Provably-efficient adaptive scheduling of multiprocessors for performance and energy

Sun, Hongyang

2011

Sun, H. Y. (2011). Provably-efficient adaptive scheduling of multiprocessors for performance and energy. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/43571>

<https://doi.org/10.32657/10356/43571>

NANYANG
TECHNOLOGICAL
UNIVERSITY

Provably-Efficient Adaptive Scheduling of Multiprocessors for Performance and Energy

Hongyang Sun

Thesis Supervisor:
Assoc. Prof. Wen-Jing Hsu

School of Computer Engineering

This thesis is submitted to Nanyang Technological University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

2011

Abstract

Performance and energy are two crucial but conflicting objectives in the design of modern high-performance computing systems. This thesis considers efficient scheduling of parallel applications on multiprocessors to optimize performance and energy consumption. We mainly focus on a two-level framework and dynamic speed scaling with minimal or no assumption about the characteristics of the jobs, such as their release time, remaining work and parallelism profile. Both processor allocations and processor speeds (if possible) are periodically readjusted based on the estimates of a job's parallelism and the scheduling policy of the operating system.

We present a novel adaptive scheduler that calculates parallelism estimations for a job using an adaptive controller and allocates processors based on the dynamic equi-partitioning policy. Compared to an existing adaptive scheduler proposed under the same two-level framework but experiences feedback instability, we show that our scheduler achieves stable feedback along with other desirable control-theoretic properties, such as zero steady-state error, zero overshoot and a user-configurable convergence rate. Moreover, we analyze from algorithmic perspective the performances of our scheduler and its variants with respect to several objective functions, including total response time, makespan and set response time on fixed speed processors, as well as total response time plus energy on variable speed processors. We show that our algorithms achieve optimal or nearly optimal performances with respect to these objectives. The simulation results, which are obtained based on malleable parallel jobs constructed with a set of internal parallelism variations, further confirmed the advantage of our scheduler in terms of both control-theoretic properties and algorithmic performances under a wide range of workloads.

Furthermore, we consider adaptive scheduling where exact parallelism of a job is available to the scheduler at any time. We show that such information significantly improves performance and energy on variable speed processors. In particular, we present algorithms that effectively utilize this information and achieve optimal performances with respect to either total response time plus energy or makespan plus energy.

To the best of our knowledge, this thesis presents the first study on multiprocessor scheduling that offers both desirable control-theoretic properties and optimal algorithmic performance guarantees.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Wen-Jing Hsu for his excellent guidance in the past 4 years. Prof. Hsu is a wonderful supervisor and researcher. He has offered me numerous help in many aspects of my research. He always gives me freedom and trust in exploring different directions, while at the same time provides many insightful ideas and suggestions. He always challenges me for more rigor in my derivations, and shows me how to clarify my writings. I really appreciate his devotions in both supervising and research, which have led to many fruitful results in our collaboration [90, 178, 173, 174, 175, 172, 46, 91, 176, 177]. Moreover, besides much help in research, Prof. Hsu also shares many of his life experiences and offers valuable advices for my future career.

Many ideas in this thesis could not have been possible without discussions with Prof Hsu's research team at NTU, whose members in the past 4 years include Yuxiong He, Yangjie Cao, Hui Fang, Shin Yee Chung and Miao Lin.

I would like to specially thank Dr. Yuxiong He now of the Microsoft Corporation. Yuxiong was a senior graduate student when I first entered my Ph.D study. I am immediately impressed by her sharp minds and research enthusiasm. She generously shared an interesting research topic with me and patiently answered many of my technical questions. I could have never entered the field so quickly without her help. Besides our fruitful collaborations [90, 91, 176, 177], she also give me many kind advices in life. From the bottom of my heart, I am very grateful to her, and hope to have the opportunity to collaborate with her again someday.

I would also like to specially thank Mr. Yangjie Cao of Xi'an Jiaotong University in China. Yangjie collaborated with me as an exchange student at NTU. Over his one year stay here, we had a lot of fun developing algorithms and hacking simulations together [173, 174, 175, 172, 46]. His dedication has greatly improved the quality of our research, especially the simulation results. I sincerely cherish our friendship, as well as those wonderful times we spent on conference trips.

Many thanks to my family and friends for their great support in the past 10 years while I was in Singapore. Their presence has made my life so much enjoyable and colorful. Last but not least, I would like to thank NTU for giving me the opportunity to pursue my Ph.D. study, and for CAIS for providing a pleasant research environment.

Contents

Abstract	i
Acknowledgements	iii
List of Figures	ix
List of Tables	x
Nomenclature	xi
1 Introduction	1
1.1 Adaptive Scheduling	1
1.2 A Two-Level Scheduling Framework	3
1.3 Dynamic Speed Scaling	5
1.4 Job Model and Scheduling Model	6
1.5 Objectives Functions	7
1.6 Performance Evaluations	8
1.6.1 Competitive Analysis	8
1.6.2 Empirical Simulations	9
1.7 Statement of Results	11
1.8 Synopsis	13
2 Literature Review	14
2.1 Adaptive Scheduling	15
2.1.1 Algorithmic Perspective	15
2.1.2 Control-Theoretic Perspective	19
2.1.3 Empirical Perspective	24
2.2 Minimizing Various Objectives	29
2.2.1 Total Response Time	30
2.2.2 Makespan	31
2.2.3 Set Response Time	32
2.2.4 Energy-Related Objectives	33

3	Two-Level Adaptive Scheduling: From A-GREEDY to ACDEQ	38
3.1	Revisit Adaptive Task Scheduler: A-GREEDY	39
3.2	Stable Two-Level Adaptive Scheduler: ACDEQ	40
3.3	Control-Theoretic Analysis	42
3.4	Algorithmic Preliminaries	45
3.4.1	Properties of ACDEQ	46
3.4.2	Properties of AGDEQ	49
3.5	Empirical Evaluations	51
3.5.1	Transient Responses	51
3.5.2	Individual Job Performances	52
4	Total Response Time and Makespan on Fixed Speed Processors	55
4.1	Total Response Time of ACDEQ	56
4.1.1	Lower Bounds and Preliminaries	56
4.1.2	Analysis for Batched Jobs	58
4.1.3	Analysis for Nonbatched Jobs	60
4.2	Analysis Framework for Total Response Time and Performance of AGDEQ	63
4.3	Makespan of ACDEQ and AGDEQ	65
4.4	Discussions	67
4.5	Empirical Evaluations	68
5	Set Response Time on Fixed Speed Processors	72
5.1	Preliminaries and Lower Bounds	73
5.2	EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ	75
5.3	Analysis of EQUI $\circ$ AGDEQ for Batched Job sets	77
5.4	Analysis of EQUI $\circ$ AGDEQ for Nonbatched Job sets	79
5.5	Analysis Framework for Set Response Time and Performance of EQUI $\circ$ ACDEQ	82
5.6	Discussions	84
5.7	Empirical Evaluations	85
6	Total Response Time plus Energy on Variable Speed Processors	89
6.1	A Lower Bound on Non-clairvoyant Uniform Speed Scaling	90
6.2	Analysis Framework for Total Response Time plus Energy	91
6.2.1	Preliminaries and the N-X algorithm	91
6.2.2	Analysis of N-X for Batched Jobs	95
6.2.3	Analysis of N-X for Nonbatched Jobs	97
6.3	Performances of N-AGCEQ and N-ACCEQ	101
6.4	A Lower Bound on Non-clairvoyant Non-uniform Speed Scaling	104
6.5	Discussions	107
6.6	Empirical Evaluations	108

7	Semi-Clairvoyant Algorithms for Performance plus Energy on Variable Speed Processors	110
7.1	U-CEQ for Total Response Time plus Energy	111
7.2	P-FIRST for Makespan plus Energy	112
7.2.1	Preliminaries and Lower Bounds	113
7.2.2	Performance of P-FIRST	114
7.3	Discussions	117
8	Conclusion	119
8.1	Summary	119
8.2	Future Work	121
8.3	Concluding Remarks	123
	List of Publications	124
	Bibliography	126

List of Figures

1.1	Static scheduling that allocates a fixed number of processors to jobs can cause processor wastes or execution delays.	2
1.2	The two-level framework for adaptive scheduling of processor resources.	4
1.3	Five different parallelism structures specified by Step, Impulse, Ramp, Poly(I) and Poly(II) profiles.	11
3.1	Feedback control structure of ACDEQ.	41
3.2	Transient and steady-state behaviors of A-CONTROL and A-GREEDY over several phases of a synthetic job.	45
3.3	Transient responses of A-CONTROL and A-GREEDY on (a) Step and (b) Poly(II) profiles.	52
3.4	Transient responses of A-CONTROL and A-GREEDY on (a) Ramp and (b) Poly(I) profiles.	53
3.5	Individual job performances of A-CONTROL and A-GREEDY on the five parallelism profiles in terms of response time and processor waste. .	54
4.1	An example of $n_t(z)$ at a particular time t , and the changes of $n_t(z)$ in an infinitesimal interval Δt in the total response time analysis of ACDEQ for batched jobs.	59
4.2	An example of $n_t(z)$ and $n_t^*(z)$ at a particular time t , the changes of $n_t(z)$ and $n_t^*(z)$ after a new job arrives, and the changes of $n_t(z)$ and $n_t^*(z)$ in an infinitesimal interval Δt in the total response time analysis of ACDEQ for nonbatched jobs.	61
4.3	The performance comparisons of ACDEQ and AGDEQ with the EQUI scheduler in terms of makespan and total response time.	69
4.4	The number of processor reallocations of ACDEQ and AGDEQ on the last completed job and all jobs in the job set.	70
4.5	The performance comparison of ACDEQ and AGDEQ with different cost for reallocation overhead in terms of makespan and total response time.	71
5.1	An example of $m_t(z)$ at a particular time t , and the changes of $m_t(z)$ in an infinitesimal interval Δt in the set response time analysis of EQUI $\circ$ AGDEQ for batched job sets.	78

5.2	An example of $m_t(z)$ and $m_t^*(z)$ at a particular time t , the changes of $m_t(z)$ and $m_t^*(z)$ after a new job set arrives, and the changes of $m_t(z)$ and $m_t^*(z)$ in an infinitesimal interval Δt in the set response time analysis of EQUI $\circ$ AGDEQ for nonbatched job sets.	80
5.3	The performance comparison of EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ with EQUI $\circ$ EQUI in terms of set response time for light loads and heavy loads.	86
5.4	The performance comparison of EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ with EQUI $\circ$ EQUI in terms of processor utilization for light loads and heavy loads.	87
6.1	An example of $n_t(z)$ at a particular time t , and the changes of $n_t(z)$ in an infinitesimal interval Δt in the total response time plus energy analysis of N-X for batched jobs.	96
6.2	An example of $n_t(z)$ and $n_t^*(z)$ at a particular time t , the changes of $n_t(z)$ and $n_t^*(z)$ after a new job arrives, and the changes of $n_t(z)$ and $n_t^*(z)$ in an infinitesimal interval Δt in the total response time plus energy analysis of N-X for nonbatched jobs.	98
6.3	The performance comparison of N-ACCEQ and N-AGCEQ with N-EQUI in terms of total response time plus energy.	108

