policy.

For each subsection we choose to investigate, we adopt a simplified model of the real-life manufacturing system. As such, there are certain details of the real-life system that are ignored. For example, we discount the the possibility that the size of the batch processor can be dependent on the job family, which can occur in wafer fabs. We also ignore setup times at the batch processor, as these setups are usually performed on a much larger time-scale than the control issues we are focused on. We also do not extend analysis in Chapter 4 to Chapter 6 for the case where the number of processors per stage is greater than one. Furthermore, the typical wafer fab processes more than two job families, as what is assumed in Chapter 5 and Chapter 6. While these do not change the fundamental insights gathered from the research, it may have a significant effect on the absolute performance of the systems analyzed.

In the interest of keeping the solutions to problems mathematically tractable, certain assumptions were made that may not be strongly reflective of the real situation. In particular, the frequent use of geometric and exponential distributions is motivated by their desired mathematical properties, rather than by their good approximation of the actual distributions.

While these assumptions do not change the fundamental conclusions derived from the study, the actual optimal thresholds for the buffer size, for example, may be different from what our derived equations suggest. Furthermore, the estimated relative improvements we see when comparing different upstream processor policies may differ from what is experienced in practice. Thus, additional experimentation on more detailed systems is recommended before actual implementation can be considered.

7.4 Future Research Directions

The models looked into are merely theoretical approximations of a subsystem of the semiconductor wafer fab, of which several complicating factors are ignored. Thus, while we have shown the feasibility of controlling the upstream processors as a function of the batch processor's anticipated needs in improving system performance, we have yet to determine the over-all effect of combining these concepts into an actual control policy.

Implementing the concept of making the upstream processor control policy subservient to the batch processor's needs on an actual system will also require substantial experimentation to tune a policy to several layers of complexity, including processor failures, multiple stages per processor, set-up times, reentrancy and product mix, among others. As manufacturing systems are rarely composed of two or three stages, additional work on combining this concept with a control policy for larger systems is also a promising avenue of research. Evaluating the efficacy of making the upstream processor subservient to the needs of the batch processor for other objective functions are also interesting open problems.

Currently, the processing time windows for jobs entering the batch processor buffer are treated as constraints. These values are recommended values of process engineers, based on the expected contamination rate. It would be interesting to determine the optimal processing time window for jobs, given costs for contaminated parts, rewards for good parts, and the dynamics of the contamination process.

List of References

Aalto S., *"Optimal control of batch service queues with compound Poisson arrivals and finite service capacity,"* Mathematical Methods of Operations Research **48** (1998) 317-335

Aalto S., *"Optimal control of batch service queues with finite service capacity and linear holding costs,"* Mathematical Methods of Operations Research **51** (2000) 263-285

Adams J., Balas E. and Zawack D., *"The shifting bottleneck procedure for job-shop scheduling,"* Management Science **34** (1988) 391-401

Ahmadi J.H., Ahmadi R.H., Dasu S. and Tang C.S., *"Batching and scheduling jobs on batch and discrete processors,"* Operations Research **40** (1992) 750-63

Akcali E., Uzsoy R., Hiscock D.G., Moser A.L. and Teyner T.J., "Alternative loading and dispatching policies for furnace operations in semiconductor manufacturing: a comparison by simulation," Proceedings of the Winter Simulation Conference **2** (2000) 1428-1435 vol.2

Avramidis A.N., Healy K.J. and Uzsoy R., *"Control of a batch-processing machine: a computational approach,"* International Journal of Production Research **36** (1998) 3167-3181

Azizoglu M. and Webster S., *"Scheduling a batch processing machine with incompatible job families,"* Computers & Industrial Engineering **39** (2001) 325-335

Balasubramanian H., Monch L., Fowler J. and Pfund M., *"Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness,"* International Journal of Production Research **42** (2004) 1621-38

Bar-Noy A., Gupta S., Katz Y., Naor J., Schieber B. and Shachnai H., "Throughput maximization of real-time scheduling with batching," Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms (2002) 742-751

Bertsekas D.P., Dynamic programming and optimal control, Athena Scientific (2005)

Bhatnagar R., Chandra P., Loulou R. and Jin Q., *"Order release and product mix coordination in a complex PCB manufacturing line with batch processors,"* International Journal of Flexible Manufacturing Systems **11** (1999) 327-51

Boudhar M., *"Dynamic scheduling on a single batch processing machine with split compatibility graphs,"* Journal of Mathematical Modelling and Algorithms **2** (2003) 17-35

Boudhar M., *"Scheduling a batch processing machine with bipartite compatibility graphs,"* Mathematical Methods of Operations Research **57** (2003) 513-527

Cai M.C., Deng X.T., Feng H.D., Li G.J. and Liu G.Z., *"A PTAS for minimizing total completion time of bounded batch scheduling,"* International Journal of Foundations of Computer Science **13** (2002) 817-827

Chamness L., "2005: a year in review for the semiconductor equipment and materials market and trends moving forward," AVEM Seminar on the latest market trends (2006)

Chandra P. and Gupta S., *"Managing batch processors to reduce lead time in a semiconductor packaging line,"* International Journal of Production Research **35** (1997) 611-633

Chandru V., Lee C.Y. and Uzsoy R., *"Minimizing total completion time on batch processing machines "* International Journal of Production Research **31** (1993) 2097-2121

Chang S.H. and Gershwin S.B., *"Modeling and exact analysis of a production line with two unreliable batch machines and a finite buffer: Part 1-full batches,"* Operations Research Center, Massachusetts Institute of Technology (2005)

Chen B., Deng X.T. and Zang W.N., *"On-line scheduling a batch processing system to minimize total weighted job completion time,"* Journal of Combinatorial Optimization **8** (2004) 85-95

Chen J., Fu L., Lin M. and Huang A., *"Petri-Net and GA-Based approach to modeling, scheduling and performance evaluation for wafer fabrication,"* IEEE Transactions on Robotics and Automation **17** (2001) 619-636

Cheng T.C.E., Liu Z. and Yu W., *"Scheduling jobs with release dates and deadlines on a batch processing machine,"* IIE Transactions **33** (2001) 685-690

Cheng T.C.E., Yuan J.J. and Yang A.F., *"Scheduling a batch processing machine subject to precedence constraints, release dates, and identical processing times,"* Computers and Operations Research **32** (2005) 849-859

Chung S. and Huang H., *"The block-based cycle time estimation algorithm for wafer fabrication factories "* International Journal of Industrial Engineering **6** (1999) 307-316

Chung S.H., Pearn W.L., Lee A.H.I. and Ke W.T., *"Job order releasing and throughput planning for multi-priority orders in wafer fabs,"* International Journal of Production Research **41** (2003) 1765-1784

Conway R.W., Maxwell W.L. and Miller L.W., Theory of scheduling, Addison-Wesley Publishing Company (1967)

Damodaran P. and Srihari K., *"Mixed integer formulation to minimize makespan in a flow shop with batch processing machines,"* Mathematical and Computer Modelling **40** (2004) 1465-1472

Danneberg D., Tautenhahn T. and Werner F., *"A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size,"* Mathematical and Computer Modelling **29** (1999) 101-126

Deb R.K. and Serfozo R.F., *"Optimal control of batch service queues,"* Advances in Applied Probability **5** (1973) 340-361

Deng X.T., Feng H.D., Zhang P.X. and Zhu H., *"Minimizing mean completion time in a batch processing system,"* Algorithmica **38** (2004) 513-528

Deng X.T., Poon C.K. and Zhang Y.Z., *"Approximation algorithms in batch processing,"* Journal of Combinatorial Optimization **7** (2003) 247-257

Dobson G. and Nambimadom R.S., *"The batch loading and scheduling problem,"* Operations Research **49** (2001) 52-65

Duenyas I. and Neale J.J., *"Stochastic scheduling of batch processing machine with incompatible job families,"* Annals of Operations Research **70** (1997) 191-220

Dupont L. and Dhaenens-Flipo C., *"Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure,"* Computers and Operations Research **29** (2002) 807-819

Fonseca N.L.S.d. and Facanha R.d.A., *"The look-ahead-maximize-batch batching policy,"* IEEE Transactions on Multimedia **4** (2002) 114-120

Fowler J.W., Hogg G.L. and Phillips D.T., *"Control of multiproduct bulk server diffusion/oxidation processes. Part 2: Multiple servers,"* IIE Transactions (Institute of Industrial Engineers) **32** (2000) 167-176

Fowler J.W., Phillips D.T. and Hogg G.L., *"Real-time control of multiproduct bulk-service semiconductor manufacturing processes,"* IEEE Transactions on Semiconductor Manufacturing **5** (1992) 158-163

Fowler J.W., Phojanamongkolkij N., Cochran J.K. and Montgomery D.C., *"Optimal batching in a wafer fabrication facility using a multiproduct G/G/c model with batch processing,"* International Journal of Production Research **40** (2002) 275-292

Ganesan V.K., Gupta A.K. and Iyer S.A., "Bi-objective schedule control of batch processes in semiconductor manufacturing," Proceedings of the Winter Simulation Conference **2** (2004) 1077-1082 vol.2

Gershwin S.B., Manufacturing system engineering, private printing (2002)

Gershwin S.B., *"Sparse Matrix Iteration,"* Manuscript (2005)

Ghare P.M., *"Multichannel queueing system with bulk service,"* Operations Research **16** (1968) 189-192

Ghazvini F.J. and Dupont L., *"Minimizing mean flow times criteria on a single batch processing machien with non-identical job sizes,"* International Journal of Production Economics **55** (1998) 273-280

Glassey C.R. and Weng W.W., *"Dynamic batching heuristic for simultaneous processing,"* IEEE Transactions on Semiconductor Manufacturing **4** (1991) 77-82

Gupta A.K., Ganesan V.K. and Sivakumar A.I., "Hot lot management: minimizing cycle time in batch processes," IEEE International Engineering Management Conference **3** (2004) 1217-1221 Vol.3

Gupta A.K. and Sivakumar A.I., *"Job shop scheduling techniques in semiconductor manufacturing,"* International Journal of Advanced Manufacturing Technology **27** (2006) 1163-1169

Gupta A.K. and Sivakumar A.I., *"Optimization of due-date objectives in scheduling semiconductor batch manufacturing,"* International Journal of Machine Tools & Manufacture **46** (2006) 1671-9

Gupta A.K., Sivakumar A.I. and Ganesan V.K., "Look ahead batching to minimize Earliness/Tardiness measures in batch processes," 2004 IEEE Conference on Robotics, Automation and Mechatronics (2004)

Gurnani H., Anupindi R. and Akella R., *"Control of batch processing systems in semiconductor wafer fabrication facilities,"* IEEE Transactions on Semiconductor Manufacturing **5** (1992) 319-328

Hochbaum D.S. and Landy D., *"Scheduling semiconductor burn-in operations to minimize total flowtime,"* Operations Research **45** (1997) 874-885

Hopp W. and Spearman M., Factory physics: foundations of manufacturing management, Irwin/McGraw-Hill (2000)

Hsieh B.W., Chen C.H. and Chang S.C., *"Scheduling semiconductor wafer fabrication by using ordinal optimization-based simulation,"* IEEE Transactions on Robotics and Automation **17** (2001) 599-608

Huff M.A., *"MEMS and nanotechnology Exchange Catalog,"*

Hung Y., *"Scheduling of mask shop e-beam writers,"* IEEE Transactions on Semiconductor Manufacturing **11** (1998) 165-172

Ikura Y. and Gimple M., *"Efficient scheduling algorithms for a single batch processing machine,"* Operations Research Letters **5** (1986) 61-65

In J., Hong Y. and Jun C.-H., *"Analysis of a batch machine fed by an unreliable discrete machine,"* Production Planning and Control **14** (2003) 656-661

Jacobs J.H., van Bakel P.P., Etman L.F.P. and Rooda J.E., *"Quantifying variability of batching equipment using effective process times,"* IEEE Transactions on Semiconductor Manufacturing **19** (2006) 269-275

Jolai F., *"Minimizing number of tardy jobs on a batch processing machine with incompatible job families,"* European Journal of Operational Research **162** (2005) 184-190

Jones A. and Rabelo L., *"Survey of job shop scheduling techniques,"* NISTIR, National Institute of Standards and Technology, Gaithersburg, MD. (1998)

Kim B. and Kim S., *"Application of genetic algorithms for scheduling batch-discrete production system,"* Production Planning and Control **13** (2002) 155-165

Kim Y.-D., Kim J.-U., Lim S.-K. and Jun H.-B., *"Due-date based scheduling and control policies in a multiproduct semiconductor wafer fabrication facility,"* IEEE Transactions on Semiconductor Manufacturing **11** (1998) 155-164

Koh S.-G., Koo P.-H., Kim D.-C. and Hur W.-S., *"Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families,"* International Journal of Production Economics **98** (2005) 81-96

Koh S., Koo P., Ha J. and Lee W., *"Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families,"* International Journal of Production Research **42** (2004) 4091-4107

Koole G. and Righter R., *"A stochastic batching and scheduling problem,"* Probability in the Engineering and Informational Sciences **15** (2001) 465-479

LaPedus M., *"Fab utilization jumps to 95 percent,"* Web article for EE Times (Link: http://www.eetimes.com/news/semi/showArticle.jhtml;jsessionid=VUNUTM0EHERO0QSNDLRSKH0CJUNN2JVN?articleID=193004233), (2006)

Law A.M. and Kelton W.D., Simulation modeling and analysis, McGraw-Hill (2000)

Lee C., Uzsoy R. and Martin-Vega L.A., *"Efficient algorithms for scheduling semiconductor burn-in operations,"* Operations Research **40** (1992) 764-775

Lee C.Y., Lei L. and Pinedo M., *"Current trends in deterministic scheduling,"* Annals of Operations Research **70** (1997) 1-41

Lenstra J.K., Kan A. and Brucker P., *"Complexity of machine scheduling problems,"* Annals of Discrete Mathematics **1** (1977) 343-362

Li C.-L. and Lee C.-Y., *"Scheduling with agreeable release times and due dates on a batch processing machine,"* European Journal of Operational Research **96** (1997) 564-569

Li S., Li G., Wang X. and Liu Q., *"Minimizing makespan on a single batching machine with release times and non-identical job sizes,"* Operations Research Letters **33** (2005) 157-164

Li S., Li G. and Zhang S., *"Minimizing maximum lateness on identical parallel batch processing machines,"* Lecture Notes in Computer Science 3106 (2004) 229-237

Li S., Li G. and Zhang S., *"Minimizing makespan with release times on identical parallel batching machines,"* Discrete Applied Mathematics **148** (2005) 127-134

Li W. and Yuan J., *"Single machine parallel batch scheduling problem with release dates and three hierarchical criteria to minimize makespan, machine occupation time and stocking cost,"* International Journal of Production Economics **102** (2006) 143-148

Liu Z. and Yu W., *"Scheduling one batch processor subject to job release dates,"* Discrete Applied Mathematics **105** (2000) 129-136

Liu Z., Yuan J. and Cheng T.C.E., *"On scheduling an unbounded batch machine,"* Operations Research Letters **31** (2003) 42-48

Madou M.J., Fundamentals of microfabrication: the science of miniaturization, CRC Press LLC (2002)

Makis V., *"Optimal control of a batch service queueing system with bounded waiting time,"* Kybernetika **21** (1985) 262-271

Mason S.J. and Fowler J.W., "Maximizing delivery performance in semiconductor wafer fabrication facilities," Proceedings of the Winter Simulation Conference **2** (2000) 1458-1463 vol.2

Mason S.J., Fowler J.W. and Carlyle W.M., *"A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops,"* Journal of Scheduling **5** (2002) 247-62

Mathirajan M. and Sivakumar A.I., *"A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor,"* International Journal of Advanced Manufacturing Technology **29** (2006) 990-1001

Mehta S.V. and Uzsoy R., *"Minimizing total tardiness on a batch processing machine with incompatible job families,"* IIE Transactions **30** (1998) 165-178

Melouk S., Damodaran P. and Chang P.-Y., *"Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing,"* International Journal of Production Economics **87** (2004) 141-147

Ming-Guang H., Pao-Long C. and Ying-Chyi C., *"Analytic approximations for multiserver batch-service workstations with multiple process recipes in semiconductor wafer fabrication,"* IEEE Transactions on Semiconductor Manufacturing **14** (2001) 395-405

Monch L., Balasubramanian H., Fowler J.W. and Pfund M.E., *"Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times,"* Computers and Operations Research **32** (2005) 2731-2750

Monch L. and Driessel R., *"A distributed shifting bottleneck heuristic for complex job shops,"* Computers and Industrial Engineering **49** (2005) 363-380

Monch L., Schabacker R., Pabst D. and Fowler J.W., *"Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops,"* European Journal of Operational Research **in press** (2006)

Monch L. and Unbehaun R., *"Decomposition heuristics for minimizing earliness-tardiness on parallel burn-in ovens with a common due date,"* Computers and Operations Research (2006)

Monch L., Zimmermann J. and Otto P., *"Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines,"* Engineering Applications of Artificial Intelligence **19** (2006) 235-45

Neale J.J. and Duenyas I., *"Control of manufacturing networks which contain a batch processing machine,"* IIE Transactions **32** (2000) 1027-1041

Neale J.J. and Duenyas I., *"Control of a batch processing machine serving compatible job families,"* IIE Transactions **35** (2003) 699-710

Noben R., van Driel R. and Claasen-Vujcic T., "Cycle time advantages of mini batch manufacturing and integrated metrology in a 300 mm vertical furnace," 2001 IEEE International Semiconductor Manufacturing Symposium (2001) 411-414

Oey K. and Mason S.J., "Scheduling batch processing machines in complex job shops," Proceedings of the Winter Simulation Conference **2** (2001) 1200-1207 vol.2

Oulamara A., *"Makespan minimization in a no-wait flow shop problem with two batching machines,"* Computers and Operations Research **34** (2005) 1033-1050

Papadimitriou C.H. and Steiglitz K., Combinatorial optimization: algorithms and complexity, Prentice-Hall (1998)

Park Y., Kim S. and Jun C.-H., *"Mean value analysis of re-entrant line with batch machines and multi-class jobs,"* Computers and Operations Research **29** (2002) 1009-1024

Park Y., Kim S. and Jun C., *"Performance analysis of re-entrant flow shop with single-job and batch machines using mean value analysis,"* Production Planning and Control **11** (2000) 537-546

Pei-Chann C. and Hui-Mei W., *"A heuristic for a batch processing machine scheduled to minimise total completion time with non-identical job sizes,"* International Journal of Advanced Manufacturing Technology **24** (2004) 615-20

Pei-Chann C., Yun-Shiow C. and Hui-Mei W., "Dynamic scheduling problem of batch processing machine in semiconductor burn-in operations," Proceedings of the ICCSA 2005 International Conference (2005) 172-81

Perez I.C., Fowler J.W. and Carlyle W.M., *"Minimizing total weighted tardiness on a single batch process machine with incompatible job families,"* Computers and Operations Research **32** (2005) 327-341

Poon C.K. and Zhang P., *"Minimizing makespan in batch machine scheduling,"* Algorithmica (New York) **39** (2004) 155-174

Ram B. and Patel G., "Modeling furnace operations using simulation and heuristics," Proceedings of the Winter Simulation Conference (1998) 957-963

Robinson J.K., Fowler J.W. and Bard J.F., *"A review of real-time control strategies for furnace batch sizing in semiconductor manufacturing,"* FabTime, Inc. (1995)

Robinson J.K., Fowler J.W. and Bard J.F., *"The use of upstream and downstream information in scheduling semiconductor batch operations,"* International Journal of Production Research **33** (1995) 1849-1869

Rostami S. and Hamidzadeh B., *"An optimal residency-aware scheduling technique for cluster tools with buffer module,"* IEEE Transactions on Semiconductor Manufacturing **17** (2004) 68-73

Rulkens H.J.A., Van Campen E.J.J., Van Herk J. and Rooda J.E., "Batch size optimization of a furnace and pre-clean area by using dynamic simulations," IEEE/SEMI advanced semiconductor manufacturing conference (1998) 439-444

Schomig A.K. and Kahnt M., "Performance modelling of pull manufacturing systems with batch servers," Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation **vol.3** (1995) 175-83

Schwindt C. and Trautmann N., *"Scheduling the production of rolling ingots: industrial context, model and solution method,"* International Transactions in Operational Research **10** (2003) 547-563

Simons J.V.J. and Russell G.R., *"A case study of batching in a mass service operation,"* Journal of Operations Management **20** (2002) 577-592

Sivakumar A.I., "Optimization of cycle time and utilization in semiconductor test manufacturing using simulation based, on-line, near-real-time scheduling system," Proceedings of the Winter Simulation Conference **1** (1999) 727-735 vol.1

Sivakumar A.I., "Simulation based cause and effect analysis of cycle time distribution in semiconductor backend," Proceedings of the Winter Simulation Conference **2** (2000) 1464-1471 vol.2

Solomon L., Fowler J.W., Pfund M. and Jensen P.H., *"The inclusion of future arrivals and downstream setups into wafer fabrication batch processing decisions,"* Journal of Electronics Manufacturing **11** (2002) 149-159

Sung C.S. and Choung Y.I., *"Neural network approach for batching decisions in wafer fabrication,"* International Journal of Production Research **37** (1999) 3101-3114

Sung C.S. and Choung Y.I., *"Minimizing makespan on a single burn-in oven in semiconductor manufacturing,"* European Journal of Operational Research **120** (2000) 559-74

Sung C.S., Choung Y.I., Hong J.M. and Kim Y.H., *"Minimizing makespan on a single burn-in oven with job families and dynamic job arrivals,"* Computers and Operations Research **29** (2002) 995-1007

Sung C.S. and Kim Y.H., *"Minimizing due date related performance measures on two batch processing machines,"* European Journal of Operational Research **147** (2003) 644-656

Sung C.S., Kim Y.H. and Yoon S.H., *"A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines,"* European Journal of Operational Research **121** (2000) 179-192

Sung C.S. and Min J.I., *"Scheduling in a two-machine flowshop with batch processing machine(s) for earliness/tardiness measure under a common due date,"* European Journal of Operational Research **131** (2001) 95-106

Tsai C.H., Feng Y.M. and Li R.K., *"A hybrid dispatching rule in wafer fabrication factories,"* International Journal of the Computer, the Internet and Management **11** (2003) 64-72

Uzsoy R., *"Scheduling a single batch processing machine with non-identical job sizes,"* International Journal of Production Research **32** (1994) 1615-1635

Uzsoy R., *"Scheduling batch processing machines with incompatible job families,"* International Journal of Production Research **33** (1995) 2685-2708

Uzsoy R. and Wang C.S., "Decomposition procedures for global scheduling of complex job shops," IEEE/CPMT International Electronics Manufacturing Technology Symposium (1997) 425-429

Uzsoy R. and Yang Y., *"Minimizing total weighted completion time on a single batch processing machine,"* Production and Operations Management **6** (1997) 57-73

Vaessens R.J.M., Aarts E.H.L. and Lenstra J.K., *"Job shop scheduling by local search,"* INFORMS Journal of computing **8** (1996) 302-317

Van Der Zee D.J., "Real-time adaptive control of multi-product multi-server bulk service processes," Proceedings of the Winter Simulation Conference (2001) 930-936

Van Der Zee D.J., *"Dynamic scheduling of batch servers with compatible product families,"* International Journal of Production Research **42** (2004) 4803-4826

Van Der Zee D.J., Van Harten A. and Schuur P., *"On-line scheduling of multi-server batch operations,"* IIE Transactions **33** (2001) 569-586

Wang C.-S. and Uzsoy R., *"A genetic algorithm to minimize maximum lateness on a batch processing machine,"* Computers and Operations Research **29** (2002) 1621-1640

Webster S. and Baker K.R., *"Scheduling groups of jobs on a single machine,"* Operations Research **43** (1995) 692-703

Wein L.M., *"Scheduling semiconductor wafer fabrication,"* IEEE Transactions on Semiconductor Manufacturing **1** (1988) 115-130

Wein L.M., *"On the relationship between yield and cycle time in semiconductor wafer fabrication,"* IEEE Transactions on Semiconductor Manufacturing **5** (1992) 156-158

Weng W.W. and Leachman R.C., *"An improved methodology for real-time production decisions at batch-process work stations,"* IEEE Transactions on Semiconductor Manufacturing **6** (1993) 219-225

Winston W., *"Optimality of the shortest line discipline "* Journal of Applied Probability **14** (1977) 181-189

Xia C.H., Michailidis G., Bambos N. and Glynn P.W., *"Optimal control of parallel queues with batch service,"* Probability in the Engineering and Informational Sciences **16** (2002) 289-307

Xiangtong Q. and Fengsheng T., *"Earliness and tardiness scheduling problems on a batch processor,"* Discrete Applied Mathematics **98** (1999) 131-45

Yoon H.J. and Lee D.Y., *"Online scheduling of integrated single-wafer processing tools with temporal constraints,"* IEEE Transactions on Semiconductor Manufacturing **18** (2005) 390-398

Yuan J.J., Liu Z.H., Ng C.T. and Cheng T.C.E., *"The unbounded single machine parallel batch scheduling problem with family jobs and release dates to minimize makespan,"* Theoretical Computer Science **320** (2004) 199-212

Zhang G.C., Cai X.Q. and Wong C.K., *"Optimal on-line algorithms for scheduling on parallel batch processing machines,"* IIE Transactions **35** (2003) 175-181

Zhang Y., Cao Z. and Bai Q., "A PTAS for scheduling on agreeable unrelated parallel batch processing machines with dynamic job arrivals," proceedings of the First International Conference on Algorithmic Applications in Management (2005) 162-171

Appendix A

Control of multiple batch processors in a single stage

A.1 Problem Statement

We consider a manufacturing stage containing $K > 1$ identical batch processors in parallel, each with size Q. A batch processor has size Q. There are n jobs to be scheduled, and each Job i becomes available at time instance r_i, and belongs to a Job Family j. There are m total job families. Only jobs belonging to the same family can be batched together. The processing time of a batch is dependent only on the job family that is currently being processed. The objective is to minimize the sum of completion times. The batch processor is assumed to be initially available, and jobs are indexed in increasing order of their release times.

This is a strongly NP-Hard problem, as the problem obtained when $K = 1$ is strongly NP-Complete (Section 3.3.2). We provide both an integer linear programming and a dynamic programming formulation that can solve small problem instances to optimality.

A.2 Integer Linear Programming Formulation

This problem can be written as an integer linear program.

We define the following variables:

- n number of jobs to be scheduled
- m number of job families
- i index for jobs, $i=1,2,...,n$. Let I be the set of all jobs.
- j index for families $j=1,2,...,m$.
- k index for batch processors $k=1,2,...,K$.
- l index for batches $l=1,2,...,n$. Within the same batch processor, batches are processed in ascending order of the index.
- f_j set of job indices in family j. The sets f_j, $j=1, 2, ..., m$ form a partition of the set I. If $f_1=\{1, 2, 3, 4\}$, Jobs One, Two, Three, Four belong to Family One.
- Q batch processor size.
- r_i time instance job i arrives in front of the batch processor.
- p_j processing time for family j. For convenience, assume $p_j \leq p_{j+1}$ for $j = 1$ to $m-1$.
- C_l time instance the l^{th} batch finishes processing.
- S_l time instance the l^{th} batch starts processing.

The decision variables are:

- x_{il} =1 if job i is assigned to batch l and =0 otherwise
- y_{jl} =1 if family j is assigned to batch l and =0 otherwise
- z_{lk} =1 if l^{th} batch is assigned to batch processor k and =0 otherwise

The integer linear program formulation becomes:

$$\text{Min} \sum_{l=1}^{n}\sum_{i=1}^{n} C_l x_{il} \qquad \text{(Equation A- 1)}$$

Subject to:

$$\sum_{i=1}^{n} x_{il} \leq Q \qquad \forall l \qquad \text{(Equation A- 2)}$$

$$\sum_{l=1}^{n} x_{il} = 1 \qquad \forall i \qquad \text{(Equation A- 3)}$$

$$\sum_{j=1}^{m} y_{jl} \leq 1 \qquad \forall l \qquad \text{(Equation A- 4)}$$

$$x_{il} \leq y_{jl},\ i \in f_j \qquad \forall l,\ \forall j \qquad \text{(Equation A- 5)}$$

$$C_l = S_l + \sum_{j=1}^{m} p_j y_{jl} \qquad \forall l \qquad \text{(Equation A- 6)}$$

$$S_l \geq r_i x_{il} \qquad \forall l\ ,\ \forall i \qquad \text{(Equation A- 7)}$$

$$C_q - S_l \leq (r_n + n \times p_m) \times (2 - z_{qk} - z_{lk}) \qquad \forall q < l, \forall k, \forall l \qquad \text{(Equation A- 8)}$$

$$\sum_{k=1}^{K} z_{lk} = 1 \qquad \forall l \qquad \text{(Equation A- 9)}$$

$$x_{il}, y_{jl}, z_{lk} \in \{0,1\} \qquad \forall l\ , \forall i\ , \forall j\ , \forall k \qquad \text{(Equation A- 10)}$$

Equation A-1: Objective function minimizes the total completion time

Equation A-2: All batches do not violate the size constraint.

Equation A-3: All jobs get assigned to a batch.

Equation A-4: At most one job family is assigned to any batch.

Equation A-5: A job is not assigned to a batch if the batch is not assigned to the job family the job belongs to. Equation A-4 and A-5 ensure only empty batches are not assigned any job families.

Equation A-6: Computation of completion time of each batch

Equation A-7: A batch does not start processing until all jobs assigned to the batch has arrived

Equation A-8: A batch does not start processing until the processor it is assigned to finishes previously assigned batches. For the optimal solution to the problem, for any two batches q and l, where $q < l$, the following condition will not exist: $C_q - S_l > r_n + n \times P_m$

This is because within the time interval $(C_q,\ C_q + r_n + n \times P_m)$, it is already possible to process all the jobs to be processed. Thus, Equation A-8 is only restrictive if $(r_n + n \times P_m) \times (2 - z_{qk} - z_{lk}) = 0$, which occurs when the batches q and l are assigned to the same processor.

When q and l are assigned to the same processor, batch q is processed earlier than batch l, since $q < l$. We thus require that the completion time of batch q, C_q, be less than the start time of batch l, S_l.

Equation A-9: A batch is assigned to only one processor.

Equation A-10: Constrain integer decision variables to 0 and 1

A solution that reduces the sum of completion times is also reduces the mean cycle time, since $\left(\sum_{l=1}^{n}\sum_{i=1}^{n} C_l x_{il} - \sum_{i=1}^{n} r_i\right) \Big/ n = AveCycleTime$.

A.3 DP Algorithm

We provide an alternative to solving the problem as an integer linear program (ILP) by presenting a dynamic programming (DP) formulation of the problem.

A.3.1 State representation

We define the following variables:

- t – time instance decision has to be made
- $D_k(t)$ – time instance Processor k will be available. If, at Time Instance t, Processor k is idle, then $D_k(t) = 0$.
- $q_j(t)$ – queue length for jobs belonging to Job Family j at Time Instance t. There are m job families.
- a_i – job family to which Job i belongs to. If f_j is the set of job indices in Family j, then $a_i = \{j \mid i \in f_j\}$.
- r_i – earliest time instance Job i can be processed.
- n_j – total number of jobs to be scheduled that belong to Job Family j. $\sum_{j=1}^{m} n_j = n$

When there are $K > 1$ batch processors with a common queue, the system state is: $X(t) = (t, D_1(t), D_2(t),..., D_K(t), q_1(t), q_2(t),..., q_m(t))$. It is convenient to consider $X(t)$ to be composed of three components: the scalar t, the $1 \times K$ vector $\overrightarrow{D(t)} = D_1(t), D_2(t),..., D_K(t)$ representing the future time instances processors become idle, and the $1 \times m$ vector $\overrightarrow{Q(t)} = q_1(t), q_2(t),..., q_m(t)$ representing the buffer contents. Since all processors are

identical to each other, we can reassign the processor index such that $\overrightarrow{D(t)}$ is strictly non-decreasing.

A.3.2 Set of feasible controls

Let $b_j(t)$ be the number of batches composed of Job Family j to start processing at Time Instance t. Then, at Time Instance t, the set of possible controls is:

- Wait for the next job arrival, or,
- Process $b_j(t)$ batches of jobs belonging to Job Family j. We assume that batches are assigned to processors such that $\overrightarrow{D(t)}$ is non-decreasing.

We assume that within each Job Family j, jobs are grouped into batches such that only the last batch processed at Time Instance t may be partially full. Let $e(t) = \left\| D_k(t) \mid D_k(t) = 0; i = 1,2,...,K \right\|$. $e(t)$ is the number of available processors at Time Instance t. The control $u(t)$ at time instance t is the vector: $(b_1(t), b_2(t), ..., b_m(t))$, with the constraint that the total number of batches to be loaded, $\sum_{j=1}^{m} b_j(t) \le e(t)$. $u(t) = (0,0,...,0)$ is the control used to indicate waiting for the next job arrival. Let $U(t)$ be the set of all possible controls at Time Instance t.

A.3.3 State transition

We declare the following variables:

- $d_k(t)$ - amount of processing time assigned to Batch Processor k at Time Instance t. If Batch Processor k is to be left idle, or if Batch Processor k is busy and unable to process a new batch, then $d_k(t) = 0$. The components $d_1(t), d_2(t), ..., d_K(t)$ are obtained from $b_1(t), b_2(t), ..., b_m(t)$ by:
    - generating a list of idle processors

  - assigning an idle processor to each batch. If the batch assigned to Processor k belongs to job family j, then $d_k(t) = t + P_j$.

- $y_j(t)$ - the number of jobs belonging to Job Family j that are loaded at Time Instance t. $y_j(t) = min(b_j(t) \times Q, q_j(t))$, for all j. Let $\overrightarrow{Y(t)} = y_1(t), y_2(t), ..., y_m(t)$
- $s(t)$ - $\min\{r_i : r_i > t\}$. $s(t)$ returns the time instance of the next job arrival. Set $s(t)$ = infinity if no more future job arrivals are expected.
- $\varphi(t)$ - $min(D_k(t) | D_k(t) > 0)$. $\varphi(t)$ is the earliest time a batch processor that is busy at Time Instance t will become available. If $\{D_k(t) | D_k(t) > 0\}$ is NULL, set $\varphi(t)$ = infinity.
- $\alpha(t)$ - $min(d_k(t) | d_k(t) > 0)$. $\alpha(t)$ is the earliest time instance a processor that starts processing at Time Instance t becomes idle. If $\{d_k(t) | d_k(t) > 0\}$ is NULL, set $\alpha(t)$ = infinity.
- $V_j(x, y)$ - $\left\| i \mid \left(x < r_i \le y\right) \cap a_i = j \right\|$. $V_j(x, y)$ returns the number of jobs belonging to job family j that arrives in front of the batch processor in the time interval $(x, y]$. Let $\overrightarrow{V(x,y)} = V_1(x,y), V_2(x,y), ..., V_m(x,y)$.
- $c_k(t)$ – flag variable to determine whether Processor k becomes idle at Time Instance t. $c_k(t') = 0$ if $D_k(t) + d_k(t) = t'$ and is 1 otherwise.

Applying the control $u(t)$ to the current state $X(t)$, the future state $X(t')$ is obtained in the following manner.

- t' is the next time instance a decision has to be made. Set $t' = min\ (s(t), \varphi(t), \alpha(t))$.
- $\overrightarrow{Q(t')} = \overrightarrow{Q(t)} - \overrightarrow{Y(t)} + \overrightarrow{V(t,t')}$
- $D_k(t') = c_k(t') \times \left(D_k(t) + d_k(t)\right)$ for k = 1 to K. The variables $D_k(t')$ are then sorted in increasing order. The sorted vector formed is $\overrightarrow{D(t')}$.

Forcing $\overrightarrow{D(t)}$ to be always non-decreasing will allow us to recognize only one permutation of $\overrightarrow{D(t)}$, which reduces the state space. For example, if the value of being in state $X_1(t)$ is required, and $X_1(t)$ is identical to another state $X_2(t)$ except that their corresponding $\overrightarrow{D(t)}$

vectors do not have the same ordering, then $f(X_1(t)) = f(X_2(t))$ as the two states are identical except for the indexing of the processors.

The recursion equation is:

$$f(t,\overrightarrow{D(t)},\overrightarrow{Q(t)}) = \min_{u(t)\in U(t)}\left[\sum_{j=1}^{m} y_j(t)\times(t+p_j)+f\left(t',\overrightarrow{D(t')},\overrightarrow{Q(t')}\right)\right]$$

The boundary conditions are:

$\min\{r_i : r_i > t\} = \infty$ for $t > r_n$

let $\bullet \geq 0$, then $f(\infty,\bullet,\bullet,...,\bullet) = \infty$

$f(t,0,0,...,0) = 0$ for $t > r_n$

The minimum completion time is obtained by $f(0,0,...,0,0)$.

A.4 Complexity Analysis

Input analysis:

The input size is bounded by: $\log Q + n(\log r_n + \log m) + m \log p_m + \log K$, with the following breakdown:

- batch processor size Q — $\log Q < n$
- n release times — $n \log r_n$
- n job families — $n \log m$
- m processing times — $m \log p_m$
- number of batch processors — $\log K$

We assume $m > r_n$ and $m > p_m$ to state the bound in terms of n, m and K. The input size is bounded from above by $\log n + (2n + m)\log m + \log K$.

Time analysis:

There can be at most n distinct time instances a job will arrive. For each Job Family j, the maximum number of processed batches is n_j, when each batch is composed of only one job, and $n_j < n$. An upper bound on the number of different values the first component of the state

vector can take is $n(n + 1)^m$. For the same reason, each of the next K components is bounded by: $n(n + 1)^m$. Thus, there can be $[n(n + 1)^m]^{K+1} = n^{K+1}(n + 1)^{m(K+1)}$ different values for the first $K + 1$ components.

The buffer level for each job family j can vary from 0 to n_j, and $n_j < n$. Each of the remaining components of the state vector can have up to $(n + 1)$ values each. This provides an upper bound of $n^{K+1}(n + 1)^{m(K+2)}$ different states. A tighter bound can be achieved by invoking the arithmetic mean-geometric mean inequality, and substituting $\lceil nm^{-1} + 1 \rceil^m$ for $(n+1)^m$. This results in an upper bound of $n^{K+1}\lceil nm^{-1} + 1 \rceil^{m(K+2)}$ states.