List of Tables

2.1	Some algorithmic results on EQUI and its variants.	17
2.2	Some algorithmic results on DEQ and its variants.	19
2.3	Some control-theoretic results on adaptive scheduling as well as their application domains and design strategies.	24
2.4	Some experimental results on adaptive scheduling with implementation environments and workloads.	26
2.5	Some simulation results on adaptive scheduling with simulation envi- ronments and workloads.	29
2.6	Some results on sequential job scheduling that minimize total response time.	31
2.7	Some results on sequential job scheduling that minimize makespan. . .	33
2.8	Some results on parallel job scheduling that minimize set response time.	34
2.9	Some results that minimize total response time or makespan plus energy.	36

Nomenclature

α	Power parameter
δ	Utilization parameter of A-GREEDY
Γ_i^k	Speedup function of phase J_i^k
$\hat{a}_A(\mathcal{J})$	Squashed accounted processor allocation for job set $\mathcal{J}$
$\hat{w}(\mathcal{J})$	Squashed work of job set $\mathcal{J}$
$\mathcal{J}$	Job set (or job set collection in Chapter 5)
$\Phi(t)$	Potential function at time t
ρ	Responsiveness parameter of A-GREEDY
$A_i(q)$	Average parallelism of job J_i in quantum q
$a_i(q)$	Processor allocation of job J_i in quantum q
$a_i(t)$	Processor allocation of job J_i at time t
C	Maximum transition factor of all jobs in $\mathcal{J}$
C_i	Transition factor of job J_i
c_i	Completion time of job J_i
$d_i(q)$	Processor desire of job J_i in quantum q
$E(\mathcal{J})$	Energy consumption of job set $\mathcal{J}$
e_i	Energy consumption of job J_i
$F(\mathcal{J})$	Set response time of job set collection $\mathcal{J}$
$G(\mathcal{J})$	Total response time plus energy of job set $\mathcal{J}$
$H(\mathcal{J})$	Makespan plus energy of job set $\mathcal{J}$
h_i^k	Parallelism of phase J_i^k

J_i	i -th job in job set $\mathcal{J}$
J_i^k	k -th phase of job J_i
L	Quantum length
$l(J_i)$	Span of job J_i
$l_i(q)$	Span reduced for job J_i in quantum q
l_i^k	Span of phase J_i^k
m	Number of job sets in job set collection $\mathcal{J}$
$M(\mathcal{J})$	Makespan of job set $\mathcal{J}$
m_t	Number of active job sets at time t
n	Number of jobs in job set $\mathcal{J}$
n_t	Number of active jobs at time t
P	Total number of processors
$R(\mathcal{J})$	Total response time of job set $\mathcal{J}$
$R(J_i)$	Response time of job J_i
r_i	Release time of job J_i
$s_{ij}(t)$	Speed of j -th processor allocated to job J_i at time t
$t_B(\mathcal{J})$	Total deductible processing time for job set $\mathcal{J}$
$u_i(t)$	Power consumption of job J_i at time t
u_t	Power consumption of all jobs at time t
v	Convergence rate of A-CONTROL
$w(J_i)$	Work of job J_i
$w_i(q)$	Work completed for job J_i in quantum q
w_i^k	Work of phase J_i^k
$X(J_i)$	Processor waste of job J_i

Chapter 1

Introduction

With the increasing demands for high-performance computing, more software developers nowadays have started programming in parallel and migrating existing sequential applications onto parallel platforms. While the eventual success of such “parallel revolution” depends on many factors, an important challenge lies in efficiently scheduling the applications with possibly a wide range of parallelism on the available parallel computing resources. Recently, as multicore computers begin to proliferate, the needs for such efficient schedulers have become increasingly imminent.

Traditionally, the focus of parallel scheduling has been on performance alone. However, in today’s world of high-performance computing, performance seems no longer the only concern. In particular, energy consumption has also been widely recognized as a key consideration. This is not only crucial for multicore-enabled laptops, which often rely on batteries for energy, but also critical to server farms and supercomputer centers, where the heat generated by large numbers of processors is getting harder to dissipate. Thus, how to take energy consumption into account has become another important challenge in designing today’s efficient schedulers.

This thesis addresses these challenges by presenting provably-efficient scheduling algorithms that provide guarantees for both performance and energy consumption under various scheduling scenarios.

1.1 Adaptive Scheduling

Although multiprocessor scheduling has been an important area of research in computer science and operational research for decades, most existing work in this area focused on *static scheduling* (e.g. [82, 166, 181, 55, 113, 33]), which allocates a fixed number of processors to an application throughout its execution lifetime. However, parallel applications often have different processor requirements over different phases of execution, and thus they naturally exhibit time-varying parallelism. In this case, static scheduling can either incur unnecessary processor waste and hence low resource utilization if more processors are allocated to a job than the job’s parallelism (see Fig-

Figure 1.1: Static scheduling that always allocate a fixed number of processors to a job can cause (a) resource waste if more processors are allocated than the job’s parallelism, or (b) execution delay if less processors are allocated than the job’s parallelism.

ure 1.1(a)), or cause execution delays if fewer processors are allocated than the job’s parallelism (see Figure 1.1(b)).

In fact, many parallel jobs nowadays can be designed to run on a variable number of processors, and such jobs are referred to as *malleable jobs* [76]. Indeed, parallel applications created in the popular *workpile* programmed model [76] using languages or libraries such as MultiLisp [86], High Performance Fortran [95], Cilk [31], Intel Thread Building Blocks [1], Microsoft Task Parallel Library [120], Presto [29], CThread [58], can all be regarded as malleable jobs. Basically, workpile programming model allows an application to create more tasks (threads) than the allocated processors. The tasks are all kept on a workpile and the workers (processors) can pick up the ready tasks from the workpile in arbitrary order and execute them [76]. Such flexibility in task execution effectively allows the number of processors allocated to an application to vary during execution, which lays the foundation for an exciting and promising scheduling paradigm, known as *adaptive scheduling* [76, 3, 88].

In this thesis, we study adaptive scheduling of parallel applications by dynamically changing the number of processors allocated to them, as well as the speeds of the processors if possible. Specifically, we consider scheduling a set $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ of n parallel jobs on a total number P of processors.¹ If the processors have fixed speeds, which we assume are identical in this thesis, a scheduling algorithm at any time t needs to specify the number $a_i(t)$ of processors allocated to each job J_i . In addition, if the processors have variable speeds that can be dynamically scaled, then besides processor allocation, the algorithm also needs to assign the speed $s_{ij}(t)$ for each allocated processor to job J_i at time t , where $1 \leq j \leq a_i(t)$. The objectives include optimizing a set of metrics regarding the performance and the energy consumption of all jobs in $\mathcal{J}$, whose formal descriptions are given in Section 1.5.

¹In Chapter 5, however, we will consider scheduling a collection of job sets, and therefore the setting differs slightly from the one described here. For clarity of presentation, we will defer the description of the particular setting to that chapter.

Intuitively, adaptive scheduling should provide apparent benefits over its static counterpart. However, there are also challenges when applying it to actual systems. First of all, all adaptive schedulers need to work in an *online* manner, that is, they will not be aware of the existence of a job until the job is released, since job arrival information is generally not available to the system scheduler. Moreover, characteristics of a job, such as its total work and parallelism profile, are often not available to the operating system as well. Hence, an adaptive scheduler should also be *non-clairvoyant*, that is, it should make all scheduling decisions without any prior knowledge of the job’s characteristics. Sometimes, if certain specific characteristics of the jobs are available at runtime or compile time, we call the adaptive schedulers *semi-clairvoyant*. While most of this thesis focuses on non-clairvoyant adaptive scheduling, Chapter 7 presents semi-clairvoyant adaptive schedulers, which are only aware of the *instantaneous parallelism* of a job at any time, but not the job’s remaining work and future parallelism profile. All of these restrictions on the knowledge of the jobs make it a challenging problem to design efficient adaptive schedulers.

In addition to these challenges, one also needs to consider some practical issues when implementing adaptive scheduling in real systems, such as the overheads incurred due to context switching as well as process management when varying the number of processors and their speeds for a job. However, as efficient process management schemes have been developed and context switching costs have been significantly reduced over the years in modern multiprocessor systems [145, 5, 112], those complications seem no longer a major issue to hinder the adoption of adaptive scheduling into mainstream computing. Moreover, as we will describe shortly in the next section, we employ a quantum-based adaptive mechanism, in which processor allocations and their speeds are only changed between scheduling quanta. Within a quantum, the assignments and the speeds of the processors remain static, even if the parallelism of a job can change drastically. Hence, any overhead incurred will be amortized over the entire scheduling quantum, thus only contributes to a limited and controlled performance degradation. Hence, in the rest of this thesis, we will ignore the various issues associated with system implementations and only focus on providing algorithmic foundations for efficient adaptive scheduling. The main approaches we employ include a two-level scheduling framework as well as dynamic speed scaling.

1.2 A Two-Level Scheduling Framework

It has been shown that *single-level* scheduling, in which the operating system alone is responsible for scheduling the tasks of all jobs, is not necessarily the best choice for parallel applications [147, 85, 190, 76]. The reason is primarily because the operating system lacks job-specific information such as synchronization points of the tasks within a job. Thus, it often leads to poor performance for individual jobs as well as unfair-

Figure 1.2: The two-level framework for adaptive scheduling of processor resources.

ness among the jobs. Therefore, we mainly focus on a promising *two-level* scheduling framework [139, 76, 3, 88, 126] in this thesis. Under this two-level framework, the functions of processor allocation and task scheduling are conveniently decoupled into two separate components. Specifically, a system-level *OS allocator* is responsible for allocating processors to jobs and a user-level *task scheduler* schedules the tasks of a job with the allocated processors. The executions of the jobs are divided into *scheduling quanta*, where a quantum is an interval of time with a pre-defined length usually configured depending on the specific system environments and performance considerations. Moreover, in order to guide processor allocations, the task scheduler periodically provides feedbacks to the OS allocator between each scheduling quantum. The feedbacks are in the form of the job’s processor desires, and are usually calculated according to the execution characteristics of the job in the previous quantum. Based on the processor desires of all jobs and the scheduling policy of the system, the OS allocator then decides an appropriate number of processors to allocate to each job in the next quantum. This process repeats until all jobs in the system complete execution.