At each state, there can be at most K batch processors available, and $\sum_{j=1}^{m} b_j(t)$ can vary from 0 to K. $u(t) = (b_1(t), b_2(t), ..., b_m(t))$ can have $\lceil Km^{-1} + 1 \rceil^m$ values (from the arithmetic mean-geometric mean inequality).

Even with the simplifying assumption that obtaining $X(t')$ from $X(t)$ and $u(t)$ takes constant time, finding the minimum among the values associated with $\lceil Km^{-1} + 1 \rceil^m$ possible actions takes $O\left(\lceil Km^{-1} + 1 \rceil^m\right)$ time. As the upper bound on the number of states is $n^{K+1}\lceil nm^{-1} + 1 \rceil^{m(K+2)}$, the complexity of the algorithm is $O\left(n^{K+1}\lceil Km^{-1} + 1 \rceil^m \lceil nm^{-1} + 1 \rceil^{m(K+2)}\right)$. This is not an efficient algorithm, and only very small problem instances may be solved.

A.5 Suggested Heuristic Development

In Chapter 3, a heuristic based on Model Predictive Control (MPC) was proposed, and it was determined that this MPC-based heuristic, with properly selected parameters, has significantly lower mean cycle time than a popular look-ahead method proposed by [Fowler *et al.* 1992].

We discuss how the MPC-based heuristic developed in Chapter 3 can be adopted for the case when there are multiple processors in parallel in this section

A.5.1 Adaptation of the MPC-based heuristic to multiple processors

When $K > 1$, the problem becomes more complex, due to two reasons:

1. the need to keep track of the time instances each of the K processors becomes available
2. the possibility of processing more than one batch at the same time instance.

The first reason causes the state space to increase. The second reason causes the control space at each instance to increase. These problems are not alleviated when the number of future job arrivals and the number of job families considered are truncated. Thus, a straightforward implementation of MPC for this problem will still result in long computational requirements if the number of processors is large.

We propose to use the MPC-based heuristic in Chapter 3 as the kernel for a heuristic when $K > 1$. At Time Instance t, if there is only one batch processor available, we apply the MPC-based heuristic, without any modification. This assumes that there is only one batch processor, and risks overestimating the cost of processing a batch immediately, since other batch processors may become available to process the remaining batches before the current batch finishes processing.

When there are two or more idle batch processors, we evaluate three different control policies. When the manufacturing stage contains only a single processor, if it is possible to process a batch of jobs and have the processor available before the next job arrives, then it is never optimal to wait for new job arrivals (Section 3.3.3.2). If there is more than one processor idle at the current time instance, then the next time instance a processor will become available will always be before any job arrival. Thus, this suggests that batch processors should not be left idle as long as there will be at least one idle processor left. We develop two policies based on this idea. In the third policy, we apply the MPC-based heuristic at any time instance a batch processor is available. The three different policies are:

- Policy One:

When there is more than one batch processor idle, we treat the problem as a static scheduling problem, where no future arrivals are considered. We use the heuristic proposed by [Uzsoy 1995] for the static version of our problem. Batches are processed according to the weighted

shortest processing time rule, and we assign batches to processors such that there will be at least one idle processor left. If there are still jobs queueing at the batch processing stage buffer, the remaining processor is controlled via the MPC-based heuristic.

- Policy Two:

At time instance t where there is more than one batch processor available, we iteratively use a modified version of the MPC-based heuristic (called MPC_MOD) to determine the decision to be made for each processor. If there are m job families, the (standard) MPC-based heuristic will generate 1 of $(m+1)$ possible controls: either wait for the state to change, or process a batch from a certain job family j. MPC_MOD is a modification of the MPC-based heuristic which does not allow the batch processor to wait for more job arrivals, at time instance t. Thus, at time instance t, the control space of MPC_MOD has only m elements, and MPC_MOD can only choose which job family to process. If there are k batch processors available at a given time instance, the MPC_MOD is performed for each batch processor until no more jobs are queued, or only one idle processor remains. If there are still jobs queueing at the batch processing stage buffer, the remaining processor is controlled via the MPC-based heuristic.

- Policy Three:

The third policy simply uses the unmodified MPC-based heuristic for all idle processors, one by one. If there are k batch processors available, the unmodified MPC-based heuristic is performed up until the heuristic instructs a processor to wait for future jobs, or until all k batch processors are assigned a batch.

We illustrate these three policies via an example. Suppose, at time instance t, there are two idle batch processors BP_1 and BP_2, each with size of six. There are three jobs from Family One and four jobs from Family Two queueing at the stage, and only one future job arrival remains. This future job arrival belongs to Family One, and will arrive at $(t+0.5)$. Family One has processing time of 2 time units, and Family Two has processing time of 3 time units.

Policy One looks at the current queue contents, and determines that 3 jobs belonging to Job Family One is to be loaded to BP_1 immediately. This is because Family One has shorter weighted processing time than Family Two ($3/2 > 4/3$). Since only one processor (BP_2) is left

idle, BP_2 is controlled via the MPC-based heuristic. If there were two more processors (BP_3 and BP_4) idle, then BP_2 would be assigned 4 jobs of Job Family Two, and BP_3 and BP_4 would be left idle due to lack of jobs.

Policy Two controls BP_1 by ignoring all other processors and determining the optimal sequence of events that minimizes the total completion time, conditioned on a batch being assigned to BP_1 at Time Instance *t*. Thus, Policy Two compares the relative merits of:

a.) processing Job Family Two at Time Instance *t* followed by Family One at *t+3*,

b.) processing Family One at *t*, then another batch of Family One (comprised of the future job arrival) at *t*+2 then processing Family Two at *t*+4,

c.) processing Family One at *t*, processing Family Two at *t*+2 and processing Family One at *t*+5.

In this case, sequence **c** is optimal and 3 jobs from Family One are assigned to BP_1, with the control of BP_2 left to the MPC-based heuristic.

If there were two more processors (BP_3 and BP_4) idle, then BP_2 would be assigned four jobs from Family Two, since there are no other jobs at the queue and BP_2 is forced by Policy Two to process a job as long as there are jobs queued and at least one other processor is idle. BP_3 and BP_4 would be left idle due to lack of jobs at the queue.

Policy Three initially ignores all other processors and determines the optimal sequence of events to minimize the total completion time, assuming only a single processor is present at the batch processing stage. Compared to the three sequences available at Policy Two for BP_1, Policy Three has the additional option of **d.)** waiting for the future job arrival at Time Instance *t*+0.5, processing Family One at *t*+0.5 and then processing Family Two at *t*+2.5 is available for BP_1. In this case, the optimal sequence is **d**, and BP_1 waits for the next job arrival. This automatically means BP_2 also waits for the next job arrival.

A.5.2 Experimental Setup for Performance Evaluation of proposed policies

We use the MPC-based heuristic (4,10) as a kernel of the three policies to be evaluated. (4,10) means that only up to four job families and a maximum of 10 future job arrivals are considered. The benchmark policy is an extension of the Next Arrival Control Heuristic for multiple job families proposed by [Fowler *et al.* 1992] when the number of processors is greater than one. This policy is documented in [Fowler *et al.* 2000], which we call NACH-MM. NACH-MM also opts to immediately process a batch if there are more than one processor available, and [Fowler *et al.* 2000] opined that this is the cause for the observed deterioration of the relative performance of NACH-MM versus the optimal threshold policy when the number of processors are high compared to the number of job families.

Each of the three policies are compared against NACH-MM. Table A- 1 contains the various system parameters varied. Corr (X, Y) refers to the correlation coefficient between the job families of two consecutive job arrivals X and Y. Correlation is introduced in the same manner as in Section 3.4.2.2.2.

TABLE A- 1: VALUES GIVEN TO VARIOUS SYSTEM PARAMETERS FOR COMPARING POLICIES.

No. of processors	No. of job families	Traffic intensity	Corr (X, Y)
2, 4	4, 8	0.5, 0.8	0, 0.25, 0.7

The batch processor size is fixed at six jobs and the processing time of a batch is evenly distributed between job families from three to nine time units. For each set of parameters, 2000 jobs are generated, with the mean time between arrivals dictated by the traffic intensity and having an exponential distribution. The cycle time of the first 200 jobs are discarded. This warm-up period is deemed to be sufficient using Welch's method ([Law and Kelton 2000]). The paired T-test is used to determine whether the mean cycle times of two policies are significantly different from each other for a given set of system parameters.

A.5.3 Discussion of Simulation Results

Table A- 2 contains summary data for the comparison performed between Policy one and NACH-MM. The percentage difference column is the percentage difference of the cycle time from Policy one, relative to the cycle time of NACH-MM, and only has a value if the estimated difference between the two policies is significantly different from zero at 95% confidence level. A negative value in the percentage difference means that Policy One has lower mean cycle time than NACH-MM. The same information is presented in Figure A- 1.

TABLE A- 2: SUMMARY OF DATA COMPARING POLICY ONE AND NACH-MM

Traffic intensity	Number of processors	Number of job families	*Corr(X,Y)*	Policy one Mean cycle time	NACH-MM Mean cycle time	% difference
0.5	2	4	0	10.024	10.768	-6.910%
0.5	2	8	0	16.228	16.769	-3.232%
0.5	4	4	0	8.293	8.437	-1.695%
0.5	4	8	0	11.058	11.349	-2.564%
0.8	2	4	0	11.598	12.287	-5.608%
0.8	2	8	0	20.406	18.195	12.152%
0.8	4	4	0	9.061	9.235	-1.895%
0.8	4	8	0	13.224	12.158	8.768%
0.5	2	4	0.25	10.358	10.176	1.779%
0.5	2	8	0.25	14.559	14.852	-1.973%
0.5	4	4	0.25	8.090	8.142	-0.639%
0.5	4	8	0.25	10.358	10.434	-0.738%
0.8	2	4	0.25	11.631	12.202	-4.680%
0.8	2	8	0.25	19.452	17.732	9.700%
0.8	4	4	0.25	9.000	9.057	
0.8	4	8	0.25	12.699	11.826	7.382%
0.5	2	4	0.7	9.067	8.631	5.051%
0.5	2	8	0.7	10.874	9.921	9.606%
0.5	4	4	0.7	7.828	7.527	3.999%
0.5	4	8	0.7	8.579	8.065	6.373%
0.8	2	4	0.7	11.268	11.159	0.986%
0.8	2	8	0.7	15.942	15.132	5.353%
0.8	4	4	0.7	9.207	8.870	3.799%
0.8	4	8	0.7	10.946	10.499	4.267%

When the correlation coefficient is zero, Policy One results in slightly lower cycle time than NACH-MM, if the number of job families considered by the MPC-based kernel is as large as the number of job families being processed. The relative magnitudes of cycle time reduction is noticeably lower than the relative magnitudes of cycle time reduction obtained when there is only one processor (seen in Section 3.4.2.2.1). When the number of job families in the system is higher than the number of job families considered by the kernel and the traffic intensity is high, the performance of Policy One is significantly worse than that of NACH-MM.

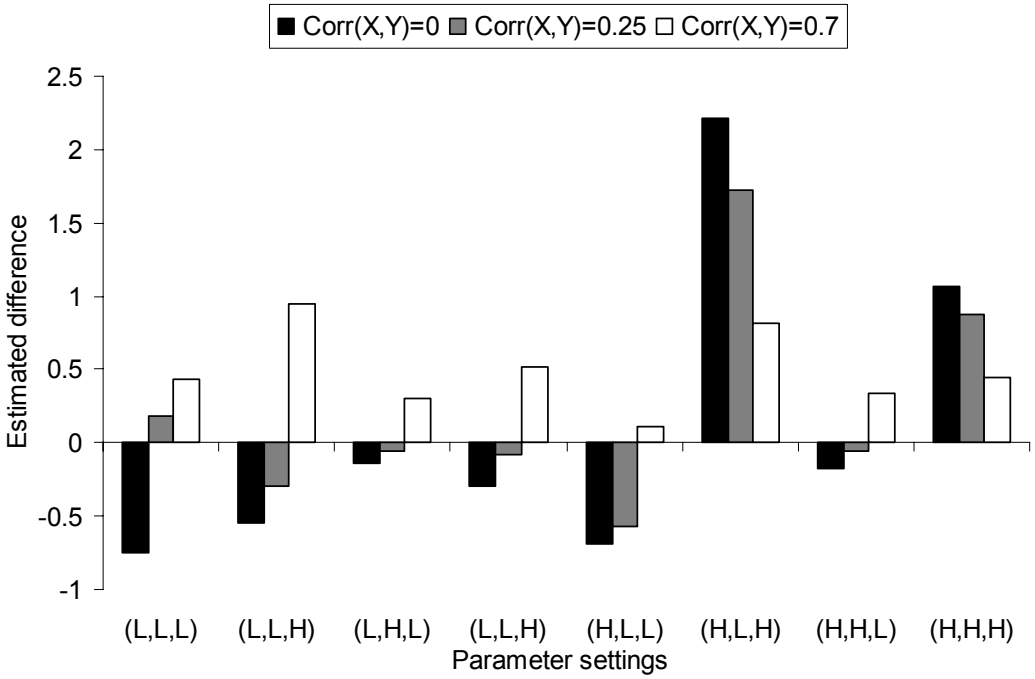


Figure A- 1: The difference between mean values under Policy One and NACH-MM.

A negative value means that Policy One has lower cycle time than NACH-MM. The parameter settings are described by the triplet (X,Y,Z), where X refers to traffic intensity, Y refers to the number of processors and Z is the number of families. L refers to a low setting and H refers to a high setting. This aliasing is used throughout the figures in Appendix A.

When there is only a single processor, the hypothesized major source of improvement of the MPC-based heuristic over NACHM is that the MPC-based heuristic optimizes the sequence of batches until all jobs within its horizon are depleted, while NACHM only optimizes the first batch. While this distinction is also true for Policy One and NACH-MM, Policy One assumes there is only one batch processor present. Thus, the time a batch needs to wait before it starts processing can be over-estimated, since it is possible that other processors become idle before the current processor. For this reason, the relative performance improvement of Policy One over NACH-MM is smaller than the relative improvement of the MPC-based heuristic over NACHM. Furthermore, the relative improvement of Policy One is insufficient to counter the deterioration that occurs when some job families are ignored by kernel.

When positive correlation between the job families of successive job arrivals is introduced, the mean cycle time is generally reduced, regardless of the policy used. We support this assertion by using a two-sample T-test to compare the cycle time distributions of systems under

identical policies, but with different correlation coefficients between the job families of two consecutive job arrivals.

Table A- 3 contains the summary of comparing identical systems under NACH-MM when the correlation is increased, Table A- 4 contains the same information, with the system under Policy One, Table A- 5 is for the system under Policy Two and Table A- 6 is for the system under Policy Three. Each table has the estimated difference in mean cycle time, the 95% confidence level and the percentage reduction due to increased correlation.

TABLE A- 3: EFFECT OF INCREASED CORRELATION ON PERFORMANCE OF NACH-MM

μ_0	$\mu_{0.25}$	$\mu_{0.7}$	Estimated difference (μ_0 - $\mu_{0.25}$)	Estimated difference ($\mu_{0.25}$ - $\mu_{0.7}$)
System parameters: 8 job families, 4 processors, 0.5 traffic intensity				
11.05	10.38	8.63	Mean: 0.667 95% CI: (0.591,0.744) % reduction: 6.04%	Mean: 1.754 95% CI: (1.650,1.858) % reduction: 16.90%
System parameters: 8 job families, 2 processors, TI = 0.5				
16.27	14.54	10.92	Mean: 1.731 95% CI: (1.535,1.926) % reduction: 10.64%	Mean: 3.62 95% CI: (3.444, 3.796) % reduction: 24.90%
System parameters: 8 job families, 4 processors, 0.8 traffic intensity				
13.21	12.84	11.09	Mean: 0.370 95% CI: (0.180,0.559) % reduction: 2.80%	Mean: 1.758 95% CI: (1.537, 1.979) % reduction: 13.69%
System parameters: 8 job families, 2 processors, 0.8 traffic intensity				
20.53	19.40	15.75	Mean: 1.126 95% CI: (0.831,1.421) % reduction: 5.48%	Mean: 3.652 95% CI: (3.236, 4.070) % reduction: 18.82%
System parameters: 4 job families, 4 processors, 0.5 traffic intensity				
8.23	8.08	7.85	Mean: 0.150 95% CI: (0.109,0.191) % reduction: 1.82%	Mean: 0.237 95% CI: (0.176, 0.298) % reduction: 2.93%
System parameters: 4 job families, 2 processors, 0.5 traffic intensity				
9.94	9.87	9.00	Not significant	Mean: 0.867 95% CI: (0.694, 1.040) % reduction: 8.78%
System parameters: 4 job families, 4 processors, 0.8 traffic intensity				
9.05	8.93	9.11	Mean: 0.120 95% CI: (0.024,0.215) % reduction: 1.33%	Mean: -0.183 95% CI: (-0.334,-0.032) % reduction: -2.05%
System parameters: 4 job families, 2 processors, 0.8 traffic intensity				
11.47	11.57	11.36	Not significant	Not significant

Data from Table A- 3 is also presented in graphical form in Figure A- 2 and shows that increasing the correlation coefficient generally leads to a statistically significant reduction in the mean cycle time when the system is under NACH-MM. There are exceptions; when all parameter settings are low, an initial increase in correlation did not result in statistically significant reduction in the mean cycle time, although further increases did result in

statistically significant reductions in mean cycle time. Furthermore, increasing correlation does not cause a significant effect in the mean cycle time when the traffic intensity is high and both the number of processors and the number of job families are low. Under these circumstances, the time till the next arrival of a particular job family is short, and it typically pays to wait for the next job arrival, regardless of the level of correlation in future job arrivals.

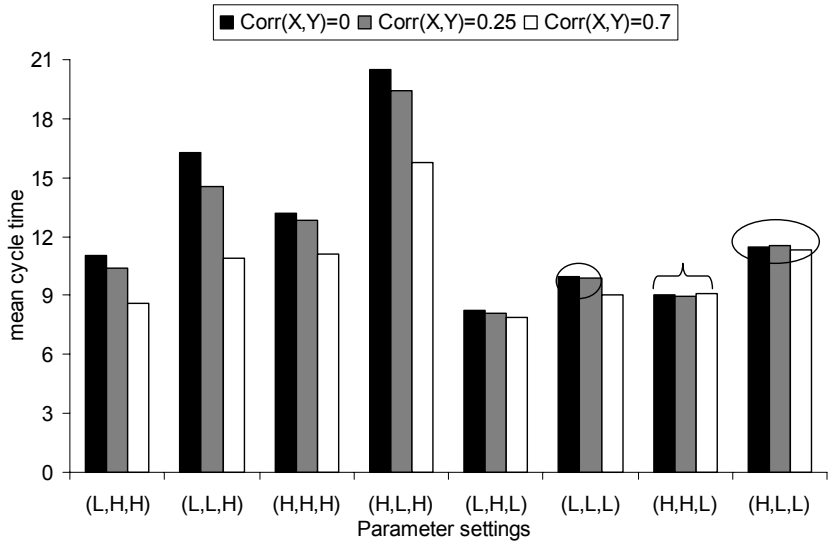


Figure A- 2: Mean cycle time values for NACH-MM.

Groups of graphs marked with ovals have mean cycle time values that cannot be significantly differentiated from each other. As the correlation between two successive job arrivals are increased, the mean cycle time of the job is typically reduced. Bracket marks instance where initial cycle time reduction is eventually counteracted with cycle time increase as correlation is further increased.

When the number of job families is small, the estimated cycle time reduction due to increasing correlation is lower. When the number of processors is increased, two key values change: the mean amount of time till a processor becomes free and the mean time between arrivals is reduced. Since NACH-MM uses the mean time till the next processor is idle to calculate the cost of waiting for future jobs to arrive, the increase in number of processors reduces the cost of waiting for future jobs to arrive, encouraging NACH-MM to wait for more job arrivals. This condition is exacerbated when job families of future job arrivals are positively correlated; after waiting for a job arrival from Family One, the processor will tend to wait for the next job arrival, also from Family One.

TABLE A- 4: EFFECT OF INCREASED CORRELATION ON PERFORMANCE OF POLICY ONE

μ_0	$\mu_{0.25}$	$\mu_{0.7}$	Estimated difference (μ_0 - $\mu_{0.25}$)	Estimated difference ($\mu_{0.25}$ - $\mu_{0.7}$)
System parameters: 8 job families, 4 processors, 0.5 traffic intensity				
11.35	10.43	8.07	Mean: 0.915 95% CI: (0.800,1.029) % reduction: 8.06%	Mean: 2.337 95% CI: (2.284,2.455) % reduction: 22.41%
System parameters: 8 job families, 2 processors, 0.5 traffic intensity				
16.77	14.85	9.92	Mean: 1.917 95% CI: (1.723,2.112) % reduction: 11.43%	Mean: 4.931 95% CI: (4.744,5.117) % reduction: 33.21%
System parameters: 8 job families, 4 processors, 0.8 traffic intensity				
12.16	11.83	10.50	Mean: 0.332 95% CI: (0.228,0.436) % reduction: 2.73%	Mean: 1.327 95% CI: (1.092,1.562) % reduction: 11.22%
System parameters: 8 job families, 2 processors, 0.8 traffic intensity				
18.19	17.73	15.13	Mean: 0.463 95% CI: (0.221,0.705) % reduction: 2.55%	Mean: 2.600 95% CI: (2.225,2.975) % reduction: 14.66%
System parameters: 4 job families, 4 processors, 0.5 traffic intensity				
8.44	8.14	7.53	Mean: 0.295 95% CI: (0.235,0.354) % reduction: 3.50%	Mean: 0.615 95% CI: (0.535,0.694) % reduction: 7.56%
System parameters: 4 job families, 2 processors, 0.5 traffic intensity				
10.77	10.18	8.63	Mean: 0.591 95% CI: (0.495,0.688) % reduction: 5.49%	Mean: 1.545 95% CI: (1.432,1.658) % reduction: 15.18%
System parameters: 4 job families, 4 processors, 0.8 traffic intensity				
9.24	9.06	8.87	Mean: 0.178 95% CI: (0.074,0.282) % reduction: 1.93%	Not significant
System parameters: 4 job families, 2 processors, 0.8 traffic intensity				
12.29	12.20	11.16	Not significant	Mean: 1.044 95% CI: (0.775,1.312) % reduction: 8.56%

For Policies One, Two and Three, only ten data points are available. Consequently, the probability is larger that an actual significant difference will go undetected for Policies One, Two and Three. Nevertheless, the general trends observed from NACH-MM (Table A- 3) also hold true for Policies One, Two and Three (Table A- 4, Table A- 5 and Table A- 6). The amount of cycle time reduction when correlation is increased decreases when: the number of processors increase, the number of job families decrease or when the time between arrivals decrease. Furthermore, there is little evidence for an increase in correlation causing an increase in mean cycle time for Policies One, Two and Three, unlike NACH_MM.

TABLE A- 5: EFFECT OF INCREASED CORRELATION ON PERFORMANCE OF POLICY TWO

μ_0	$\mu_{0.25}$	$\mu_{0.7}$	Estimated difference (μ_0 - $\mu_{0.25}$)	Estimated difference ($\mu_{0.25}$ - $\mu_{0.7}$)
System parameters: 8 job families, 4 processors, 0.5 traffic intensity				
11.35	10.37	8.15	Mean: 0.977 95% CI: (0.808,1.146) % reduction: 8.61%	Mean: 2.221 95% CI: (1.979,2.463) % reduction: 21.42%
System parameters: 8 job families, 2 processors, 0.5 traffic intensity				
16.90	15.00	9.94	Mean: 1.900 95% CI: (1.503,2.297) % reduction: 11.24%	Mean: 5.056 95% CI: (4.714,5.399) % reduction: 33.71%
System parameters: 8 job families, 4 processors, 0.8 traffic intensity				
12.12	11.98	10.66	Not significant	Mean: 1.321 95% CI: (0.992,1.649) % reduction: 11.03%
System parameters: 8 job families, 2 processors, 0.8 traffic intensity				
18.19	17.55	15.09	Mean: 0.635 95% CI: (0.312,0.958) % reduction: 3.49%	Mean: 2.462 95% CI: (1.663,3.261) % reduction: 14.03%
System parameters: 4 job families, 4 processors, 0.5 traffic intensity				
8.45	8.18	7.36	Mean: 0.268 95% CI: (0.176,0.360) % reduction: 3.17%	Mean: 0.826 95% CI: (0.715,0.936) % reduction: 11.22%
System parameters: 4 job families, 2 processors, 0.5 traffic intensity				
10.79	10.14	8.73	Mean: 0.658 95% CI: (0.472,0.843) % reduction: 6.10%	Mean: 1.408 95% CI: (1.223,1.592) % reduction: 13.89%
System parameters: 4 job families, 4 processors, 0.8 traffic intensity				
9.27	9.09	8.73	Mean: 0.187 95% CI: (0.025,0.349) % reduction: 2.02%	Mean: 0.357 95% CI: (0.127,0.587) % reduction: 3.93%
System parameters: 4 job families, 2 processors, 0.8 traffic intensity				
12.22	12.08	9.033	Not significant	Mean: 3.050 95% CI: (2.707,3.393) % reduction: 25.25%

From Table A- 2, the effect of introducing positive correlation to the relative performance of the Policy one against NACH-MM is dependent on the number of job families being processed by the system, compared against *F*, the number of job families considered by kernel. If the total number of job families is not greater than *F*, then no job family gets ignored by the MPC-based heuristic. In this case, the relative performance of NACH-MM improves at a faster rate than that of Policy One, as the correlation increases, and Policy One eventually becomes inferior to NACH-MM, given a sufficiently high correlation coefficient. When the number of job families is greater than *F*, some job families are disregarded by the MPC-based heuristic, resulting in Policy One having higher cycle time than NACH-MM, even when the correlation coefficient is zero. When positive correlation is introduced, it becomes more likely that majority of the WIP in front of the batch processing stage only belong to a small subset of job

families. Thus, in the presence of positive correlation, the penalty for ignoring a certain amount of job families in the decision making process is reduced. This causes the relative reduction in cycle time by NACH-MM over Policy One to be reduced as the correlation coefficient increases, when the number of job families is greater than *F*.

TABLE A- 6: EFFECT OF INCREASED CORRELATION ON PERFORMANCE OF POLICY THREE

μ_0	$\mu_{0.25}$	$\mu_{0.7}$	Estimated difference (μ_0 - $\mu_{0.25}$)	Estimated difference ($\mu_{0.25}$ - $\mu_{0.7}$)
System parameters: 8 job families, 4 processors, 0.5 traffic intensity				
11.29	10.55	7.93	Mean: 0.741 95% CI: (0.533,0.948) % reduction: 6.56%	Mean: 2.620 95% CI: (2.421,2.819) % reduction: 24.83%
System parameters: 8 job families, 2 processors, 0.5 traffic intensity				
16.85	14.61	10.07	Mean: 2.242 95% CI: (1.912,2.572) % reduction: 13.31%	Mean: 4.537 95% CI: (4.160,4.914) % reduction: 31.05%
System parameters: 8 job families, 4 processors, 0.8 traffic intensity				
12.06	11.92	10.52	Mean: 0.142 95% CI: (0.037,0.248) % reduction: 1.18%	Mean: 1.400 95% CI: (1.203,1.597) % reduction: 11.74%
System parameters: 8 job families, 2 processors, 0.8 traffic intensity				
18.27	17.65	15.13	Mena: 0.625 95% CI: (0.257,0.992) % reduction: 3.42%	Mean: 2.513 95% CI: (1.957,3.070) % reduction: 16.61%
System parameters: 4 job families, 4 processors, 0.5 traffic intensity				
8.41	8.16	7.43	Mean: 0.250 95% CI: (0.148,0.352) % reduction: 2.97%	Mean: 0.725 95% CI: (0.539,0.910) % reduction: 8.88%
System parameters: 4 job families, 2 processors, 0.5 traffic intensity				
10.76	10.31	8.48	Mean: 0.450 95% CI: (0.318,0.581) % reduction: 4.18%	Mean: 1.834 95% CI: (1.595,2.072) % reduction: 17.79%
System parameters: 4 job families, 4 processors, 0.8 traffic intensity				
9.24	9.01	8.80	Mean: 0.227 95% CI: (0.082,0.371) % reduction: 2.46%	Not significant
System parameters: 4 job families, 2 processors, 0.8 traffic intensity				
12.18	12.12	11.28	Not significant	Mean: 0.837 95% CI: (0.461,1.212) % reduction: 6.91%

The observations made from data comparing Policies Two and Three with NACH-MM are similar to those made in comparing Policy One and NACH-MM, and are no longer repeated. Table A- 7 contains the comparison results for Policy Two and NACH-MM, while Table A- 8 contains the comparison results for Policy Three and NACH-MM. When a significant difference cannot be detected, the "% difference" column is left blank.

TABLE A- 7: SUMMARY OF DATA COMPARING POLICY TWO AND NACH-MM

Traffic Intensity	Number of processors	Number of families	Corr(X,Y)	Policy two Mean cycle time	NACH-MM Mean cycle time	% difference
0.5	2	4	0	9.901	10.793	-8.265%
0.5	2	8	0	16.360	16.898	-3.184%
0.5	4	4	0	8.216	8.451	-2.781%
0.5	4	8	0	11.015	11.347	-2.926%
0.8	2	4	0	11.403	12.216	-6.655%
0.8	2	8	0	20.472	18.188	12.558%
0.8	4	4	0	9.030	9.272	-2.610%
0.8	4	8	0	13.227	12.122	9.116%
0.5	2	4	0.25	9.552	10.135	-5.752%
0.5	2	8	0.25	14.527	14.997	-3.134%
0.5	4	4	0.25	8.081	8.183	-1.246%
0.5	4	8	0.25	10.347	10.370	
0.8	2	4	0.25	11.508	12.083	-4.767%
0.8	2	8	0.25	19.351	17.554	10.237%
0.8	4	4	0.25	8.964	9.085	-1.332%
0.8	4	8	0.25	13.047	11.984	8.879%
0.5	2	4	0.7	9.139	8.727	4.709%
0.5	2	8	0.7	10.966	9.940	10.317%
0.5	4	4	0.7	7.800	7.357	6.021%
0.5	4	8	0.7	8.802	8.149	8.013%
0.8	2	4	0.7	11.431	9.033	26.547%
0.8	2	8	0.7	15.291	15.092	
0.8	4	4	0.7	9.033	8.728	3.494%
0.8	4	8	0.7	11.287	10.663	5.862%

TABLE A- 8: SUMMARY OF DATA COMPARING POLICY THREE AND NACH-MM

Traffic Intensity	Number of processors	Number of families	Corr(X,Y)	Policy three Mean cycle time	NACH-MM Mean cycle time	% difference
0.5	2	4	0	9.894	10.761	-8.048%
0.5	2	8	0	16.230	16.852	-3.685%
0.5	4	4	0	8.189	8.409	-2.604%
0.5	4	8	0	11.077	11.290	-1.887%
0.8	2	4	0	11.398	12.177	-6.397%
0.8	2	8	0	20.704	18.271	13.316%
0.8	4	4	0	9.056	9.242	-2.002%
0.8	4	8	0	13.190	12.059	9.379%
0.5	2	4	0.25	9.697	10.311	-5.955%
0.5	2	8	0.25	14.541	14.610	
0.5	4	4	0.25	8.076	8.159	-1.017%
0.5	4	8	0.25	10.443	10.549	
0.8	2	4	0.25	11.587	12.121	-4.405%
0.8	2	8	0.25	19.401	17.646	9.945%
0.8	4	4	0.25	8.824	9.015	-2.119%
0.8	4	8	0.25	12.786	11.916	7.301%
0.5	2	4	0.7	8.800	8.478	3.810%
0.5	2	8	0.7	10.927	10.073	8.478%
0.5	4	4	0.7	7.909	7.434	6.389%
0.5	4	8	0.7	8.505	7.929	7.265%
0.8	2	4	0.7	11.387	11.285	
0.8	2	8	0.7	16.014	15.133	5.822%
0.8	4	4	0.7	9.096	8.804	3.317%
0.8	4	8	0.7	11.024	10.516	4.831%

To determine whether there is a superior policy between Policies One, Two and Three, we perform an additional set of simulation experiments directly comparing the performance of the three policies. Two levels for the number of job families (four and eight), two levels for the number of processors (two and four) and two levels for the traffic intensity (0.5 and 0.8) were selected. For each setting, ten simulation runs lasting for 2000 jobs are performed, and data from the first 200 jobs are discarded. The MPC-based heuristic used for all three policies is (4, 10), which means four job families and the next 10 job arrivals are considered.

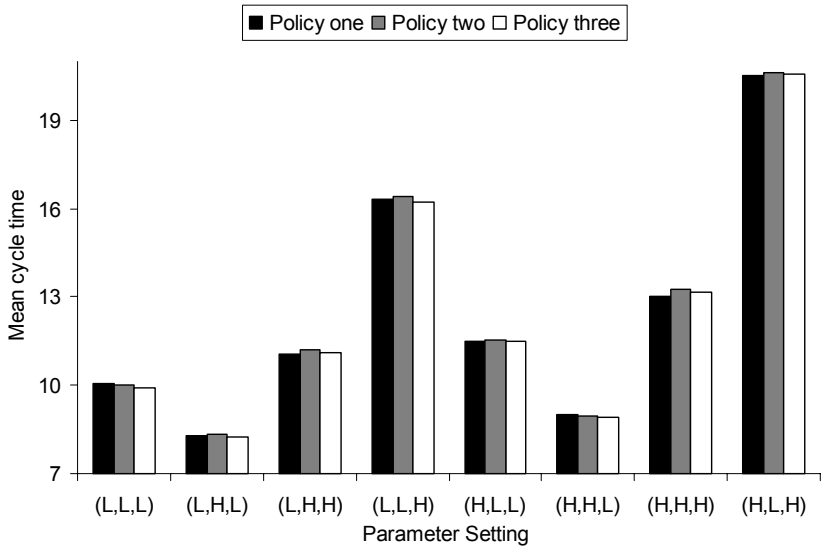


Figure A- 3: Comparing the mean cycle time for the three policies when the job families of successive job arrivals are uncorrelated.

All three policies have similar mean cycle times, although Policy two seems to always be inferior to Policy three, through visual inspection.

Since the differences between the three policies only occur when two or more processors are simultaneously idle, we expect the differences to be subtle. Consequently, we anticipate that there will be only a few settings for which a significant difference between the performances of the policies can be detected, with a paired T-test. Figure A- 3 plots the mean cycle times for all three policies. Furthermore, we expect differences to be more pronounced when the number of processors is high, since there is a greater probability of having more than one processor simultaneously idle. When the traffic intensity is low, the probability of finding idle processors is also higher, which increases the probability of detecting a significant difference, if it exists.

Table A- 9 contains the settings for which a significant difference is detected, including the confidence interval for the hypothesized difference in means. All of the settings for which a significant difference is detected have either low traffic intensity, or high number of processors. From Table A- 9, when the kernel does not discard any job families from consideration, Policy One is never significantly better than either Policy Two or Three. This suggests that using the weighted shortest processing time (WSPT) rule when more than one processor is simultaneously idle is not a good heuristic, compared to variants of the MPC-based kernel. Furthermore, Policy Two is also determined to have significantly worse performance than Policy Three when both the number of job families and the traffic intensity are low. At low traffic intensities, continuously processing jobs without waiting for additional job arrivals can result in high cycle times.

TABLE A- 9: SETTINGS WHERE POLICIES ONE, TWO AND THREE HAVE SIGNIFICANTLY DIFFERENT PERFORMANCE

Traffic intensity	Number of processors	Number of job families	Policies for which significant difference is detected between policies	95% confidence interval
0.5	2	4	Policy One (μ = 10.045) has higher cycle time than Policy Three (μ = 9.902)	(0.083, 0.127)
0.5	2	4	Policy Two (μ = 10.029) has higher cycle time than Policy Three (μ = 9.902)	(0.077, 0.177)
0.5	4	4	Policy One (μ = 8.288) has higher cycle time than Policy Three (μ = 8.225)	(0.008, 0.117)
0.5	4	4	Policy Two (μ = 8.316) has higher cycle time than Policy Three (μ = 8.225)	(0.030, 0.152)
0.5	4	8	Policy Two (μ = 11.214) has higher cycle time than Policy One (μ = 11.044)	(0.055, 0.285)
0.8	4	4	Policy One (μ = 9.002) has higher cycle time than Policy Three (μ = 8.927)	(0.022, 0.128)
0.8	4	8	Policy Two (μ = 13.269) has higher cycle time than Policy One (μ = 13.044)	(0.225, 0.381)

When the number of job families is high compared to the number of job families considered by the MPC-based heuristic, Policy One can perform better than Policy Two. This is primarily due to Policy Two (and Policy Three) considering only half the number of job families in determining the appropriate action for the *k* idle processors at a particular time instance. In contrast, Policy One considers all job families in determining the appropriate action for *k - 1*

idle processors, and only needs to discard some job families for the last idle processor. This additional discarding of job families results in deterioration in the performance of Policy Two (and possibly Policy Three).

A.5.4 Conclusion

In conclusion, we extend earlier analysis of the problem of minimizing the mean cycle time of a batch processing stage containing a single batch processor with incompatible job families and future job arrivals to the case where the stage contains multiple batch processors in parallel. This problem is strongly NP-Hard. Two formulations are provided for solving the finite horizon version of the problem: an integer liner programming formulation and a dynamic programming formulation. Both formulations can be used to solve small problem instances.

When the problem input size is large, we propose three different heuristics, each with a common kernel. This kernel is the MPC-based heuristic previously shown to have superior performance over a popular look-ahead when there is only one batch processor at the stage. We determine Policies One to Three slightly outperform NACH-MM when the job families of future arrivals are uncorrelated and the number of job families is not higher than the number of job families considered by the underlying MPC-based heuristic. However, the performance deterioration that occurs when Policies One to Three are forced to ignore some job families can cause Policies One to Three to have worse performance than NACH-MM when the number of job families is high.