Figure 1.2 illustrates the basic two-level framework, in which each job in the system has its own task scheduler, and there is only one OS allocator that plays a centralized role in distributing the processors. The OS allocator and the task schedulers interact with each other in a quantum-by-quantum fashion, and such interaction is an essential part of two-level adaptive scheduling and is called *request-allocation protocol* [89]. At the starting time t_q of each quantum q , the task scheduler for each job $J_i \in \mathcal{J}$ calculates its processor desire $d_i(q)$, and reports it to the OS allocator. The centralized OS allocator then based on the desires of all jobs and its scheduling policy distributes the P processors to the jobs, and each job J_i gets a share $a_i(q)$ of processors, which does not change during quantum q , i.e., $a_i(t) = a_i(q)$ for $t \in [t_q, t_q + L)$. Moreover, the allocations of all jobs should apparently satisfy $\sum_{i=1}^n a_i(q) \leq P$. Upon receiving the allotment $a_i(q)$ for job J_i , the task scheduler executes the job, and measures certain characteristics during its execution in quantum q , which may directly or indirectly re-

flect the performance of the job with allocation $a_i(q)$. Finally, based on such execution characteristics, the task scheduler calculates the new processor desire $d_i(q+1)$ for the job in the next quantum $q+1$.

To ease analysis, we make a couple of simplifying assumptions about any two-level adaptive scheduler. First of all, we assume that fractional processor allocation is possible, that is, $a_i(q)$ can be non-integral. This assumption has been commonly made in multiprocessor scheduling literature [71, 70, 73, 88, 173]. The fractional allocation can be achieved by time-sharing a processor among several jobs. Secondly, when a new job is released in the middle of a scheduling quantum, it is not scheduled until the beginning of the next quantum. Similarly, when an existing job completes execution in the middle of a quantum, it is not considered complete until the end of the quantum. This assumption allows much cleaner analysis, and in the worst case will only increase the performance bounds of our algorithms by constant additive factors, which can be ignored if the jobs under consideration are sufficiently large.

1.3 Dynamic Speed Scaling

We employ *dynamic speed scaling*, or *dynamic voltage/frequency scaling* (DVFS) [185, 41, 84, 99, 6] in this thesis to change the processor speeds, which is usually achieved by adjusting simultaneously the voltage and the frequency on some modern processors. Besides power-down mechanisms [14, 99, 6, 111], this technique has become one of the most widely used approaches to control the power consumptions, and hence to balance the performance and energy usage on variable speed processors.

It has been observed that for most CMOS-based processors, the dynamic power satisfies the *cube-root* rule, that is, the power consumption of a processor at any time is proportional to s^3 when it runs at speed s [41, 143]. However, as with most researchers so far [7, 43, 21, 117, 48, 49, 174], we assume a more general model in this thesis. Under the general model, the power consumption satisfies s^α , where $\alpha > 1$ is the power parameter and the speed s can take any value in $[0, \infty)$.² As this power function is strictly convex, the total energy usage when executing a job can be significantly reduced by slowing down the speeds of the processors allocated to the job at the expense of the job's performance. Therefore, how to appropriately employ dynamic speed scaling on variable speed processors, in particular how to assign the processor speeds at any time to balance performance and energy poses another challenge for the design of efficient adaptive schedulers.

We consider two scenarios depending on whether a scheduling algorithm must al-

²We should point out that this general power consumption model is still a much simplified one compared to reality. In practice, the speed of a processor should be bounded from both below and above, and usually the speed can only be adjusted to a set of discrete values pre-configured by the chip manufacturer. Nevertheless, this simplified model is quite commonly assumed in the algorithmic community, and various scheduling algorithms developed based on it have in fact inspired the design of other energy-efficient algorithms with more practical models [17, 6, 114, 18].

locate processors of the same speed to a parallel job or it can allocated processors with different speeds to a job at any given time. Specifically, we say that a scheduling algorithm uses *uniform* speed scaling if it always assigns the same speed for all processors allocated to a job; otherwise, we say that the algorithm uses *non-uniform* speed scaling if processors of different speeds can be allocated to the job. Note that non-uniform speed scaling requires that the speed of each processor can be individually scaled, which is possible on some modern multiprocessor systems [94, 112]. It turns out such per-processor speed scaling capability plays an essential role in the design of energy-efficient non-clairvoyant scheduling algorithms, as we will see in Chapter 6 of the thesis.

1.4 Job Model and Scheduling Model

In this section, we formally describe the job model and the scheduling model. We also define some notations that will be used throughout this thesis.

Many parallel job models exist in the scheduling literature, including the DAG (directed acyclic graph) model [3, 88], the parallelism profile model [39, 63], and the speedup function model [70, 160]. In particular, the DAG model specifies the detailed precedence constraints of the tasks in a parallel job; the parallelism profile model gives the number of processors that can be effectively utilized by a job at any time of execution; and the speedup function generalizes the parallelism profile model by specifying the execution rate of a job at any time as an arbitrary (usually non-decreasing and sub-linear) speedup function. Although these models differ in terms of the granularity or the level of details in representing parallel applications, all scheduling algorithms proposed in this thesis can be shown to work uniformly on these different models with little modification. Moreover, our experiences with these job models, which are also supported by many results in the literature (see, e.g., [62, 63, 71, 70, 88, 160, 173, 91, 172]), suggest that the theoretical performances of the scheduling algorithms are not affected by the choice of the job models. Hence, in this thesis, we choose the parallelism profile model, which appears to be more intuitive and analytically amenable in terms of understanding the executions of the jobs.

We adopt the notations introduced by Edmonds et al. [71, 70, 73, 48], which models a parallel job with time-varying parallelism using multiple phases of speedup functions. However, instead of modeling the phases by arbitrary speedup functions, we choose the model where each phase has a linear speedup function up to a certain degree of parallelism. Hence, it is equivalent to the parallelism profile model [39, 63]. Specifically, each job J_i contains k_i phases $\langle J_i^1, J_i^2, \dots, J_i^{k_i} \rangle$, and each phase J_i^k , where $1 \leq k \leq k_i$, has an amount of *work* w_i^k and a *linear speedup function* Γ_i^k up to a certain parallelism h_i^k , where $h_i^k \geq 1$. Suppose that at any time t , job J_i is in its k -th phase and a scheduling algorithm allocates to the job $a_i(t)$ processors that need not have the same

speed. We assume that the execution of the job at time t is then based on the *maximum utilization policy* [100, 24], which always utilizes faster processors before slower ones until the total number of utilized processors exceeds the parallelism of the job. In particular, let $s_{ij}(t)$ denote the speed of the j -th processor allocated to job J_i at time t , and we can assume without loss of generality that $s_{i1}(t) \geq s_{i2}(t) \geq \dots \geq s_{ia_i(t)}(t)$. Then, only $\bar{a}_i(t) = \min\{a_i(t), h_i^k\}$ fastest processors are effectively utilized, and the speedup or the execution rate of the job at time t is given by $\Gamma_i^k(a_i(t)) = \sum_{j=1}^{\bar{a}_i(t)} s_{ij}(t)$. In the case that all processors allocated to job J_i share the same speed $s_i(t)$, the speedup is then simply given by $\Gamma_i^k(a_i(t)) = \bar{a}_i(t)s_i(t)$. The *span* l_i^k of phase J_i^k , which is a convenient parameter representing the time to execute the phase with h_i^k or more processors of unit speed, is given by $l_i^k = w_i^k/h_i^k$. We say that phase J_i^k is *fully-parallel* if $h_i^k = \infty$ and it is *sequential* if $h_i^k = 1$. Moreover, if job J_i consists of only sequential and fully-parallel phases, we call it (PAR-SEQ)* job [160]. Let r_i denote the *release time* of job J_i . If all jobs are released together in a *batch*, then their release time can be assumed to be 0. Otherwise, we can assume without loss of generality that the first released job arrives at time 0. Let c_i^k denote the *completion time* of the k -th phase of job J_i , and let $c_i = c_i^{k_i}$ denote the completion time of job J_i under a scheduling algorithm. We require that the job must be completed in finite amount of time and any of its phase cannot begin unless all its preceding phases have been completed, i.e., $r_i = c_i^0 \leq c_i^1 \leq \dots \leq c_i^{k_i} = c_i < \infty$, and $\int_{c_i^{k-1}}^{c_i^k} \Gamma_i^k(a_i(t))dt = w_i^k$ for all $1 \leq k \leq k_i$. Finally, for each job J_i , we define its *total work* to be $w(J_i) = \sum_{k=1}^{k_i} w_i^k$ and define its *total span* to be $l(J_i) = \sum_{k=1}^{k_i} l_i^k$.

1.5 Objectives Functions

We apply various objective functions as the metrics to evaluate the performance and the energy consumption of our scheduling algorithms under different scenarios.

The *response time* $R(J_i)$ of any job J_i is the duration between its completion time and release time, i.e., $R(J_i) = c_i - r_i$. The *total response time* $R(\mathcal{J})$ of all jobs in job set $\mathcal{J}$ is then given by $R(\mathcal{J}) = \sum_{J_i \in \mathcal{J}} R(J_i)$, and the makespan $M(\mathcal{J})$ of the job set is the completion time of the last completed job, i.e., $M(\mathcal{J}) = \max_{J_i \in \mathcal{J}} c_i$. Job J_i is said to be *active* at time t if it is released but not completed at t , i.e., $r_i \leq t \leq c_i$. Hence, an alternative expression for the total response time of the job set is given by $R(\mathcal{J}) = \int_0^\infty n_t dt$, where n_t denotes the number of active jobs at time t . Let $u_i(t)$ denote the power consumption of job J_i at time t , i.e., $u_i(t) = \sum_{j=1}^{a_i(t)} s_{ij}(t)^\alpha$. The energy consumption e_i of the job is then given by $e_i = \int_0^\infty u_i(t)dt$, and the total energy consumption $E(\mathcal{J})$ of the job set is $E(\mathcal{J}) = \sum_{i=1}^n e_i$, or alternatively $E(\mathcal{J}) = \int_0^\infty u_t dt$, where $u_t = \sum_{J_i \in \mathcal{J}} u_i(t)$ denotes the total power consumption of all jobs at time t .

First of all, we consider total response time $R(\mathcal{J})$ and makespan $M(\mathcal{J})$ for a set $\mathcal{J}$ of jobs on fixed speed processors. Both objectives are widely used metrics

in the scheduling literature: the former often measures the average waiting time of all users in the system while the latter is closely related to the throughput of the system. In addition, we consider the set response time $F(\mathcal{J})$ for a collection $\mathcal{J}$ of job sets on fixed speed processors. This objective can be used to measure the waiting time of all users who submit simultaneously more than one jobs into the system and it will be formally introduced in Chapter 5. Finally, we consider total response time plus energy, i.e., $G(\mathcal{J}) = R(\mathcal{J}) + E(\mathcal{J})$, as well as makespan plus energy, e.g., $H(\mathcal{J}) = M(\mathcal{J}) + E(\mathcal{J})$, for a set $\mathcal{J}$ of jobs on variable speed processors. These two objectives provide means to measure both performance and energy consumption of a scheduling algorithm and have drawn much attention recently in energy-efficient scheduling community [21, 117, 115, 116, 18, 48, 49, 10, 83].³

Although energy and time have different units, optimizing a combination of the two can be naturally interpreted by looking at both objectives from a unified point of view. Specifically, suppose that a user is willing to spend one unit of energy in order to reduce certain units, say γ units, of total response time or makespan. Then, by changing the units of time and energy, we can assume without loss of generality that $\gamma = 1$. In fact, minimizing sum of conflicting objectives is quite common in many bi-criteria optimization problems. For example, even in the scheduling literature, similar metrics have been considered previously that combine both performance and cost of scheduling as the objective functions [179, 167, 54].

1.6 Performance Evaluations

We evaluate the performances of our scheduling algorithms in this thesis using both theoretical analysis and empirical simulations. This section gives a brief introduction to the analysis tools and describes our simulation setup.

1.6.1 Competitive Analysis

We employ *competitive analysis* [36] to study our scheduling algorithms. Competitive analysis is one of the most widely used techniques to evaluate the theoretical performance of an online algorithm. In particular, it compares the performance of an online algorithm with that of an optimal offline scheduler, which uses the same amount of resources. The online algorithm is said to be *c-competitive* with respect to a particular

³In this thesis, we study the performance and energy tradeoff by optimizing a single aggregate objective function. We should point out that there are other means to study such multi-objective optimization problems. One commonly used approach is to restrict one objective while optimizing the other. Alternatively, one can use the Pareto-compliant ranking method to find all non-dominated solutions, in which one objective cannot be further improved without sacrificing the other. These two approaches have also been applied previously in energy-efficient scheduling literature. For example, Pruhs et al. [157, 156] applied the first approach to minimize total response time or makespan for a set of jobs subject to an energy budget. Bunde [43] found all non-dominated solutions with respect to makespan and energy for scheduling sequential jobs on a single processor.

metric if its performance is not more than c times that of an optimal offline scheduler for any input instance. For example, an online scheduling algorithm A is said to be c -competitive with respect to the total response time if we have $R_A(\mathcal{J}) \leq c \cdot R^*(\mathcal{J})$ for any job set $\mathcal{J}$, where $R_A(\mathcal{J})$ and $R^*(\mathcal{J})$ denotes the total response time of $\mathcal{J}$ incurred by algorithm A and an optimal offline scheduler, respectively. If the competitive ratio c is a constant, i.e., $O(1)$, then we say that the online algorithm achieves *competitive* performance. Apparently, our goal is to design scheduling algorithms with competitive ratios as small as possible. If an online algorithm has a small competitive ratio, it means that this algorithm will definitely have good performance in practice.

However, one problem with competitive analysis is that it often leads to large competitive ratios or strong lower bounds for some online algorithms while in practice these algorithms can actually perform quite well. The reason is because competitive analysis only considers the worse-case performance of an algorithm on some inputs designed by an *adversary* upon knowing the moves of the algorithm. However, in practice these inputs do not appear as often. This has motivated alternative techniques for analyzing online algorithms, and one particular technique is the *resource augmentation analysis* [104, 152].

Using resource augmentation analysis, the performances of an online algorithm and an optimal offline scheduler are similarly compared, but the online algorithm is allowed to have more resources than the optimal. For example, the online scheduling algorithm A is now said to be *s -speed c -competitive* with respect to the total response time if its performance using s times more processor resources is not more than c times that of an optimal offline scheduler. The rationale for resource augmentation is that the extra resources given to the online algorithm can now compensate for their lack of knowledge on the worst-case inputs. Hence, if an algorithm is able to achieve competitive performance with moderate increase in the amount of resources, then it is likely to perform well on most practical workloads. Our goal here is to design scheduling algorithms that can achieve competitive performance using minimal extra resources possible. For convenience, we assume in this thesis that the optimal offline scheduler always uses unit-speed processors and if resource augmentation is required, the online algorithm is given processors of speed s , where $s > 1$.

1.6.2 Empirical Simulations

To further evaluate our scheduling algorithms and to compare their performances with other schedulers, in particular the EQUI-based ones, we also conduct empirical simulations. EQUI (Equi-Partitioning) [180] is a simple two-level adaptive scheduler that at any time divides the total number of processors equally among all active jobs in the system. However, this simple strategy has been shown to offer efficient performances under many relevant settings [71, 70, 160, 161, 174, 176]. Hence, in this thesis, we use EQUI as a baseline scheduler against which our scheduling algorithms

are compared.

In two-level adaptive scheduling, the processor allocations of the jobs are dynamically changed according to the load of the system and/or the parallelism variations of the jobs. Therefore, it will be helpful to evaluate the schedulers on malleable jobs with changing parallelism characteristics. In this thesis, we construct such jobs based on a traditional workload model, namely Downey’s model [67] and augment it with a set of internal parallelism variations.

In particular, Downey’s workload model generates non-malleable parallel jobs with external information such as their total work, arrival time, average parallelism, etc., which are derived from workload logs in two supercomputer centers, namely San Diego Supercomputer Center and Cornell Theory Center.⁴ Internally, we divide a parallel job into a series of segments with the same length and each segment is identified by a specific parallelism structure. Instead of using completely random parallelism structures, which does not allow clear account of the feedback-allocation process of a two-level adaptive scheduler, we identify five generic forms of parallelism variations, which are specified by *Step*, *Impulse*, *Ramp*, *Poly(I)*, and *Poly(II)* profiles respectively as shown in Figure 1.3 and they describe precisely how the parallelism of a segment evolves with time. These profiles provide a comprehensive coverage of the changing parallelism dynamics and reflect a wide range of parallel programming patterns. For instance, the Step profile can describe constant and stable parallelism over a period of time from typical data-parallel sections of a job. The Impulse profile on the other hand can emulate a drastic one-off increase in parallelism typically encountered in, e.g., a short FOR loop. The Ramp and the two Poly profiles, which are constructed by polynomials of degree 1, 3, and 1/3 in our simulation, can model changes in the job’s parallelism with different rates of spawning and joining parallel threads.

Moreover, to maintain consistency with the original non-malleable jobs, we ensure that the work and the average parallelism of a job adhere to those initially generated by Downey’s model. In particular, the average parallelism of each segment is chosen uniformly according to that of the original model, and no more segment is added after the aggregate work reaches that initially generated. For each segment, we construct its parallelism structure by randomly selecting one out of the five profiles. However, we always construct the Step profile first, which is used as a blueprint to derive other profiles in case they have been selected. This ensures that all profiles are coherent with each other, as shown in Figure 1.3, where the five different parallelism profiles share the same work and span, hence the same average parallelism. In all simulations

⁴We should point out that Downey’s model may not accurately reflect the characteristics of all realistic workloads. For example, it assumes that the work and the average parallelism of a job follow the uniform-log distribution while other forms of heavy-tail distributions have been observed in some studies [77]. In addition, jobs are assumed to arrive into the system according to the Poisson process while many other job arrivals actually exhibit a very bursty nature [171]. In fact, workload characterization by itself is still an evolving area of research. Since Downey’s model is based on a set of common assumptions and is derived from real measurements, it provides an reasonable basis for us to generate malleable parallel jobs.

Figure 1.3: Five different parallelism structures specified by Step, Impulse, Ramp, Poly(I) and Poly(II) profiles.

shown in this thesis, the quantum length is set to $1/5$ of the segment length and the number of processors is set to 64.

1.7 Statement of Results

In this thesis, we primarily focus on non-clairvoyant scheduling for malleable parallel jobs on multiprocessors using the two-level framework and dynamic speed scaling. Our main study is centered around two specific two-level adaptive schedulers, namely ACDEQ and AGDEQ. In particular, ACDEQ is proposed to improve upon the existing scheduler AGDEQ in terms of the stability of its processor desire calculation. The main results are summarized in the following:

- We present a stable task scheduler A-CONTROL and combine it with the OS allocator DEQ (Dynamic Equi-partitioning) [139] to form a two-level scheduler ACDEQ. We show using control-theoretic analysis that the processor desires calculated by A-CONTROL achieve much improved transient and steady-state performances that the existing task scheduler A-GREEDY and hence the two-level scheduler AGDEQ fail to attain. Specifically, ACDEQ achieves BIBO-stable, zero steady-state error, zero overshoot and a user-configurable convergence rate.
- We show that on processors of fixed speeds both ACDEQ and AGDEQ achieve $O(1)$ -competitive with respect to the makespan for a set of jobs regardless of the jobs' arrival pattern. In addition, both algorithms achieve $O(1)$ -competitive with respect to the total response time if all jobs are released in a batch; for non-batched jobs, they achieve $O(1)$ -speed $O(1)$ -competitive with respect to the total

response time. Moreover, we present frameworks for analyzing both makespan and total response time of any two-level adaptive scheduler.

- We show that the ACDEQ and AGDEQ algorithms when combined the EQUI scheduler achieve $O(1)$ -competitive with respect to the set response time for a collection of job sets if all job sets are released in a batch; for nonbatched job sets, both algorithms achieve $O(1)$ -speed $O(1)$ -competitive with respect to the set response time. Furthermore, we also provide a framework for analyzing the set response time of the EQUI $\circ$ Y family of algorithms. Again, the results are obtained on fixed speed processors.
- We show that, on processors whose speeds can be dynamically scaled, any non-clairvoyant algorithm that employs uniform speed scaling performs poorly, or specifically $\Omega(P^{(\alpha-1)/\alpha^2})$ -competitive, with respect to the total response time plus energy, where P is the total number of processors. Hence, to obtain better results, we present the N-ACCEQ and N-AGCEQ algorithms, which use respectively the variants of ACDEQ and AGDEQ as well as a particular non-uniform speed scaling policy.
- We show that both N-ACCEQ and N-AGCEQ achieve $O(\ln^{1/\alpha} P)$ -competitive with respect to the total response time plus energy for a set of jobs if all jobs are released in a batch; for nonbatched jobs, they achieve $O(\ln P)$ -competitive with respect to the total response time plus energy. Moreover, we show a lower bound of $\Omega(\ln^{1/\alpha} P)$ on the competitive ratio of any non-clairvoyant algorithm. Again, a framework is given for analyzing the total response time plus energy of any algorithm X that works with the non-uniform speed scaling policy.
- Our empirical study confirms the theoretical analysis of the ACDEQ and AGDEQ algorithms and their variants in different settings. In particular, the simulation results show that the task scheduler A-CONTROL indeed exhibits better transient and steady-state performances than A-GREEDY. Moreover, ACDEQ-based schedulers also outperform AGDEQ-based ones with respect to the various metrics. In addition, schedulers based on both algorithms outperform simple EQUI-based schedulers under a wide range of workloads.

We also study semi-clairvoyant scheduling in this thesis and show that algorithms of this kind significantly improve upon their non-clairvoyant counterparts on variable speed processors. The results are summarized in the following:

- We present the U-CEQ algorithm, which combines the CEQ (Conservative Equipartitioning) algorithm with a particular uniform speed scaling policy. We show that U-CEQ achieves $O(1)$ -competitive with respect to the total response time plus energy for a set of jobs regardless of the jobs' arrival pattern.