Similar to the case where there is only a single processor at the stage, the addition of positive correlation into the job families of future job arrivals generally causes the performance of the system to improve, regardless of policy. This reduction decreases when the number of processors increases, the number of job families decreases or the time between arrivals decrease. Thus, when we have a high number of processors, high traffic intensity, a low number of job families and the system is under NACH-MM, the initial reduction in cycle time

due to a moderate increase in correlation is later offset by a subsequent cycle time increase when the correlation coefficient is further increased.

Similar to the case where there is only one batch processor, the results suggest that two ways cycle time can be reduced. We can use a heuristic that has better mean performance, which may require much larger computational costs. We can also control the upstream processors, such that the arrival distribution is manipulated to benefit the batch processing stage. This strategy also allows us to use a much simpler heuristic at the batch processing stage, and the combination of increased correlation and a simple heuristic is shown to be superior to that of increased correlation and complicated heuristic.

Appendix B

Simulation Results for Evaluating MPC-Based Heuristic Performance

This section contains the mean values for each simulation run, for each heuristic. The simulation experiment parameters can be found in Section 3.4.2.

B.1 Experiment 1: System has deterministic arrival

TABLE B- 1: MEAN CYCLE TIME VALUES OF HEURISTICS UNDER DETERMINISTIC ARRIVAL TIMES

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	31.3	29.3	30.1	27.6	28.3	29.2	27.6	30.6	27.5	28.1
		(4,10)	29.8	28.0	29.5	27.3	28.5	29.8	27.2	31.0	27.5	28.2
	8	NACHM	31.3	30.3	30.6	29.6	29.6	30.1	31.0	30.8	29.6	30.3
		(3,15)	16.8	18.2	17.8	19.1	17.3	17.9	18.4	17.5	17.3	18.2
		(4,10)	15.7	16.9	16.7	17.4	15.8	16.3	16.8	16.2	16.2	16.6
0.80	4	NACHM	18.0	18.8	18.3	19.0	18.5	18.8	19.2	18.4	18.4	19.0
		(3,15)	23.8	24.3	23.2	23.1	23.7	23.2	23.4	23.1	23.1	24.2
		(4,10)	23.6	24.0	23.9	23.6	23.9	23.4	23.9	23.1	23.6	23.4
	8	NACHM	29.1	28.2	28.4	27.6	26.6	28.1	28.9	28.0	28.0	29.2
		(3,15)	15.3	15.6	15.5	15.4	15.3	15.3	15.7	15.3	15.3	15.3
		(4,10)	13.4	13.7	13.5	13.4	13.1	13.5	13.8	13.5	13.3	13.1
0.50	4	NACHM	15.3	15.7	15.7	15.7	15.5	15.4	15.8	15.7	15.7	15.5

B.2 Experiment 2: System has positive correlation between job families of consecutive job arrivals

In this section, *Corr(X,Y)* is the correlation coefficient between the job families of two consecutive job arrivals.

TABLE B- 2: MEAN CYCLE TIME VALUES OF HEURISTICS WHEN CORR(X,Y) = 0.25

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	27.0	27.6	29.2	29.2	27.2	28.6	29.1	26.6	25.9	28.1
		(4,10)	26.9	29.2	28.9	28.7	27.7	28.4	26.9	26.4	25.7	28.3
	8	NACHM	29.6	30.0	31.3	29.8	29.5	29.4	29.4	29.1	28.9	30.0
		(3,15)	18.6	16.5	16.7	18.6	17.9	17.2	18.2	18.7	17.1	17.8
		(4,10)	16.9	15.2	15.2	16.8	16.7	15.8	16.9	17.1	15.6	16.3
0.80	4	NACHM	18.4	17.3	17.3	18.6	18.2	17.6	19.1	19.2	17.6	18.0
		(3,15)	22.2	21.8	20.8	21.3	21.8	21.1	21.2	21.2	21.3	21.0
		(4,10)	21.4	22.6	20.7	21.0	21.1	21.4	21.9	20.5	21.4	21.4
	8	NACHM	24.9	25.1	23.2	23.2	23.7	24.4	25.2	23.9	24.4	23.9
		(3,15)	14.4	14.3	14.6	14.1	14.4	14.3	14.6	14.6	14.3	15.0
		(4,10)	12.7	12.2	12.4	12.6	12.6	12.5	12.6	12.5	12.5	12.7
0.50	4	NACHM	14.3	14.0	13.8	14.3	14.2	14.2	14.0	14.4	14.1	13.7

TABLE B- 3: MEAN CYCLE TIME VALUES OF HEURISTICS WHEN CORR(X,Y) = 0.7

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	22.8	21.7	23.5	25.1	22.3	25.8	22.5	21.9	21.9	22.4
		(4,10)	24.1	21.7	23.6	25.1	22.1	26.6	23.0	21.0	23.0	23.3
	8	NACHM	25.8	25.3	24.6	27.3	22.9	26.2	25.0	24.8	22.5	23.9
		(3,15)	15.3	16.3	17.2	17.0	15.5	15.7	14.7	14.7	16.8	15.8
		(4,10)	14.9	15.5	16.1	16.4	14.7	14.6	13.5	14.7	15.9	14.6
0.80	4	NACHM	15.9	16.1	16.8	17.6	15.4	16.4	14.4	16.0	16.9	16.1
		(3,15)	15.7	15.5	14.8	14.7	16.4	15.1	15.8	16.0	15.1	15.3
		(4,10)	15.6	15.7	14.9	14.6	16.7	16.2	16.1	16.6	15.7	15.5
	8	NACHM	14.8	14.4	14.0	13.2	15.8	14.0	14.7	14.6	14.9	14.4
		(3,15)	11.9	12.0	12.8	11.8	12.5	11.9	12.8	11.8	12.5	11.8
		(4,10)	11.1	11.5	11.7	11.0	11.5	11.3	11.4	10.9	11.7	11.1
0.50	4	NACHM	10.8	11.1	11.7	10.9	10.9	11.2	11.3	10.9	11.4	10.9

B.3 Experiment 3: System has imperfect prediction of arrival times

In this section, Ω is the error coefficient (equivalent ot the squared coefficient of variation) experienced by the system in its predicted inter-arrival time.

TABLE B- 4: MEAN CYCLE TIME VALUES OF THE HEURISTICS WHEN Ω = 0.1.

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	29.6	29.2	28.3	28.8	29.6	28.6	29.4	27.9	35.5	28.6
		(4,10)	28.9	29.0	28.0	28.3	29.4	27.5	29.2	28.1	32.7	29.6
	8	NACHM	29.7	30.8	29.5	30.1	30.0	28.6	30.6	30.4	31.8	30.4
		(3,15)	18.8	17.9	19.6	18.0	17.6	17.4	18.6	17.9	18.0	17.2
		(4,10)	16.4	16.4	17.7	16.7	16.1	16.4	17.1	16.2	16.6	16.0
0.80	4	NACHM	18.3	19.3	19.1	18.8	18.0	18.4	19.6	18.8	19.0	18.2
		(3,15)	23.8	23.3	23.2	23.6	23.5	23.2	23.9	23.4	23.1	23.3
		(4,10)	23.2	23.6	23.4	23.7	23.8	23.4	23.5	24.2	23.2	23.6
	8	NACHM	27.9	27.9	28.0	28.9	28.1	26.9	28.6	29.0	26.9	27.9
		(3,15)	15.3	15.5	15.7	15.5	15.3	15.6	15.7	15.5	14.9	15.1
		(4,10)	13.4	13.7	13.5	13.4	13.1	13.5	13.5	13.4	13.0	13.4
0.50	4	NACHM	15.4	15.7	15.7	15.6	15.0	15.6	15.9	15.6	15.0	15.6

TABLE B- 5: MEAN CYCLE TIME VALUES OF THE HEURISTICS Ω = 0.3.

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	29.7	30.5	28.6	29.2	28.3	28.4	30.5	28.0	31.3	28.0
		(4,10)	28.7	29.2	27.4	27.7	28.0	28.9	29.6	27.5	30.0	28.7
	8	NACHM	30.3	30.1	30.6	30.9	30.1	29.9	30.0	29.3	30.8	29.1
		(3,15)	18.3	17.8	19.0	17.9	17.9	18.2	18.7	18.3	18.1	17.6
		(4,10)	16.5	16.2	17.3	16.6	16.2	16.2	17.3	16.5	17.0	16.0
0.80	4	NACHM	18.5	18.4	19.7	18.8	18.3	18.5	19.6	18.8	18.5	18.9
		(3,15)	23.3	24.0	23.1	24.0	23.8	23.5	23.4	23.8	22.8	23.6
		(4,10)	24.1	23.9	23.2	24.0	24.0	23.1	23.4	23.4	23.1	23.9
	8	NACHM	29.1	28.0	27.5	28.1	27.9	27.7	28.3	28.6	27.6	28.2
		(3,15)	15.6	15.8	15.8	15.5	15.5	15.8	15.6	15.9	15.4	15.6
		(4,10)	13.4	13.7	13.4	13.4	13.1	13.4	13.6	13.4	13.1	13.5
0.50	4	NACHM	15.8	16.1	15.8	15.6	15.6	15.6	15.7	15.6	15.4	15.5

TABLE B- 6: MEAN CYCLE TIME VALUES OF THE HEURISTICS WHEN $\Omega = 0.5$.

Traffic intensity	No. of job families	Heuristic	Run Number									
			1	2	3	4	5	6	7	8	9	10
		(3,15)	29.4	32.1	28.1	28.6	29.7	29.6	30.9	28.2	36.0	29.2
		(4,10)	28.9	28.8	27.8	28.7	29.1	27.3	29.6	28.0	31.0	29.5
	8	NACHM	30.2	30.8	30.0	30.5	30.1	30.3	30.4	30.0	31.1	29.6
		(3,15)	19.0	18.0	19.9	18.2	17.7	17.9	19.2	18.4	19.1	17.3
		(4,10)	16.5	16.3	17.4	16.4	16.3	16.4	17.3	16.6	16.8	15.9
0.80	4	NACHM	18.9	19.1	19.4	19.2	18.0	18.7	19.4	18.6	19.1	18.5
		(3,15)	23.8	24.3	23.7	23.6	24.2	23.4	23.9	23.7	23.4	23.9
		(4,10)	24.3	23.8	23.2	23.7	23.7	23.3	23.7	23.2	23.2	23.5
	8	NACHM	28.7	27.7	28.7	29.2	28.6	27.6	28.8	28.6	26.9	27.7
		(3,15)	15.6	15.9	15.9	15.5	15.8	15.9	16.1	16.0	15.6	15.7
		(4,10)	13.3	13.6	13.4	13.4	13.2	13.4	13.6	13.3	13.1	13.5
0.50	4	NACHM	15.8	16.3	15.4	15.6	15.2	16.2	16.0	15.6	15.3	16.1

Appendix C

Transition probability equations for two-stage models, with the serial processor under a threshold policy

This section contains the state space and probability transition equations for Models One to Three of Chapter 4, when the serial processor is under a threshold policy. Throughout this section, it is assumed that the threshold values are always greater than one full batch. Assuming that the state space is n states, the steady state probabilities can be obtained in $O(n^3)$. (see [Gershwin 2002], for example, for details in obtaining the steady state probability distribution given the transition probabilities)

C.1 Model One: Serial processor processing time is deterministic, batch processor processing time is geometric

The notation used is identical to what is seen in Section 4.6. L refers to the threshold value. The serial processor continues processing until $Z_t = L$. When the batch processor is under no-idling policy, the state space is composed of $L + 2$ states. These are $(0, 0)$ and $(Z_t, 1)$ where Z_t is from 0 to L.

TABLE C- 1: MODEL ONE TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER NO-IDLING POLICY

Initial State	Final State	Conditions for transition	Transition probability
(Z_t, S_t)	$(Z_t + 1, S_t)$	$Z_t < L$ and $S_t = 1$	$(1-P_B)$
$(0, 0)$	$(0, 1)$		1
(Z_t, S_t)	$(Min \{0, Z_t + 1 – Q\}, S_t)$	$Z_t < L$ and $S_t = 1$	P_B
(Z_t, S_t)	(Z_t, S_t)	$Z_t = L$ and $S_t = 1$	$(1-P_B)$
(Z_t, S_t)	$(Z_t – Q, S_t)$	$Z_t = L$ and $S_t = 1$	P_B

When the batch processor is under no-idling policy, the state space is composed of $L + Q + 1$ states. These are $(Z_t, 0)$, where Z_t is from 0 to $Q - 1$ and $(Z_t, 1)$ where Z_t is from 0 to L.

TABLE C- 2: MODEL ONE TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER FULL-BATCH POLICY

Initial State	Final State	Conditions for transition	Transition probability
(Z_t, S_t)	$(Z_t + 1, S_t)$	$Z_t < L$ and $S_t = 1$	$(1-P_B)$
(Z_t, S_t)	$(Z_t + 1, 0)$	$Z_t < Q-1$ and $S_t = 1$	P_B
$(Z_t, 0)$	$(Z_t + 1, 0)$	$Z_t < Q-1$	1
$(Z_t, 0)$	$(0, 1)$	$Z_t = Q-1$	1
(Z_t, S_t)	$(Z_t + 1 – Q, S_t)$	$Q - 1 < Z_t < L$ and $S_t = 1$	P_B
(Z_t, S_t)	(Z_t, S_t)	$Z_t = L$ and $S_t = 1$	$(1-P_B)$
(Z_t, S_t)	$(Z_t – Q, S_t)$	$Z_t = L$ and $S_t = 1$	P_B

C.2 Model Two: Processing time of both processors are geometric

The notation used is identical to what is seen in Section 4.7. L refers to the threshold value. The serial processor continues processing until $Z_t = L$. When the batch processor is under no-idling policy, the state space of the system is composed of $L + 2$ states:

- $(Z_t, 0, 1)$, where $Z_t = 0$. (The batch processor can only be idle if $Z_t = 0$).
- $(Z_t, 1, 0)$, where $Z_t = L$. (The serial processor can only be idle if $Z_t = L$).
- $(Z_t, 1, 1)$, where $Z_t = 0$ to $L – 1$

When the batch processor is under full-batch policy, the state space of the system is composed of $L + Q + 1$ states:

- $(Z_t, 0, 1)$, where $Z_t = 0$ to $Q - 1$. (The batch processor can only be idle if $Z_t < Q$).
- $(Z_t, 1, 0)$, where $Z_t = L$. (The serial processor can only be idle if $Z_t = L$).
- $(Z_t, 1, 1)$, where $Z_t = 0$ to $L - 1$

TABLE C- 3: MODEL TWO TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER NO-IDLING POLICY

Initial State	Final State	Conditions for transition	Transition probability
$(L, 1, 0)$	$(L - Q, 1, 1)$		P_B
$(L, 1, 0)$	$(L , 1, 0)$		$1 - P_B$
$(L - 1, 1, 1)$	$(L - 1 - Q, 1, 1)$		$P_B (1 - P_S)$
$(L - 1, 1, 1)$	$(L - Q, 1, 1)$		$P_B P_S$
$(L - 1, 1, 1)$	$(L, 1, 0)$		$(1 - P_B) P_S$
$(L - 1, 1, 1)$	$(L - Q, 1, 1)$		$P_B P_S$
$(Z_t, 1, 1)$	$(Z_t, 1, 1)$	$Z_t < L - 1$	$(1-P_B) (1-P_S)$
$(Z_t, 1, 1)$	$(Min \{Z_t - Q, 0\}, 1, 1)$	$0 < Z_t < L - 1$	$P_B (1-P_S)$
$(0, 1, 1)$	$(0, 0, 1)$		$P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 1)$	$Z_t + 1 < L$	$(1-P_B) P_S$
$(Z_t, 1, 1)$	$(Min \{Z_t - Q + 1, 0\}, 1, 1)$		$P_B P_S$
$(0, 0, 1)$	$(0, 0, 1)$		$1-P_S$
$(0, 0, 1)$	$(0, 1, 1)$		P_S

TABLE C- 4: MODEL TWO TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER FULL-BATCH POLICY

Initial State	Final State	Conditions for transition	Transition probability
$(L, 1, 0)$	$(L - Q, 1, 1)$		P_B
$(L, 1, 0)$	$(L , 1, 0)$		$1 - P_B$
$(L - 1, 1, 1)$	$(L - 1 - Q, 1, 1)$		$P_B (1 - P_S)$
$(L - 1, 1, 1)$	$(L - Q, 1, 1)$		$P_B P_S$
$(L - 1, 1, 1)$	$(L, 1, 0)$		$(1 - P_B) P_S$
$(L - 1, 1, 1)$	$(L - Q, 1, 1)$		$P_B P_S$
$(Z_t, 1, 1)$	$(Z_t, 1, 1)$	$Z_t < L - 1$	$(1-P_B) (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t - Q, 1, 1)$	$Q < Z_t$	$P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t, 0, 1)$	$Z_t < Q$	$P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 1)$	$Z_t + 1 < L$	$(1-P_B) P_S$
$(Z_t, 1, 1)$	$(Z_t - Q + 1, 1, 1)$	$Q < Z_t + 1 < L$	$P_B P_S$
$(Z_t, 0, 1)$	$(Z_t, 0, 1)$	$Q > Z_t$	$1-P_S$
$(Z_t , 0, 1)$	$(Z_t + 1,0, 1)$	$Q - 1 > Z_t$	P_S
$(Q - 1, 0, 1)$	$(0, 1, 1)$		P_S

C.3 Model Three: Processing time of both processors are geometric, and the batch processor is unreliable

The notation used is identical to what is seen in Section 4.8. There can be two possible threshold values, depending on the state of the batch processor. If the batch processor is busy, the serial processor continues processing until $Z_t = L_1$. If the batch processor is down, the serial processor continues processing until $Z_t = L_2$.

TABLE C- 5: MODEL THREE TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER NO-IDLING POLICY

Initial State	Final State	Conditions for transition	Transition probability
$(Z_t, 1, 0)$	$(Z_t, -1, 0)$	$Z_t > L_2$	P_F
$(Z_t, 1, 0)$	$(Z_t, -1, 1)$	$Z_t < L_2$	P_F
$(Z_t, 1, 1)$	$(Z_t, -1, 1)$		$P_F (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, -1, 1)$	$Z_t < L_2$	$P_F P_S$
$(Z_t, 1, 1)$	$(Z_t + 1, -1, 0)$	$Z_t + 1 > L_2$	$P_F P_S$
$(Z_t, -1, 0)$	$(Z_t, -1, 0)$	$Z_t > L_2$	$1-P_R$
$(Z_t, -1, 0)$	$(min \{Z_t - Q, 0\}, 1, 0)$	$Z_t - Q > L_1$	P_R
$(Z_t, -1, 0)$	$(min \{Z_t - Q, 0\}, 1, 1)$	$min \{Z_t - Q, 0\} < L_1$	P_R
$(Z_t, -1, 1)$	$(Z_t, -1, 1)$		$(1-P_R) (1-P_S)$
$(Z_t, -1, 1)$	$(Z_t + 1, -1, 1)$	$Z_t + 1 < L_2$	$(1-P_R) P_S$
$(Z_t, -1, 1)$	$(Z_t + 1, -1, 0)$	$Z_t + 1 > L_2$	$(1-P_R) P_S$
$(Z_t, -1, 1)$	$(min\{Z_t - Q, 0\}, 1, 1)$	$Z_t > 0$ and $Z_t - Q < L_1$	$P_R (1 - P_S)$
$(Z_t, -1, 1)$	$(min\{Z_t - Q + 1, 0\}, 1, 1)$	$Z_t - Q + 1 < L_1$	$P_R P_S$
$(0, -1, 1)$	$(0, 0, 1)$		$P_R (1 - P_S)$
$(0, -1, 1)$	$(0, 1, 1)$		$P_R P_S$
$(Z_t, 1, 0)$	$(Z_t - Q, 1, 1)$	$Z_t - Q < L_1$	$(1 - P_F) P_B$
$(Z_t, 1, 0)$	$(Z_t, 1, 0)$	$Z_t > L_1$	$(1 - P_F) (1 - P_B)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 0)$	$Z_t + 1 > L_1$	$(1 - P_F) (1 - P_B) P_S$
$(Z_t, 1, 1)$	$(Z_t - Q + 1, 1, 1)$	$Z_t - Q + 1 < L_1$	$(1 - P_F) P_B P_S$
$(Z_t, 1, 1)$	$(Z_t, 1, 1)$		$(1 - P_F) (1-P_B) (1-P_S)$
$(Z_t, 1, 1)$	$(Min \{Z_t - Q, 0\}, 1, 1)$	$0 < Z_t$	$(1 - P_F) P_B (1-P_S)$
$(0, 1, 1)$	$(0, 0, 1)$		$(1 - P_F) P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 1)$	$Z_t + 1 < L_1$	$(1 - P_F) (1-P_B) P_S$
$(Z_t, 1, 1)$	$(Min \{Z_t - Q + 1, 0\}, 1, 1)$	$Z_t - Q + 1 < L_1$	$(1 - P_F) P_B P_S$
$(0, 0, 1)$	$(0, 0, 1)$		$1-P_S$
$(0, 0, 1)$	$(0, 1, 1)$		P_S

The state space is dependent on which threshold L_1 or L_2 is higher. Let L^* be the higher of L_1 and L_2. When the batch processor is under no-idling policy, the state space is composed of:

- $(Z_t, 1, 0)$, with $Z_t = L_1$ to max $(L_1, L_2 - Q)$, if $L_2 = L^*$, and $Z_t = L_1$ if $L_1 = L^*$
- $(Z_t, -1, 0)$, with $Z_t = L_2$ to L^*
- $(Z_t, 0, 1)$, with $Z_t = 0$
- $(Z_t, 1, 1)$, with $Z_t = 0$ to $L_1 - 1$ if $L_1 = L^*$, and $Z_t = 0$ to $max (L_2 - Q - 1, L_1)$ if $L_2 = L^*$

- $(Z_t, -1, 1)$, with $Z_t = 0$ to $L^* - 1$

When the batch processor is under full-batch policy, the state space is composed of:

- $(Z_t, 1, 0)$, with $Z_t = L_1$ to max $(L_1, L_2 - Q)$, if $L_2 = L^*$, and $Z_t = L_1$ if $L_1 = L^*$
- $(Z_t, -1, 0)$, with $Z_t = L_2$ to L^*
- $(Z_t, 0, 1)$, with $Z_t = 0$ to $Q - 1$
- $(Z_t, 1, 1)$, with $Z_t = 0$ to $L_1 - 1$ if $L_1 = L^*$, and $Z_t = 0$ to $max\ (L_2 - Q - 1, L_1)$ if $L_2 = L^*$
- $(Z_t, -1, 1)$, with $Z_t = 0$ to $L^* - 1$

TABLE C- 6: MODEL THREE TRANSITION PROBABILITIES WITH BATCH PROCESSOR UNDER FULL-BATCH POLICY

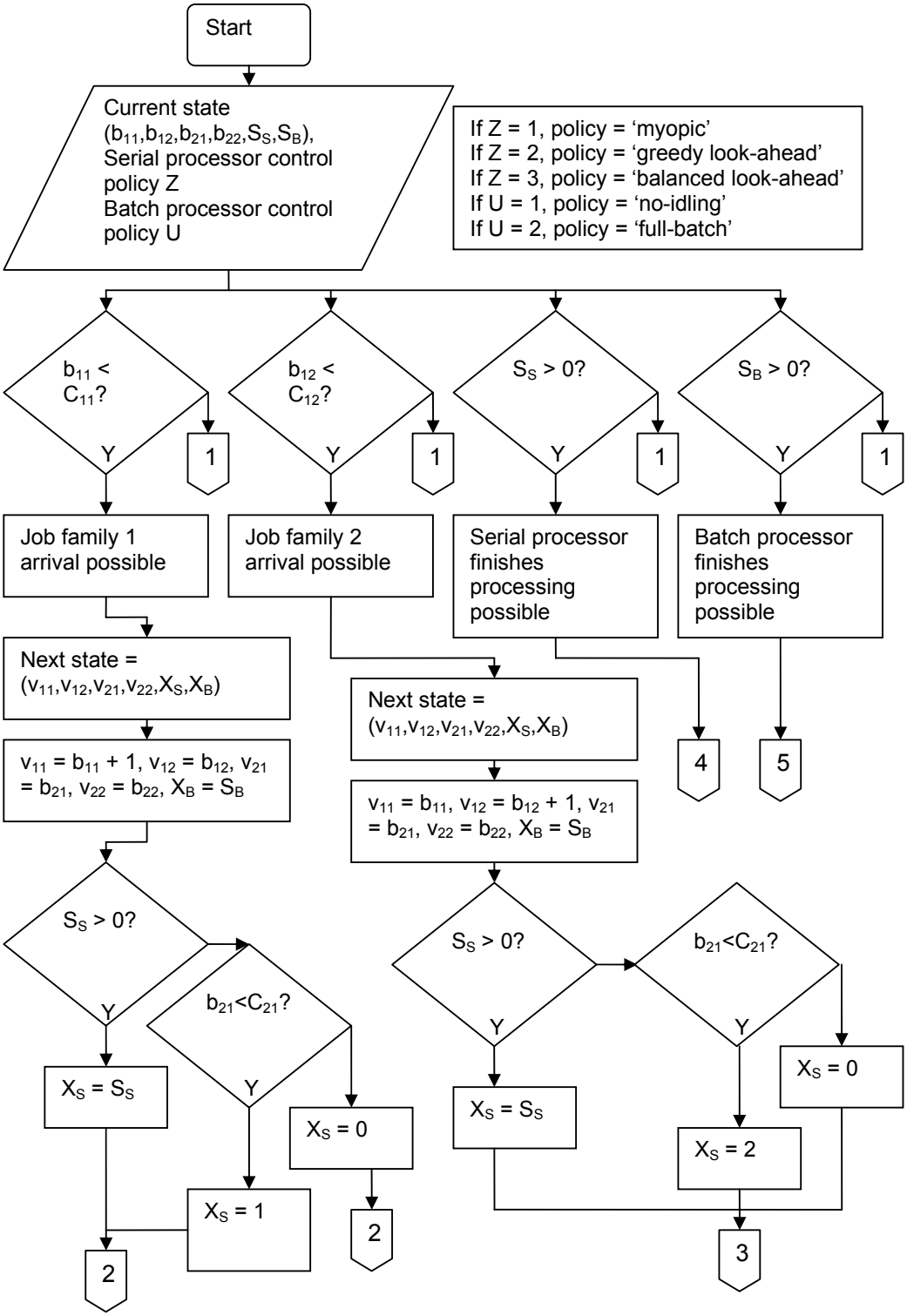
Initial State	Final State	Conditions for transition	Transition probability
$(Z_t, 1, 0)$	$(Z_t, -1, 0)$	$Z_t > L_2$	P_F
$(Z_t, 1, 0)$	$(Z_t, -1, 1)$	$Z_t < L_2$	P_F
$(Z_t, 1, 1)$	$(Z_t, -1, 1)$		$P_F (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, -1, 1)$	$Z_t < L_2$	$P_F P_S$
$(Z_t, 1, 1)$	$(Z_t + 1, -1, 0)$	$Z_t + 1 > L_2$	$P_F P_S$
$(Z_t, -1, 0)$	$(Z_t, -1, 0)$	$Z_t > L_2$	$1-P_R$
$(Z_t, -1, 0)$	$(Z_t - Q, 1, 0)$	$Z_t - Q > L_1$	P_R
$(Z_t, -1, 0)$	$(Z_t - Q, 1, 1)$	$Z_t - Q < L_1$	P_R
$(Z_t, -1, 1)$	$(Z_t, -1, 1)$		$(1-P_R) (1-P_S)$
$(Z_t, -1, 1)$	$(Z_t + 1, -1, 1)$	$Z_t + 1 < L_2$	$(1-P_R) P_S$
$(Z_t, -1, 1)$	$(Z_t + 1, -1, 0)$	$Z_t + 1 > L_2$	$(1-P_R) P_S$
$(Z_t, -1, 1)$	$(Z_t - Q, 1, 1)$	$Z_t > Q$ and $Z_t - Q < L_1$	$P_R (1 - P_S)$
$(Z_t, -1, 1)$	$(Z_t, 0, 1)$	$Z_t < Q$	$P_R (1 - P_S)$
$(Z_t, -1, 1)$	$(Z_t + 1, 1, 1)$	$Z_t < Q - 1$	$P_R P_S$
$(Z_t, -1, 1)$	$(Z_t - Q + 1, 1, 1)$	$Z_t - Q + 1 < L_1$ and $Z_t > Q - 1$	$P_R P_S$
$(Z_t, 1, 0)$	$(Z_t - Q, 1, 1)$	$Z_t - Q < L_1$	$(1 - P_F) P_B$
$(Z_t, 1, 0)$	$(Z_t, 1, 0)$	$Z_t > L_1$	$(1 - P_F) (1 - P_B)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 0)$	$Z_t + 1 > L_1$	$(1 - P_F) (1 - P_B) P_S$
$(Z_t, 1, 1)$	$(Z_t + 1, 0, 1)$	$Z_t < Q - 1$	$(1 - P_F) P_B P_S$
$(Z_t, 1, 1)$	$(Z_t - Q + 1, 1, 1)$	$Z_t - Q + 1 < L_1$ and $Z_t > Q - 1$	$(1 - P_F) P_B P_S$
$(Z_t, 1, 1)$	$(Z_t, 1, 1)$		$(1 - P_F) (1-P_B) (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t, 0, 1)$	$Q > Z_t$	$(1 - P_F) P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t - Q, 1, 1)$	$Q < Z_t$	$(1 - P_F) P_B (1-P_S)$
$(Z_t, 1, 1)$	$(Z_t + 1, 1, 1)$	$Z_t + 1 < L_1$	$(1 - P_F) (1-P_B) P_S$
$(Z_t, 1, 1)$	$(Z_t - Q + 1, 1, 1)$	$Z_t - Q + 1 < L_1$ and $Z_t > Q - 1$	$(1 - P_F) P_B P_S$
$(Z_t, 1, 1)$	$(Z_t + 1, 0, 1)$	$Z_t < Q - 1$	$(1 - P_F) P_B P_S$
$(Z_t, 0, 1)$	$(Z_t, 0, 1)$		$1-P_S$
$(Z_t, 0, 1)$	$(Z_t + 1, 0, 1)$	$Z_t < Q - 1$	P_S
$(Q - 1, 0, 1)$	$(0, 1, 1)$		P_S

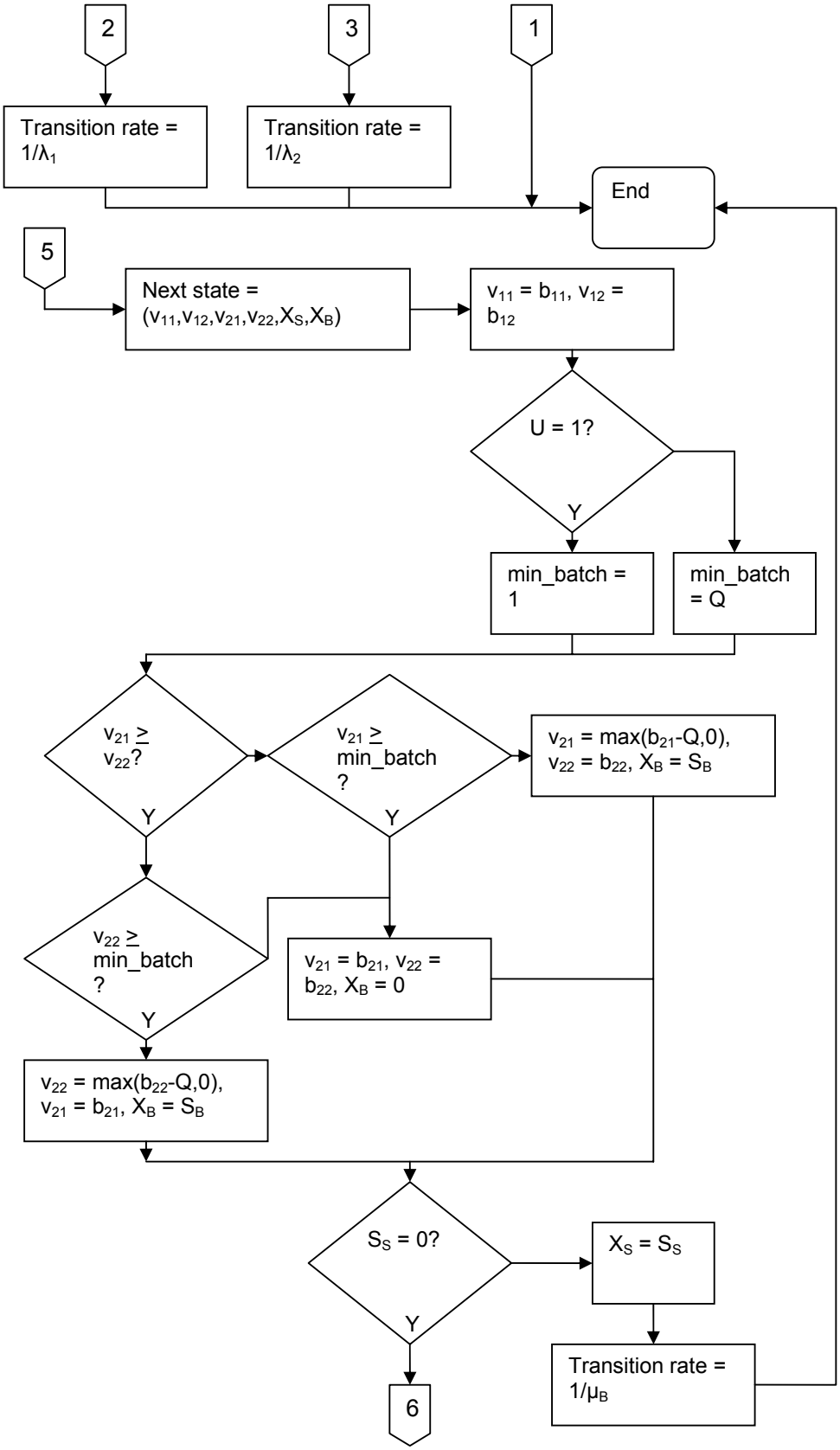
Appendix D

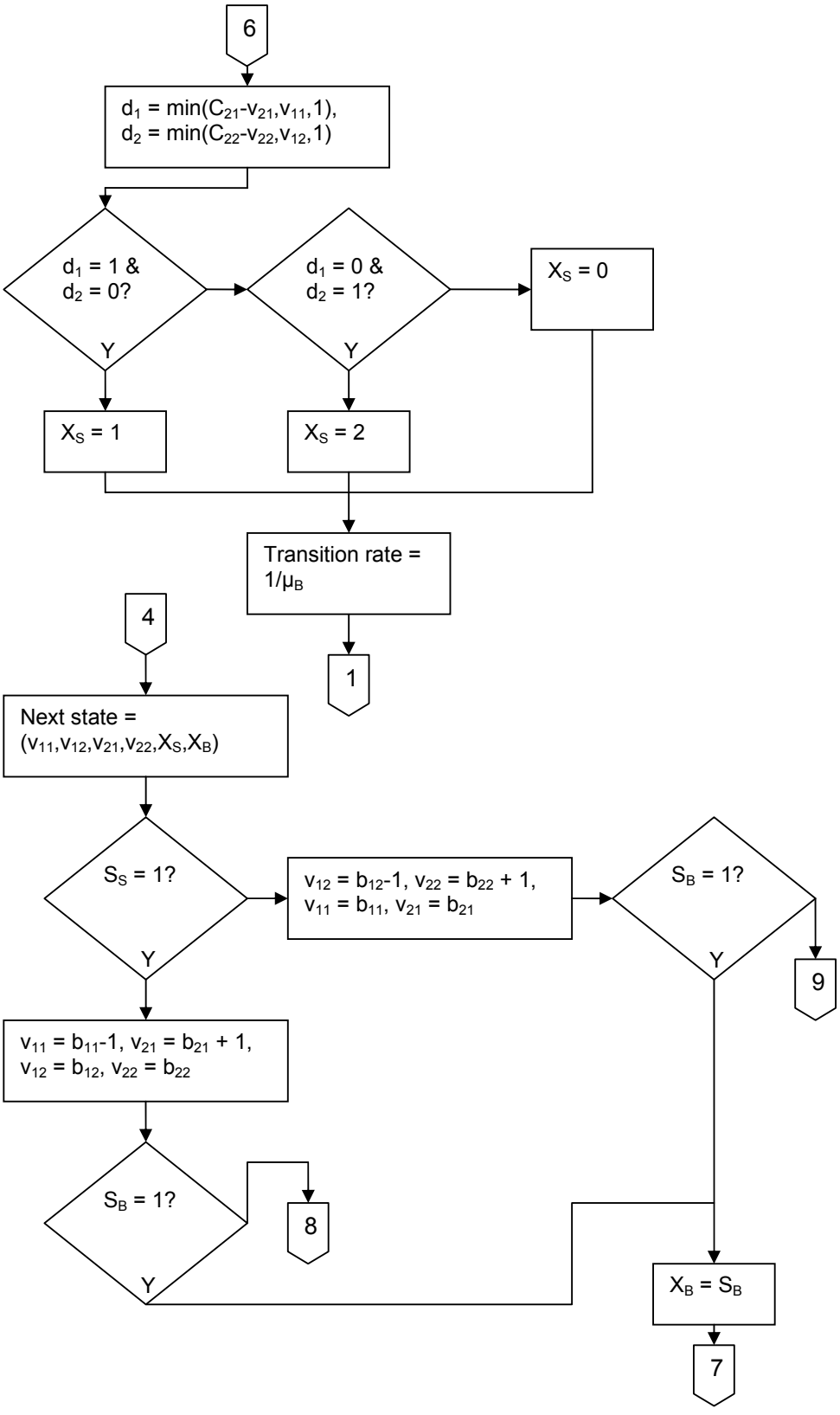
Probability Transition Equation Flowcharts for Two-Stage Markov Models