- We present the P-FIRST (Parallel-First) algorithm and show that it achieves $O(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy for any set of batched (PAR-SEQ)* jobs. Moreover, we provide a lower bound of $\Omega(\ln^{1-1/\alpha} P)$ on the competitive ratio of any semi-clairvoyant algorithm in this setting.

1.8 Synopsis

The remaining parts of this thesis are organized as follows. Chapter 2 reviews related results in the literature on adaptive scheduling as well as those that optimize various metrics. Chapter 3 presents our task scheduler A-CONTROL and the two-level scheduler ACDEQ, followed by its control-theoretic analysis and algorithmic studies for a single job. Chapter 4 analyzes both ACDEQ and AGDEQ algorithms with respect to the total response time and makespan for a set of jobs. The set response time performances of ACDEQ and AGDEQ based algorithms are analyzed in Chapter 5. Chapter 6 starts to consider both performance and energy consumption on processors whose speeds can be dynamically scaled. The N-ACCEQ and N-AGCEQ algorithms are presented and analyzed with respect to the total response time plus energy. Chapter 7 considers semi-clairvoyant scheduling by presenting and analyzing the U-CEQ algorithm for total response time plus energy as well as the P-FIRST algorithm for makespan plus energy. Finally, Chapter 8 concludes the thesis and discusses some open questions.

Chapter 2

Literature Review

In this chapter, we review the literature on adaptive scheduling, as well as results that minimize various objective functions. While adaptive scheduling appears to be a relatively new field of study, scheduling in general has been one of the most extensively studied areas in computer science and operational research. Since 1950s, hundreds if not thousands of paper have been written that minimize various objectives, such as the total response time or the makespan, under different models and assumptions. Hence, any survey on scheduling will not be exhaustive. In this chapter, therefore, besides studying the relevant work on adaptive scheduling, we will also review some recent online scheduling results that are related to the objectives considered in this thesis. Readers can refer to the surveys, e.g., [124, 119, 108] or the books, e.g., [57, 42, 153] for the more classical results on offline scheduling.

We organize our survey in this chapter into two major parts. In the first part, we focus on adaptive scheduling by reviewing related work in this area from three different perspectives, namely, algorithmic, control-theoretic and empirical perspectives. Although the results from different perspectives may vary in terms of the performance objectives and the scheduling strategies, etc., they all concern primarily about adaptive scheduling of shared system resources, and hence belong to the same scheduling paradigm. In the second part, we review the literature on online scheduling that minimizes various objective functions, including total response time, makespan, set response time and some energy-related objectives. We consider a wide range of results that span from single processor scheduling, to scheduling sequential jobs on multiprocessors, to parallel job scheduling on multiprocessors. When reviewing the related work in this chapter, we will also point out our contributions in the corresponding context. Any result that forms part of this thesis will be highlighted by an asterisk (*) at the beginning of its reference for all tables shown in the chapter.

2.1 Adaptive Scheduling

In this section, we will review related work on adaptive scheduling from three different perspectives, namely, algorithmic, control-theoretic and empirical perspectives. We mainly focus in this thesis on the scheduling of multiprocessor resources in an adaptive fashion, which generates most algorithmic and empirical results. In the control-theoretic literature, however, the related work tends to cover a much broader area than multiprocessor scheduling. However, as with multiprocessor scheduling, these results also concern primarily about adaptive scheduling of some shared system resources, and therefore belong to the same scheduling paradigm. Hence, we will review these different results in this section and the various techniques used therein from a broader resource management context.

2.1.1 Algorithmic Perspective

The algorithmic studies on adaptive scheduling can be dated back to the early 1990s, when space-sharing on multiprocessor systems have become possible and increasingly popular. In particular, the EQUI (Equi-partitioning) [180] algorithm proposed by Tucker and Gupta in 1989 and the DEQ (Dynamic Equi-partitioning) [139] algorithm proposed by McCann, Vaswani and Zahorjan in 1993 have triggered a series of algorithmic studies on the two schedulers and their variants in the following two decades. Both algorithms adjust processor allocations to jobs in an online adaptive manner according to the load of the system. Readers can refer to Chapters 3 and 5 of this thesis for more formal descriptions of the two algorithms. In this section, we will review related results on both algorithms and their variants.

Results on EQUI and Its Variants

Although many theoretical papers (see, e.g., [183, 181, 182, 135]) in the early 1990s still focused on non-adaptive scheduling or offline scheduling, these results, especially the techniques and lower bounds, established in these research have laid the foundation for the subsequent studies of online adaptive scheduling. Some of the most prominent results in the next 20 years are devoted to analyzing the EQUI algorithm and its variants [71, 70, 73, 72, 160, 159, 69]. Specifically, Edmonds et al. [71] first showed that the EQUI algorithm, which simply divides all available processors equally among the active jobs at any time, performs no worse than $(2 + \sqrt{3}) \approx 3.73$ times that of an optimal offline scheduler with respect to the total response time of any batched job set. In that paper, they also provided a lower bound of $e \approx 2.72$ on the competitive ratio of any non-clairvoyant online algorithm for batched jobs. Subsequently, Edmond [70] showed that for nonbatched jobs, EQUI achieves $(2 + \frac{4}{\epsilon})$ -competitive with respect to the total response time if it is equipped with $(2 + \epsilon)$ times more resources than an optimal offline scheduler for any $\epsilon > 0$. He also gave a lower bound of $\Omega(\sqrt{n})$ on the

competitive ratio of any non-clairvoyant algorithm for nonbatched jobs, where n is the total number of jobs. Robert and Schabanel [162] generalized the analysis of EQUI to parallel jobs with precedence constraints among the tasks of a job and each task may have time-varying parallelism. They showed that the EQUI◦EQUI algorithm, which equally divides the available processors to all active jobs, and within each job, equally divides the allocated processors to all ready tasks, essentially achieved identical total response time performance as shown in [70], i.e., $(2 + \epsilon)$ -speed $(2 + \frac{4}{\epsilon})$ -competitive.

No non-clairvoyant algorithm has beaten EQUI with respect to the total response time until Edmonds and Pruhs [73] proposed the LAPS_β (Latest Arrival Processor Sharing) algorithm. LAPS_β is a variant of the EQUI algorithm that works for non-batched jobs by equally dividing all available processors to a β fraction of the active jobs at any time with the latest arrival times. They showed that LAPS_β achieves *fully scalable* performance with respect to the total response time, i.e., it is $O(1)$ -competitive with slightly more resources than the optimal. Specifically, LAPS_β is $(1 + \beta + \epsilon)$ -speed $(\frac{4(1+\beta+\epsilon)}{\beta\epsilon})$ -competitive with respect to the total response time. However, one problem with LAPS_β is that it is only fully scalable if the parameter β is known before the extra resources are given, that is, the amount of extra resources depends on β . In a recent manuscript [69], Edmonds claimed that no deterministic non-clairvoyant algorithm can achieve $O(1)$ -competitive with respect to the total response time of parallel jobs if it is given $(1 + \epsilon)$ times more resources than the optimal for any $\epsilon > 0$. In addition, we can observe that when the parameter β is set to 1, LAPS_β becomes exactly EQUI, but the performance bound of LAPS_β increases by a factor of two than that of EQUI. This is because both results [70, 73] essentially relied on a reduction process, which transforms any set of parallel jobs with arbitrary parallelism structures into special instances of parallel jobs with only sequential and fully-parallel phases. However, the result in [70] made an additional reduction to ensure that the online algorithm after the transformation is never ahead of the optimal on the execution of any job, which was not assumed in [73], hence resulting in the blowup of the bound. Robert and Schabanel [159] proved in a recent follow-up article that this property can in fact be preserved in LAPS_β as well. Therefore, it can be shown to achieve $(1 + \beta + \epsilon)$ -speed $(\frac{2(1+\beta+\epsilon)}{\beta\epsilon})$ -competitive with respect to the total response time, hence matching exactly the bound of EQUI when β is set to 1.

Edmonds, Datta and Dymond [72] applied the analysis of EQUI to the congestion control algorithm used in TCP (Transmission Control Protocol), and showed that TCP achieves $O(1)$ -competitive with respect to the user-perceived latency when it is given $O(1)$ times more bandwidth and adjustment time than the optimal congestion control algorithm. Furthermore, Robert and Schabanel [160] have analyzed the EQUI algorithm with respect to the makespan for any set of batched jobs. They showed that EQUI is $O(\frac{\ln n}{\ln \ln n})$ -competitive with respect to the makespan, where n is the total number of jobs in the system. Moreover, they provided a matching lower bound of

Table 2.1: Some algorithmic results on EQUI and its variants.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Objective</i>	<i>Job type</i>	<i>Setting</i>
Edmonds et al. [71]	EQUI	$2 + \sqrt{3}$	total response time	batch	non-clairv.
Edmonds et al. [71]	online	e (lower bound)	total response time	batch	non-clairv.
Edmonds [70]	EQUI	$(2 + \epsilon)$ -speed $(2 + \frac{4}{\epsilon})$ -comp.	total response time	arbitrary	non-clairv.
Edmonds [70]	online	$\sqrt{n}$ (lower bound)	total response time	arbitrary	non-clairv.
Robert et al. [162]	EQUI $\circ$ EQUI	$(2 + \epsilon)$ -speed $(2 + \frac{4}{\epsilon})$ -comp.	total response time	precedence within jobs	non-clairv.
Edmond et al. [73]	LAPS $_{\beta}$	$(1 + \beta + \epsilon)$ -speed $(\frac{4(1+\beta+\epsilon)}{\beta\epsilon})$ -comp.	total response time	arbitrary	non-clairv.
Robert et al. [159]	LAPS $_{\beta}$	$(1 + \beta + \epsilon)$ -speed $(\frac{2(1+\beta+\epsilon)}{\beta\epsilon})$ -comp.	total response time	arbitrary	non-clairv.
Edmonds [69]	online	$(1 + \epsilon)$ -speed $\omega(1)$ -comp.	total response time	arbitrary	non-clairv.
Robert et al. [160]	EQUI	$O(\frac{\ln n}{\ln \ln n})$	makespan	batch	non-clairv.
Robert et al. [160]	online	$\Omega(\frac{\ln n}{\ln \ln n})$	makespan	batch	non-clairv.
Edmonds et al. [72]	TCP	$O(1)$ -bandwidth $O(1)$ -comp.	user-perceived latency	Internet requests	non-clairv.

$\Omega(\frac{\ln n}{\ln \ln n})$ on the competitive ratio of any non-clairvoyant algorithm for batched jobs with arbitrary sizes, hence suggesting that EQUI is optimal in this setting.

Table 2.1 summarizes the relevant results regarding the EQUI algorithm and its variants.

Results on DEQ and Its Variants

We now consider the other famous adaptive algorithm DEQ. Compared to the non-clairvoyant algorithm EQUI, which allocates the processors solely based on the number of active jobs in the system at any time, DEQ is semi-clairvoyant since it was originally proposed to allocate processors based on both the number of active jobs and their instantaneous parallelism. Brecht, Deng and Gu [39] first analyzed the DEQ algorithm and showed that it is $(2 - \frac{1}{P})$ -competitive with respect to the makespan for any set of jobs with arbitrary release time, where P is the total number of processors in the system. This result matches the makespan lower bound given by Shmoys, Wein and Williamson [168] for any online semi-clairvoyant algorithm that does not know the remaining work of the jobs, hence suggesting that DEQ is optimal with respect to makespan. Deng et al. [63] later showed that DEQ is $(4 - \frac{4}{n+1})$ -competitive with respect to the total response time for any batched set of parallel jobs, and $(2 - \frac{2}{n+1})$ -competitive if each parallel job contains a single phase only. In particular, the latter result also matches the total response time lower bound given by Motwani, Phillips and Torng [142] for batched sequential jobs, hence again suggesting that DEQ is optimal in this setting. Recently, He, Sun and Hsu [90] showed that DEQ in fact achieves $(3 - \frac{2}{n+1})$ -competitive with respect to the total response time for batched parallel jobs with multiple phases, thus beating the corresponding bound of EQUI, which knows nothing about the jobs' parallelism.

All the results above on the DEQ algorithm assumed the possibility of semi-clairvoyance, that is, the instantaneous parallelism of the jobs are indeed available

for processor allocation. However, in many applications, even such information is not accessible to the online scheduling algorithm, in which case one can revert back to the more restrictive non-clairvoyant setting. Besides the famous EQUI algorithm, another type of non-clairvoyant algorithms is the two-level adaptive schedulers considered in this thesis. The algorithmic study of two-level adaptive scheduling started with a paper by Agrawal et al. [3], which was inspired by a desire estimation heuristic reported in the Master thesis of Sen [165] for the MIT Cilk runtime system [31]. Prior to that, Arora, Blumofe and Plaxton [11] designed a task scheduler that can adapt to the variation in the allocated processors of a job, but does not actively provide feedback to the OS allocator. In contrast, Agrawal et al. [3] proposed the A-GREEDY task scheduler, which estimates the processor desires of a parallel job with a multiplicative-increase multiplicative-decrease strategy and feeds back the desire to the OS allocator after the expiration of each scheduling quantum. Following the design of A-GREEDY, Agrawal, He and Leiserson [5] also proposed another task scheduler A-STEAL to work in conjunction with the work-stealing scheduler [32] employed in the Cilk runtime. They showed that both A-GREEDY and A-STEAL achieve nearly linear speedup and waste a small fraction of processor resources for an individual job.

He, Hsu and Leiserson [88] later combined the task schedulers A-GREEDY and A-STEAL with the OS allocator DEQ to form two-level adaptive schedulers. They proved that the resulting algorithms AGDEQ and ASDEQ are both $O(1)$ -competitive with respect to the makespan for a set of jobs regardless of the jobs' arrival pattern, as well as $O(1)$ -competitive with respect to the total response time for batched jobs, provided that the jobs under consideration are sufficiently large. In addition, He, Hsu and Leiserson [89] also showed that when fractional processor allocation is not allowed and the system is heavily-loaded, i.e., the number of jobs is more than the number of processors, the two-level algorithms can be coupled with the RR (Round Robin) scheduler to form the GRAD and WRAD algorithms, which achieve similar results with respect to makespan and total response time. For nonbatched jobs, Sun, Cao and Hsu [173] showed that AGDEQ also achieves $O(1)$ -competitive with respect to the total response time when it is given $O(1)$ times more resources than the optimal, again for sufficiently large jobs. Finally, observing that the desire estimation strategy of A-GREEDY can cause unstable processor desires even if the parallelism of the job is constant, Sun and Hsu [178] proposed ABG (Adaptive B-Greedy) task scheduler, which was later renamed to A-CONTROL in [175]. Compared to the discrete strategy of A-GREEDY, the desire estimation of A-CONTROL relies on principles from control theory, which guarantees stability along with other desirable control-theoretic properties. In [178, 173, 175], the algorithmic performances of ACDEQ, which combines A-CONTROL with the DEQ algorithm, were also analyzed with respect to the makespan and the total response time. Basically, ACDEQ achieves similar results as AGDEQ for both total response time and makespan, provided that the jobs under consideration have

Table 2.2: Some algorithmic results on DEQ and its variants.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Objective</i>	<i>Job type</i>	<i>Setting</i>
Brent et al. [40]	DEQ	$2 - \frac{1}{P}$	makespan	arbitrary	semi-clairv.
Shmoys et al. [168]	online	$2 - \frac{1}{P}$ (lower bound)	makespan	batch seq.	semi-clairv.
Deng et al. [63]	DEQ	$4 - \frac{4}{n+1}$	total response time	batch	semi-clairv.
He et al. [90]	DEQ	$3 - \frac{2}{n+1}$	total response time	batch	semi-clairv.
Deng et al. [63]	DEQ	$2 - \frac{2}{n+1}$	total response time	batch with single phase	semi-clairv.
Motwani et al. [142]	online	$2 - \frac{2}{n+1}$ (lower bound)	total response time	batch seq.	semi-clairv.
He et al. [88]	AGDEQ ASDEQ	$O(1)$	makespan	sufficiently large	non-clairv.
He et al. [88]	AGDEQ ASDEQ	$O(1)$	total response time	sufficiently large, batch	non-clairv.
He et al. [89]	GRAD WRAD	$O(1)$	makespan	sufficiently large	non-clairv. heavy loads
He et al. [89]	GRAD WRAD	$O(1)$	total response time	sufficiently large, batch	non-clairv. heavy loads
*Sun et al. [173]	AGDEQ	$O(1)$ -speed $O(1)$ -comp.	total response time	sufficiently large	non-clairv.
*Sun et al. [178]	ACDEQ	$O(1)$	makespan	smooth parallelism	non-clairv.
*Sun et al. [178]	ACDEQ	$O(1)$	total response time	batch, smooth parallelism	non-clairv.
*Sun et al. [173]	ACDEQ	$O(1)$ -speed $O(1)$ -comp.	total response time	smooth parallelism	non-clairv.

smooth parallelism variations.

Table 2.2 summarizes the relevant results of the two-level schedulers as well as those regarding the DEQ algorithm.

2.1.2 Control-Theoretic Perspective

Classical control theory has been successfully deployed for decades in electrical and mechanical engineering. Recently, it has also been applied to some computing systems (see, e.g., [150, 97, 110, 192, 2, 125, 106, 151]). With a set of well-established and systematic design principles that can guarantee certain performance requirements [107, 92, 93], many systems that rely on control theory have demonstrated good performances in their respective application domains. In this subsection, we review some control-theoretic results on adaptive scheduling.

Compared to the algorithmic results, control-theoretic studies on adaptive scheduling have a broader application domain than multiprocessor scheduling, which includes web server scheduling [129, 184], database server load balancing [65, 66] and computing resource sharing [127, 109], etc. In addition, the control-theoretic studies tend to focus more on the transient and steady-state performances of the system instead of the algorithmic measures, such as makespan or total response time. Furthermore, most control-theoretic designs follow a set of well-established principles, which covers identifying system parameters, selecting control law, and determining controller gains. We will review some commonly used techniques in each stage of the design process.

System identification involves modeling the input-output relationship of the system under control. In the case of adaptive scheduling, it means modeling the relationship between the resource requirement (e.g., processor desire) and the system output. This relationship is often called *transfer function* in control theory. For some systems, the transfer function can be obtained by directly analyzing the system from first principles [130, 133]. For most systems, however, directly analyzing the input-output relationship can be difficult. In such cases, black-box modeling is often necessary [127, 184, 129, 132]. One widely used model is the ARMA (Auto-Regressive Moving-Average) model [93, 37], which relates the current system output to the past input(s) and output(s). The number of past inputs and outputs made use of is referred to as the *order* of the model. The higher the order, the more accurate the model is for the system. However, the order is usually kept as low as possible since increasing the order may not improve the accuracy significantly, but will introduce additional complexity into the parameter estimation, which are usually realized through linear regression techniques on the experimental data, such as LS (Least Square), or RLS (Recursive Least Square) if the parameters are estimated online. Once the system model is obtained, the transfer function can be expressed through z-transform [79, 93].

Control Laws

While the transfer function of the system is determined by its input-output relationship, the transfer function of the controller is determined by the control law, which specifies how the controller output is computed based on the difference (error) between the desirable output (reference) and the actual system output. One type of commonly used control law follows the popular PID control paradigm [12], whose controller output consists of a proportional (P), an integral (I) and a derivative (D) part expressed in term of the system error. However, despite the popularity and success of PID controllers in controlling industrial processes, most computing systems just implement a subset of PID for simplicity and robustness. In particular, the derivative part of the control is very rarely used due to its sensitivity to system noises. In contrast, controllers that implement P, or I, or PI are commonly seen in the adaptive scheduling literature. While most existing work [66, 127, 184, 149, 130, 129] uses PID controller, alternative control laws are also occasionally applied. For example, in [80], gain compensation is used to specify the control law in scheduling real-rate multimedia applications. In [109], the control law is derived from the optimization of a quadratic cost function to schedule shared computing services. An adaptive filter function is used in [132] to specify the control law when allocating web caches for QoS differentiations. However, despite these alternatives, PID remains the most popular control paradigm for specifying control laws. Again, the transfer function of the controller is obtained through z-transform on the control law.

Performance Specifications

Once the transfer functions of the system and the controller are specified, the closed loop transfer function can be obtained. The set of poles, which are the roots of the closed loop system, determines the system's transient and steady-state behaviors. We review four performance specifications [93] that are commonly used in control theory as the criteria for distinguishing good designs from bad ones.

(1) *Stability*. The system is BIBO (Bounded-Input Bounded-Output) stable if its output is bounded for bounded reference. Sometimes, the system is only considered as stable if the output stabilizes eventually rather than oscillates around some value, which is called non-oscillating stability. Apparently, the latter requirement for stability is more stringent than the former one. However, non-oscillating stability can also be achieved if the system is BIBO stable and at the same time has constant steady-state error. Thus, BIBO stability itself is an important performance measure, which is used in most work as the only consideration for stability. The system is BIBO stable if all the poles are within the unit circle on the z -plane.

(2) *Steady-state Error*. The steady-state error of the system refers to the error between the system output and the reference after the system has settled down for sufficiently long time, that is, at steady state. Steady-state error measures how precisely the controller controls the system output in tracking the reference input, which is often computed with the help of the *final value theorem* [93] in the z -domain. If the chosen control law involves an integral part, the systems under control will have zero steady-state error, which is usually desirable for most systems.

(3) *Maximum Overshoot*. The maximum overshoot of the system refers to the maximum error between the transient output and the steady-state output of the system. For first-order closed loop systems, the maximum overshoot is zero if the only pole is greater than zero. Otherwise, maximum overshoot is equal to the magnitude of the pole. For higher order systems, the maximum overshoot can usually be approximated by considering the dominant pole of the closed-loop system, which is the pole with the largest magnitude.