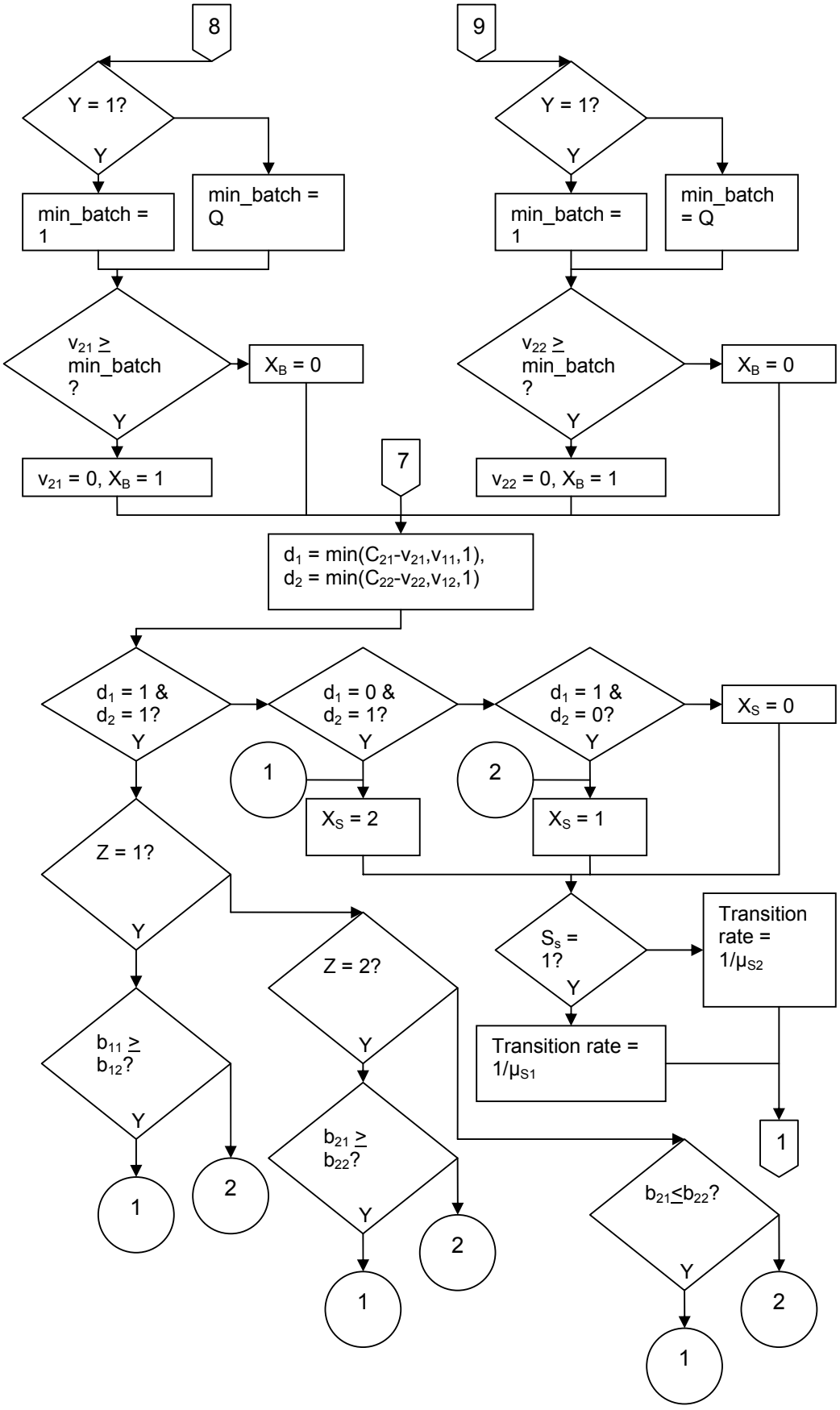
It is possible to list all possible transitions away from a current state. However, presenting the possible states from a current state through a flowchart is more intuitive, in terms of comparing the logic for each policy. We segregate the flowcharts according to the type of processor configuration seen in the manufacturing system.

D.1 Serial Processor feeding Batch Processor

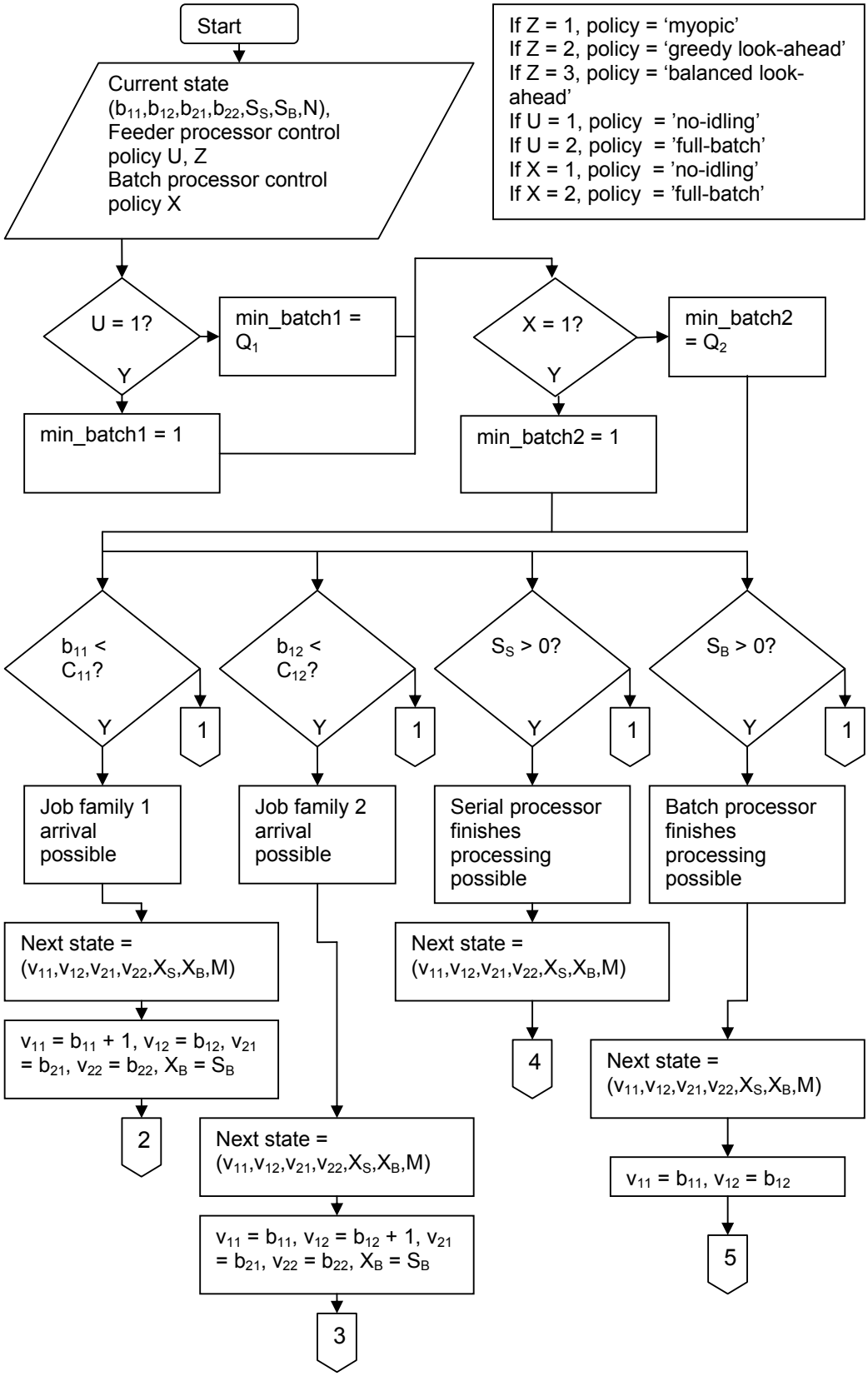


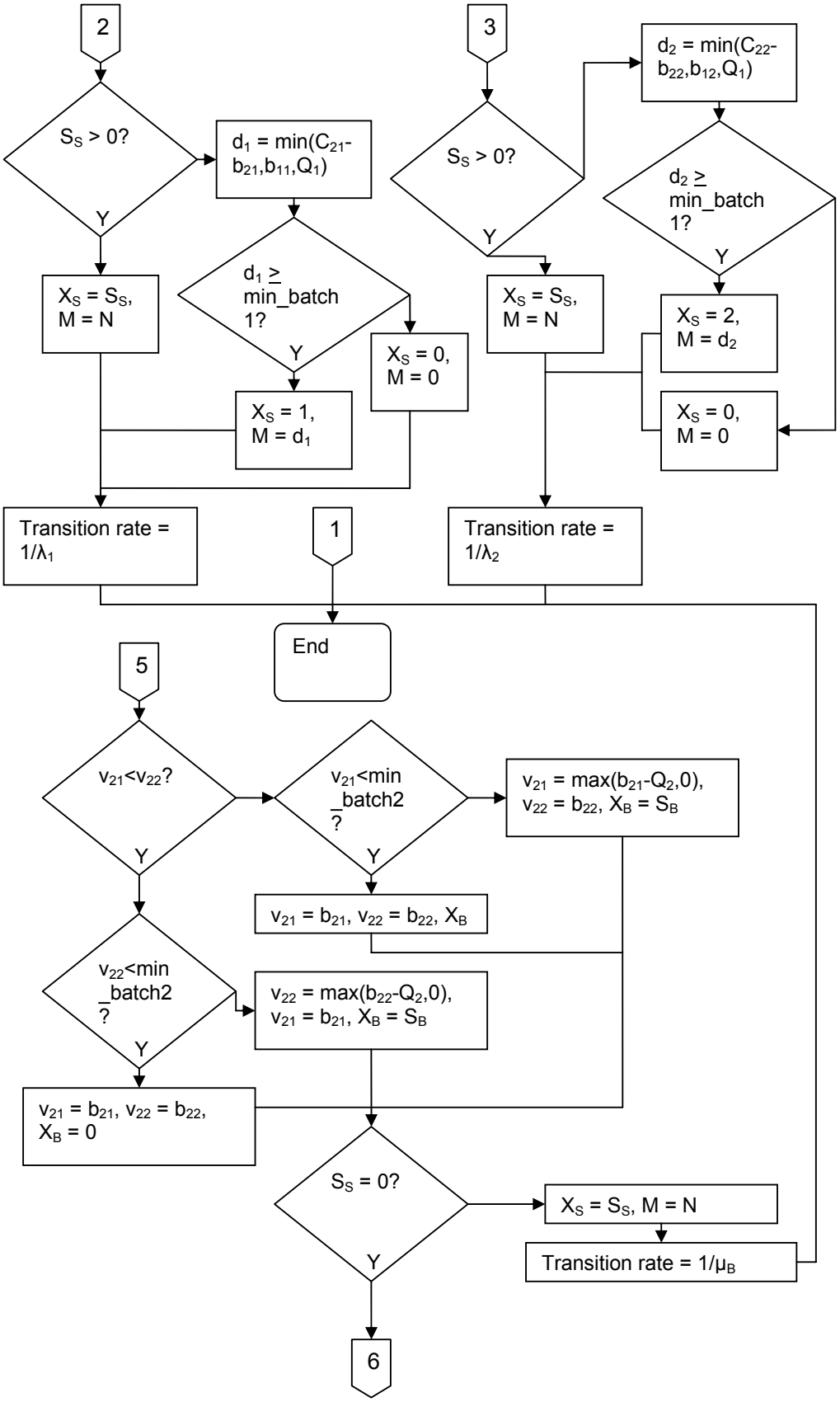


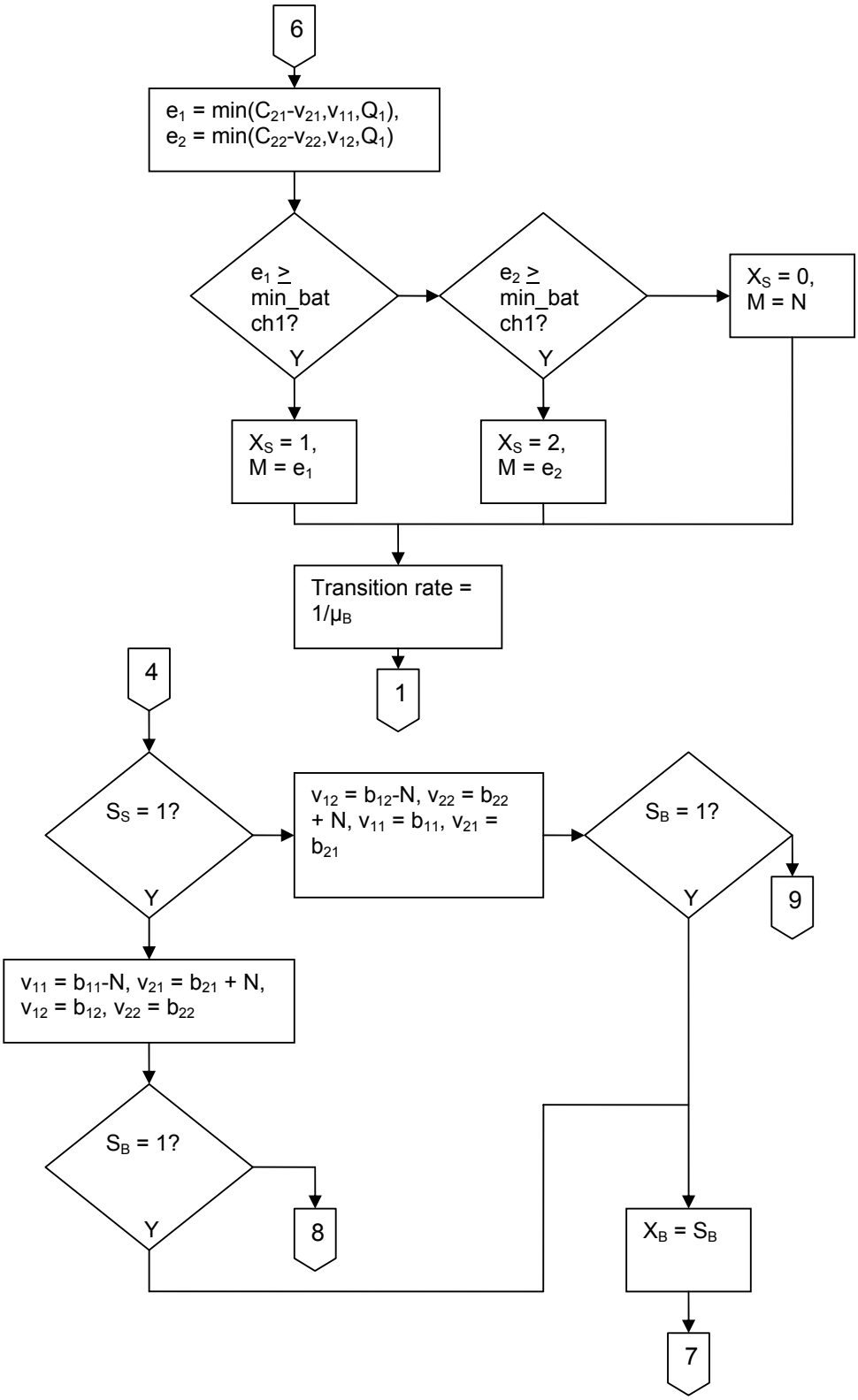


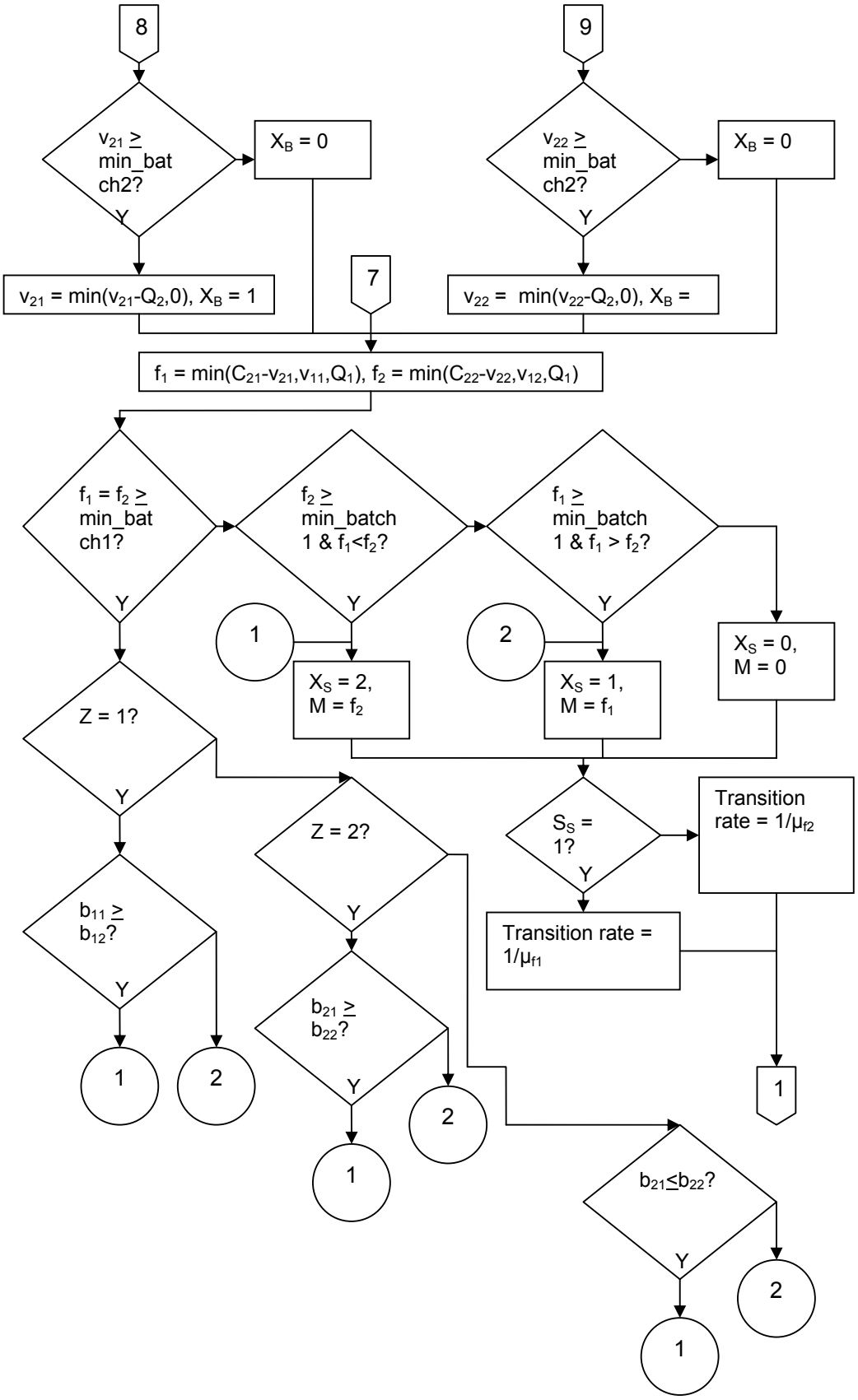


D.2 Two Batch Processors in series







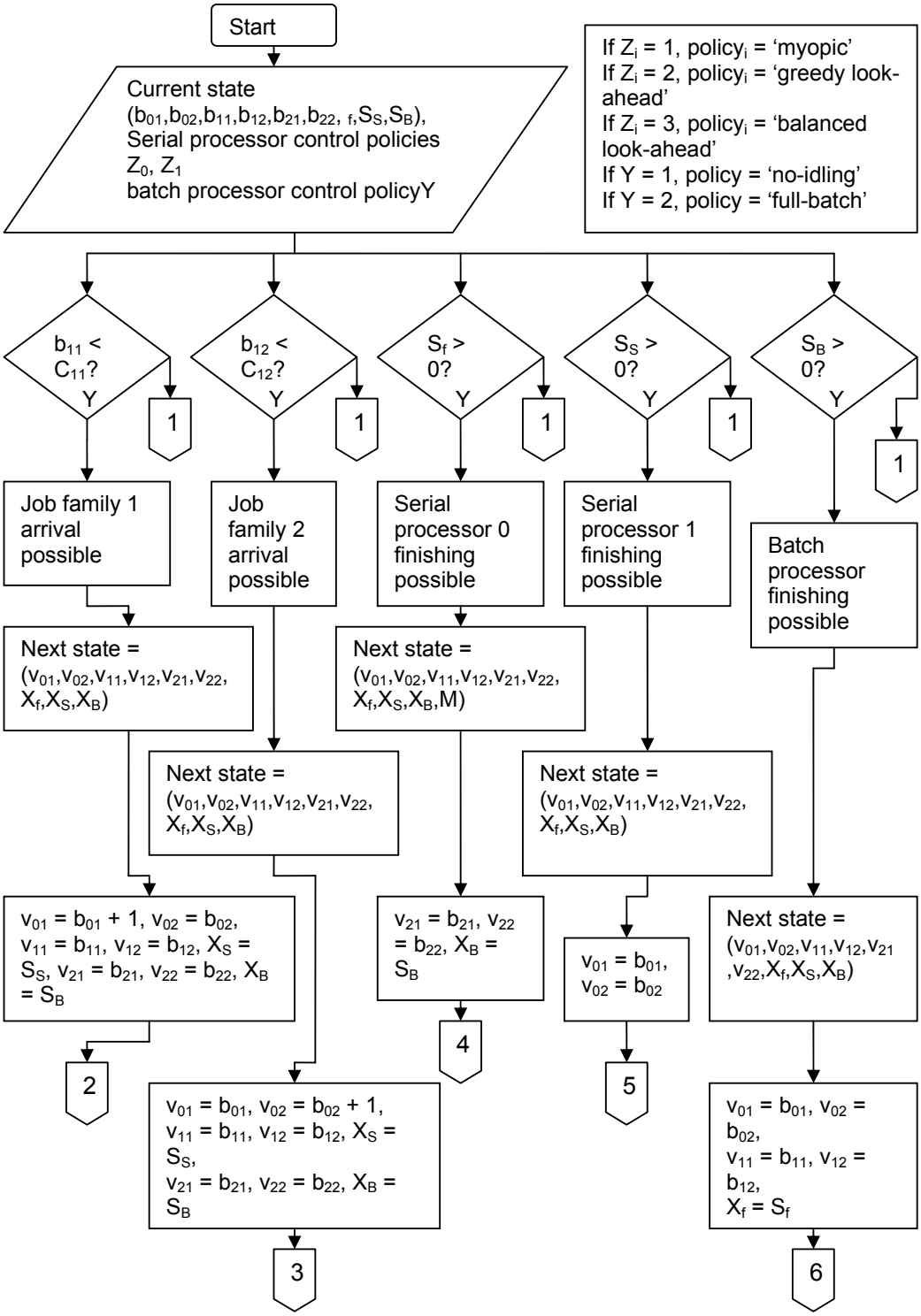


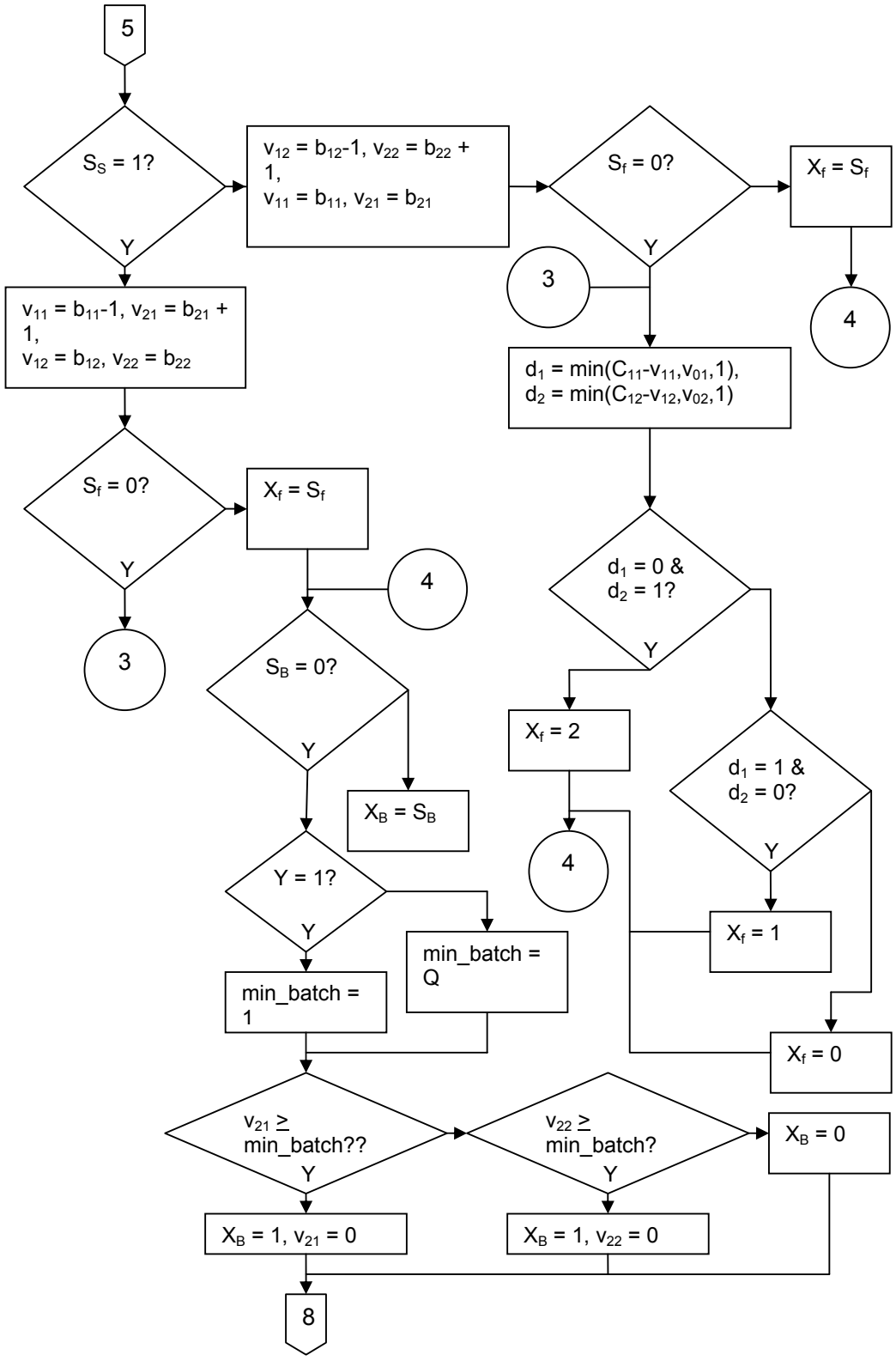
Appendix E

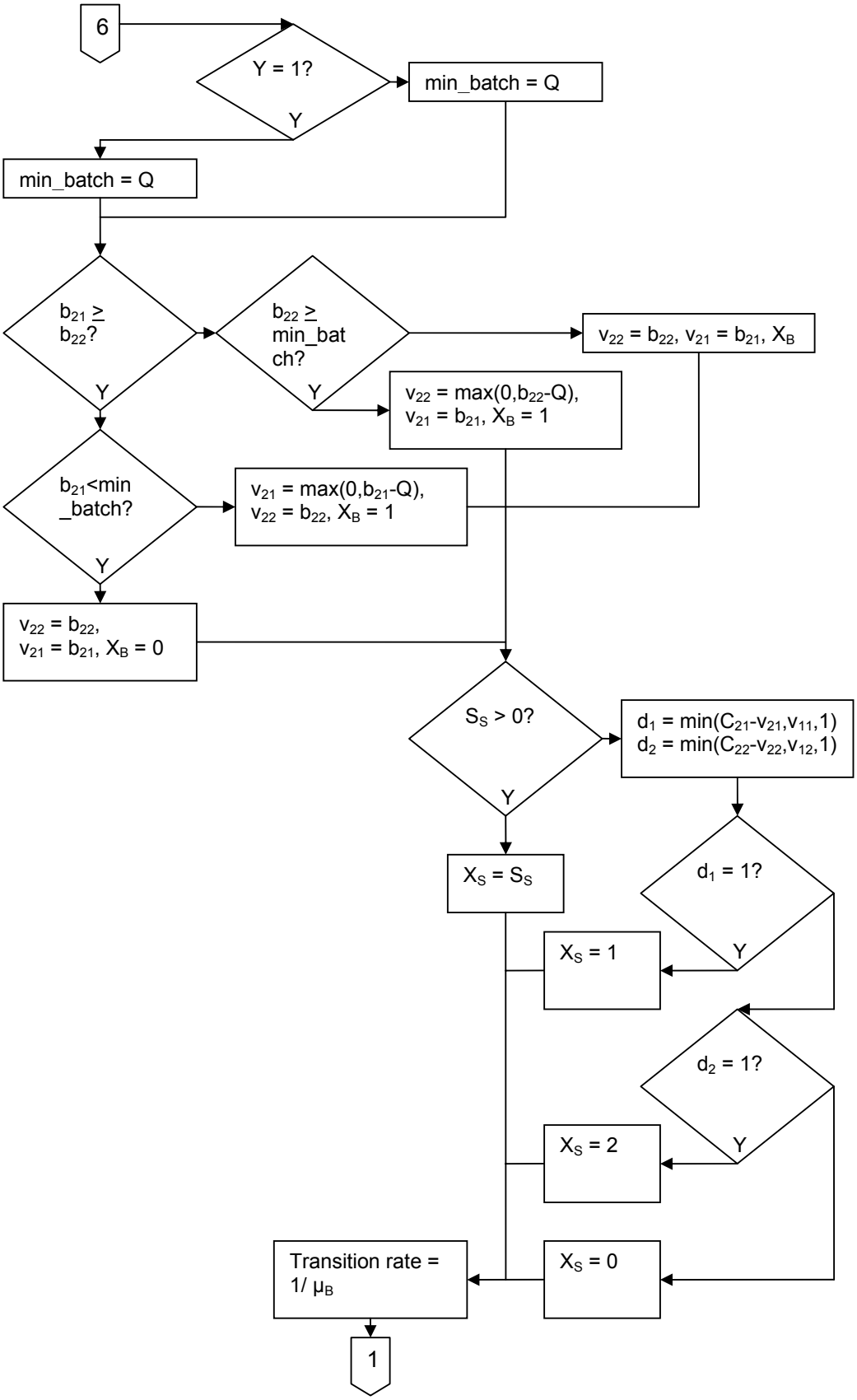
Probability Transition Equation Flowcharts for Three-Stage Markov Models

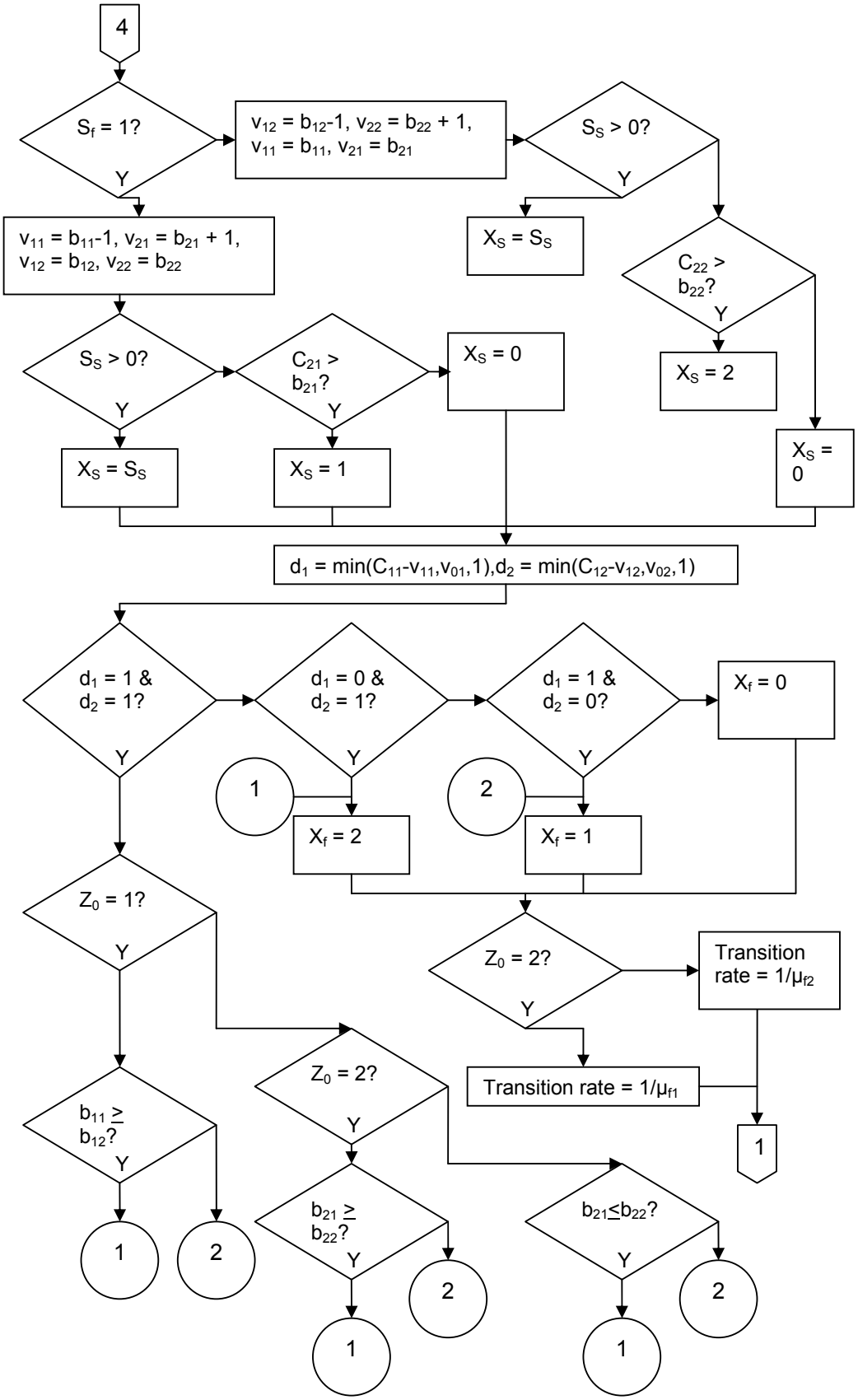
Similar to what was done for the two-stage Markov models, the process of obtaining the transition probabilities out of a specific state is shown.

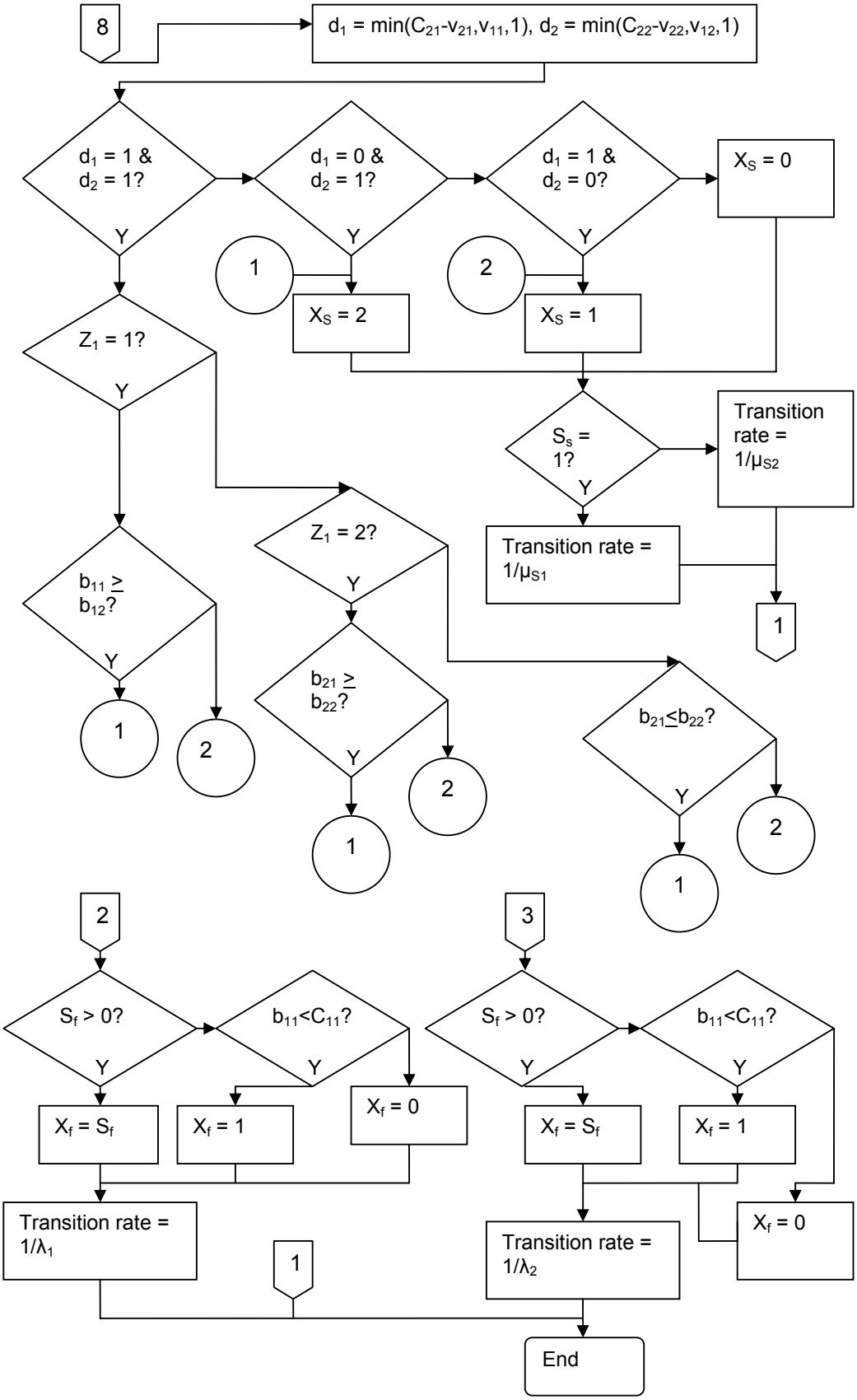
E.1 Serial-Serial-Batch Processor System











E.2 Serial processor feeds (feeder) batch processor, which feeds batch processor