(4) *Settling Time*. The settling time of the system refers to the time it takes the system output to reach within a certain percentage of the steady-state value. It is a measure of how fast the output converges to the steady state, and is usually determined by the dominant pole. Alternatively, the rate of convergence represents a similar performance measure. It indicates how much the error in a quantum has reduced in terms of its ratio to the error of the previous quantum. In this case, the rate of convergence is approximately equal to the magnitude of the dominant pole.

Controller Design Strategies

We discuss some major controller design strategies that are commonly seen in the adaptive scheduling literature. While each strategy provides a way to design the

controller and to determine the controller gains in particular, some strategies can be used in conjunction with others to achieve more effective control of the system. For instance, self-tuning regulator can be used with either pole placement or linear quadratic design to determine the controller gains. Table 2.3 summarizes the examples given in this subsection in terms of the strategies in each stage of their design.

(1) *Pole Placement.* Pole placement [93] is one of the most widely used controller design strategy. It selects the values for the controller gains in such a way that the closed-loop system poles are placed at specified locations, and hence the system satisfies the desired performance specifications. For simple first-order and sometimes second-order systems, it is often straightforward to choose the controller gains for desirable pole locations [127, 130, 133]. For higher-order systems, however, this task is not so trivial. In such cases, methods such as Root Locus [129] can be used to trace the locations of the poles on the z -plane as the controller gains change.

Lu et al. [129] designed a PI controller to control the relative delays in web servers. The output in this case is the relative connection delays between two service classes. Using system identification, the input-output relationship of the server is modeled as a second-order system using ARMA model. The values of the controller gains are determined using pole placement strategy with the help of Root Locus method.

(2) *Linear Quadratic Design.* Linear quadratic design (LQD) [79] is an optimal control strategy that determines the controller gains or the control laws by minimizing a function that often takes the control costs into account as well. In our context, the control costs correspond to the overhead of resource reallocations. This scheme is often used in MIMO (Multiple-Input Multiple-Output) systems, and can be formulated as a constraint optimization problem. Efficient numerical methods exist to compute the resource requirements with specific control laws [79, 92]. While linear quadratic design has the advantage of taking control costs into consideration, it also suffers from some apparent drawbacks, such as the requirement of prior knowledge of the system model and the difficulty of choosing the appropriate parameters in the cost function.

Diao et al. [65, 66] designed a load balancing controller for balancing the memory pools in database servers. The output is the response time benefit, and the objective is to minimize the overall response time for certain types of customers, which is equivalent to adjusting the memory resources such that each type of customers experience the same response time benefit. A PI control law was specified for the controller output and linear quadratic design was used to consider both the cost of transient load imbalance and the cost of memory pools reallocations.

(3) *Self-Tuning Regulator.* Self-tuning regulator (STR) [13] is an adaptive control strategy applied to systems with time-varying dynamics. Therefore, the controller gains in self-tuning regulators are designed to vary with time as well to compensate for the time-varying dynamics of the system, which are often estimated online using RLS. The latest estimates are provided to the self-tuning regulator to determine its

controller gains for each quantum based on standard design techniques, such as pole placement or linear quadratic design [127, 109, 149, 132].

Liu et al. [127] designed an adaptive controller to dynamically adjust the computation resources in resource containers with virtualization technology. The output is the mean response time of a web server application. Self-tuning regulator was used with pole placement to adjust the gains of a PI controller such that the server can achieve desired mean response time even with variations in the workload. Karlsson, Zhu and Karamanolis [109] combined self-tuning regulator with linear quadratic design when scheduling shared computing services. The output is either throughput or latency. The input-output relationship is estimated online using RLS with exponential forgetting factor and the estimated system parameters are used in a control law derived from optimizing a quadratic cost function that penalizes large resource requirement. Sun and Hsu [178] applied self-tuning regulator to schedule parallel jobs on multiprocessors, where the output is the estimated processor desire of a job. The gain of an I controller was determined by pole placement and the system was modeled by a simple linear relation.

(4) *Gain Scheduling.* Gain scheduling [13] is an adaptive control strategy that designs multiple controllers and several submodels, and switches among them when the system operates outside of the applicable region of the current active submodel, due to, for example, changes in the workload [184, 130, 128]. Note that gain scheduling is different from self-tuning regulator because the latter strategy can only handle systems whose model parameters may change but the model itself does not change. Gain scheduling provides a way to properly control the systems under completely different operating regions and models. Switching of the controllers are usually triggered by detecting the variations in the operating region.

Wang, Zhu and Singhal [184] applied gain scheduling to control separately the mean response time and the resource utilization in a shared server environment when the system is in overload and underload region, respectively. In the overload region, the system is modeled by a first-order ARMA model and the parameters are estimated by RLS. A PI controller was designed to control the mean response time. In the underload region, the system is represented analytically by a simple nonlinear model, which is linearized for an I controller to adjust the resource utilization.

(5) *MIMO Control.* MIMO (Multiple-Input Multiple-Output) control [79, 93] is a technique to collectively control more than one system and their outputs simultaneously. Compared with controlling multiple SISO (Single-Input Single-Output) systems independently, MIMO control has the advantage of capturing the relations among different systems, such as the task interdependencies and performance correlations. It is often used in other control strategies, such as linear quadratic design or model predictive control [45], etc., where the MIMO controller computes the resource requirements considering both the total resource constraints and the interdependencies

Table 2.3: Some control-theoretic results on adaptive scheduling as well as their application domains and design strategies.

<i>Reference</i>	<i>Applications</i>	<i>System Identification</i>	<i>Control Law</i>	<i>Design Strategy</i>
Lu et al. [129]	Guaranteeing relative delay in web servers	ARMA model (with LS)	PI controller (with Root Locus)	Pole placement
*Sun et al. [178]	Estimating processor desires on multiprocessors	Simple linear model	I controller	Pole placement
Diao et al. [65, 66]	Balancing response time in database servers	Analytical model	PI controller	MIMO control + LQD
Liu et al. [127]	Controlling mean response time in resource containers	ARAM model (with RLS)	PI controller	Pole placement + STR
Karlsson et al. [109]	Maintaining throughput and latency in shared computing services	ARMA model (with RLS and exponential forgetting factor)	By optimizing a quadratic cost function	MIMO control + STR + LQD
Wang et al. [184]	Controlling mean response and utilization in shared servers	ARMA model (with RLS) for mean response time + Analytical model (with linearization) for utilization	PI controller for response time + I controller for utilization	Gain scheduling + Pole placement + STR

and correlations among systems [65, 66, 109, 131].

Diao et al. [65, 66] applied MIMO control with linear quadratic design to balance memory pools in database servers. The same technique was used by Karlsson, Zhu and Karamanolis [109] in scheduling shared computing services, in which they also applied self-tuning regulator within MIMO control to estimate parameters of the entire systems.

2.1.3 Empirical Perspective

In this section, we review some empirical studies on adaptive scheduling. As with the review on algorithmic studies, we only focus on those results that are directly applicable to multiprocessor scheduling, while many results (see, e.g., [26, 148, 25, 101]) also exist that apply to scheduling other environments, such as Grids or data centers. There are mainly two types of empirical studies, which are based on experimental evaluations and simulations, respectively. Both approaches have their advantages. The former approach directly reflects the realistic setting by taking all aspects of the machines into account, such as their memory and the cache behaviors, which are usually not implemented in most simulations. The latter approach can often simulate larger machines under more variety of workloads that may not be possible in actual experiments. We will review separately the related work based on each approach.

Experimental Evaluations

The experimental evaluations of adaptive scheduling on space-shared multiprocessors also started around 1990s, when algorithms such as EQUI or DEQ were first proposed. In particular, Tucker and Gupta [180] evaluated the EQUI algorithm on a 16-processor Encore Multimax machine running a variant of 4.2 BSD Unix operating system. They mainly focused on the process control policy they proposed for adapting to processor changes in parallel applications. The workloads consist of several applications implemented using Brown University Threads Package, including the parallel FFT, merge-

sort, Gaussian elimination and matrix multiplication algorithms. They concluded that all applications executed much faster with processor control than without. McCann, Vaswani and Zahorjan [139] compared the performances of the DEQ algorithm with those of EQUI and some static scheduling policies on a Sequent Symmetry Machine consisting of 20 Intel 80386 processors, running the DYNIX 3.0.16 operating system. The workloads include four applications with different parallelism structures, namely MATRIX, PVERIFY, MVA, and GRAVITY that solve some industrial and scientific problems. They showed that adaptive scheduling is preferable to static scheduling, and that preemption of processors incurs insignificant overhead if they are coordinated carefully.

Nguyen, Vaswani and Zahorjan [144, 146, 145] proposed an adaptive task scheduler, called Self-tuning, to schedule jobs with iterative parallel loops, whose speedup curves often increase at first with the number of allocated processors and then start to decrease due to communication overheads and load imbalance. These loops usually exhibit similar speedups over different iterations as the amount of allocated processors vary. Then the problem of finding the optimal number of allocated processors becomes a single variable optimization problem. Self-tuning finds the optimal allocation using the method of golden section [50] that iteratively narrows the searching interval until the optimal is found. Experiments were conducted on a 60-processor KSR-2 machine running a variant of the OSF/1 Unix operating system, and the workloads consist of mix of hand-coded applications from the SPLASH [169] and SPLASH-2 [188] benchmarks and compiler-parallelized applications from the PERFECT Club [28] benchmarks. It was shown that Self-tuning provides a fairly accurate estimation of the jobs's speedup curves on these applications, and when combined with the DEQ algorithm, it outperforms the simple EQUI scheduler.

Corbalán, Martorell and Labarta [59, 61, 60] also focused on scheduling jobs with iterative parallel loops. In contrast to [144, 146, 145], they estimated the execution efficiency of an application with respect to the allocation processors in each loop iteration. They proposed a run-time library, called SelfAnalyzer, to estimate the program efficiency. In addition, a performance-driven processor allocation policy, PDPA, was also proposed to dynamically allocate the processors to each application based on the efficiency estimation of SelfAnalyzer. Specifically, PDPA allocates the processors using an additive-increase additive-decrease strategy, depending on whether the estimated efficiency is above or below certain thresholds. Again, they showed that the proposed algorithm outperforms EQUI. In this work, the experiments were carried out on an SGI Origin2000 machine with 64 processors running the IRIX operating system, and the workloads contain applications implemented using OpenMP from the SPECfp95 and the NAS [103] benchmarks.

Blumofe and Papadopoulos [35] evaluated the performance of a non-blocking implementation of the work stealing scheduler proposed by Arora, Blumofe and Plaxton

Table 2.4: Some experimental results on adaptive scheduling with implementation environments and workloads.

<i>Reference</i>	<i>Algorithm</i>	<i>Environment</i>	<i>Workloads</i>
Tucker et al. [180]	EQUI	16-processor Encore Multimax with a variant of 4.2 BSD Unix	FFT, sort, Gauss elimination, matrix multiplication
McCann et al. [139]	DEQ	20-processor Sequent Symmetry with DYNIX 3.0.16	MATRIX, PVERIFY, MVA, GRAVITY
Nguyen et al. [144, 146, 145]	Self-tuning	60-processor KSR-2 with a variant of OSF/1 Unix	SPLASH, SPLASH-2 and PERFECT Club Benchmarks
Corbalán, et al. [59, 61, 60]	SelfAnalyzer + PDPA	64-processor SGI Origin2000 with IRIX	SPECfp95 and NAS benchmarks
Blumofe et al. [35]	non-blocking work-stealing	8-processor Sun Ultra Enterprise 5000 with Solaris 2.5.1	Hood applications
Sen [165]	Cilk-AP	16-processor SGI Origin 2000 with version 6.5 of IRIX	Cilk applications
Sundarsan et al. [158]	ReSHAPE	50-node homogeneous cluster System X	matrix multiplication, LU, Jacobi solver, FFT

[11] on a Sun Ultra Enterprise 5000 machine with 8 UltraSPARC processors running Solaris 2.5.1. This task scheduler, which was called ABP in [4] after the initials of the authors, can efficiently utilize the processors allocated to an application by the OS allocator, even if the number of allocated processors can grow and shrink over time. By running a set of applications implemented in Hood [34], a C++ user-level threads library and runtime system, they showed that ABP outperforms the static implementation of the work stealing scheduler [33] in non-dedicated and multiprogrammed environments. Sen [165] also considered an adaptive version of the working stealing scheduler, which eventually inspired the later developments in [3, 4, 5]. The desire estimation heuristic in its task scheduler, called WSAP (Work Stealing Adaptive Parallel), takes a much complicated form than the simple multiplicative-increase multiplicative-decrease strategy in [5]. Running a set of applications implemented in Cilk [31] on an SGI Origin 2000 SMP with 16 processors and version 6.5 of the IRIX operating system, he showed that Cilk-AP, which combines WSAP with DEQ, incurs negligible overheads and provides significant improvements in multiprogramming environments compared to the original Cilk implementation.

Sundarsan and Ribbens [158] proposed a framework called ReSHAPE, which supports adaptive scheduling of parallel MPI applications executed on distributed memory platforms. As with many other work on adaptive scheduling, ReSHAPE actively monitors the execution status of the running applications, and it reallocates the processors among the applications based on the performance monitor as well as some pre-defined policies. The experiments were conducted on 50 nodes of a large homogeneous cluster called System X, and the workloads consist of several MPI applications, including the Matrix multiplicative, LU fractionation, Jacobi solver and FFT. Again, substantial improvements in the application performances were observed in ReSHAPE compared to the static scheduling environment.

Table 2.4 summarizes the experimental results on adaptive scheduling surveyed in this section with their implementation environments and workloads.

We now review the empirical studies on adaptive scheduling that are based on simulations. As mentioned previously, simulation is an important evaluation technique that allows systematic studies of various scheduling algorithms usually in larger scale processors and under more variety of workload conditions. However, an important challenge with adaptive scheduling simulations is to generate suitable synthetic workloads, i.e., parallel jobs with time-varying parallelism, to effectively capture the patterns of actual parallel programs and their running behaviors. Some early results on static and adaptive scheduling (see, e.g., [136, 68, 166, 123, 137, 55, 40]) generated synthetic jobs based on a few simple parameters such as a job’s average parallelism and variance of parallelism, etc. Nevertheless, the way jobs were created in these results have greatly influenced the subsequent research on parallel workload modeling as well as adaptive scheduling simulations.

According to the survey by Feitelson [76], parallel jobs can be modeled in three different levels with increasing flexibility in terms of the processor allocations. In particular, the jobs with least flexibility are *rigid* jobs, which require a fixed number of processors throughout the jobs’ execution. With more flexibility, *moldable* jobs can run on an arbitrary number of processors at launch time but the number of processors cannot be changed afterwards. The most flexible jobs are *malleable* in that they can run on an arbitrary number of processors dynamically at runtime. Feitelson [74] maintains a parallel workload archive that contains a collection of workload logs from various supercomputer centers around the world, as well as workload models derived from these logs. However, due to the difficulty of obtaining detailed parallelism information, especially the internal parallelism variations of the applications, all of these workloads only model rigid jobs [75, 102, 134] and moldable ones [67, 56]. In particular, Downey [67] provided a moldable job model that estimates the speedup of a parallel job as a function of its average parallelism and variance. He only provided two simple hypothetical parallelism profiles with low and high variance in parallelism, respectively, thus lacking detailed information about the internal parallelism. Based on the statistical analysis of a survey concerning many users’ experiences with parallel machines like IBM SP2, Cirne and Berman [56] provided a more comprehensive moldable job model by taking the partition size of the workload into account. This model uses a similar speedup function as Downey’s [67], and also does not consider the internal parallelism variations. To model the internal job structures, a flexible hierarchical model was proposed by Calzarossa et al. [44] and further extended by Feitelson and Rudolph [78]. The idea is that at the higher level, a rigid job model is generated from actual workload logs, which provides the external characteristics for the parallel jobs such as their arrival patterns, work requirements, and average parallelism, etc. Further processing at the lower level adds internal structures to the jobs while maintaining their external properties. Unfortunately, Feitelson and Rudolph

only provided a rough framework on the construction of such malleable jobs while leaving the detailed implementation unexplained. Cao et al. [46] followed the footsteps of [44, 78] and provided a specific procedure on the construction of malleable jobs using the moldable job model by Downey [67] and a set of generic parallelism variation curves.

With the malleable job model provided by Cao et al. [46], Sun, Cao and Hsu [175, 172] conducted simulations to evaluate the performance of the ACDEQ algorithm and to compare it with AGDEQ under a wide range of workload conditions. The simulation results confirm the superior performance of ACDEQ for both an individual job and a set of jobs. In addition, they also compared the performances of the EQUI $\circ$ ACDEQ and the EQUI $\circ$ AGDEQ algorithms for sets of parallel jobs, which essentially leads to the same conclusion that EQUI $\circ$ ACDEQ performs better.

Prior to the results of Sun, Cao and Hsu [175, 172], Agrawal, He and Leiserson [4] conducted simulations to evaluate their adaptive task scheduler A-STEAL and compare it ABP [11]. In the simulations, simple fork-join jobs were created with alternating sequential and parallel phases to assemble different data-parallel programs. The parallelism of each phase as well as their lengths were chosen from many distributions, include the uniform distribution and different types of heavy-tailed distributions. However, the authors observed that the results, which showed that A-STEAL significantly outperforms ABP in terms of the running time and processor waste of an individual job, were fairly insensitive to the distributions chosen.

Using similar settings as in [4], He, Hsu and Leiserson [89] also evaluated the performance of the two-level adaptive scheduler AGDEQ with respect to the makespan and the total response time for a set of jobs. They showed that under a variety of workloads the simulated performance of ACDEQ is much better than the worst-case bounds of the algorithm predicted by the theoretical analysis. In addition, Sun and Hsu [178] also compared ACDEQ with AGDEQ using simple fork-join jobs, and they again showed that ACDEQ outperforms AGDEQ under the same setting as in [89]. All the simulations shown in [4, 89, 178, 175, 172] were conducted using DESMO-J [64], a Java-based discrete-event simulation tool.

In an early study, Zahorjan and McCann [191] evaluated a variant of the adaptive algorithm DEQ using parallel jobs created based on the MVA (Mean Value Analysis) structure shown in [139] as well as two types of fork-join structures. Different job mixes with varying parameters were used in the simulation. They concluded that compared with static scheduling, adaptive scheduling exhibits better performance when the reallocation overhead is small and the system load is high.

Brecht and Guha [40] evaluated the performance of a proportional processor allocation policy, which is based on the estimates of both work and efficiency of executing a parallel job. In particular, they compared this policy with the EQUI scheduler using simple moldable jobs specified by work and an execution rate function. They showed

Table 2.5: Some simulation results on adaptive scheduling with simulation environments and workloads.

<i>Reference</i>	<i>Algorithm</i>	<i>Environment</i>	<i>Workloads</i>
Zahorjan et al. [191]	DEQ variant	20 processors, simulator implemented in C++	MVA and fork-join jobs
Brecht et al. [40]	proportional policy	100 processors	synthetic moldable jobs with work and execution rate function
Agrawal et al. [4]	A-STEAL	128, 512, 1000 processors simulator in Java (DESMO-J)	fork-join jobs with varying parallelism and span
He et al. [89]	AGDEQ	128 processors simulator in Java (DESMO-J)	fork-join jobs with varying parallelism and span
*Sun et al. [178]	ACDEQ	128 processors simulator in Java (DESMO-J)	fork-join jobs with varying parallelism and span
*Sun et al. [175, 172]	ACDEQ	64 processors simulator in Java (DESMO-J)	malleable jobs with internal parallelism variations
Weissman et al. [187]	iScheduler	64 processors	STEN, GE, CL and synthetic jobs based traces

that the proposed policy based on a single estimate of either work or efficiency does not provide significant benefit over EQUI while substantial improvement was observed when an accurate estimation of both parameters are possible.

Finally, Weissman, Abburi and England [187] proposed an adaptive scheduler called iScheduler to effectively reallocate processors among parallel applications during runtime based on the performance prediction by the Prophet system [186]. The simulations included workloads constructed from actual parallel applications such as iterative Jacobi solver (STEN), Gaussian elimination (GE) and parallel gene sequence comparison (CL), as well as synthetic parallel jobs based on the traces in various supercomputing centers [74]. It was shown that on both types of workloads, iScheduler can dramatically outperform traditional static scheduling policies.

Table 2.5 summarizes the simulation results surveyed in this section including the simulations environments and parallel workloads.

2.2 Minimizing Various Objectives

In the second part of this chapter, we will give a brief survey on the existing results that minimize various objective functions considered in this thesis, which include total response time, makespan, set response time and some energy-related objectives. As mentioned previously, due to the vast literature in the field, we will only restrict our attention to deterministic online algorithms, whose results are obtained using worst-case competitive analysis on identical processors. Readers can refer to the recent surveys [155, 154] and books [122, 163] for those results that consider alternative approaches, such as randomization or average-case analysis, or other machine models, such as uniformly related or unrelated processors.

2.2.1 Total Response Time

The total response time for a set of jobs is also known as the total flow time, which is equivalent to the mean response time or average flow time used elsewhere in some publications. Since Section 2.1.1 has already provided a survey on minimizing the total response time as well as the makespan for parallel jobs, we will focus on sequential jobs in this subsection and the next for both single processor and multiprocessor systems. Moreover, we consider both clairvoyant and non-clairvoyant settings depending on whether the work of a job is known to the online algorithm upon the job's arrival into the system.

First of all, on a single processor, it is well known that the SJF (Shortest Job First) algorithm is optimal for batched jobs and the SRPT (Shortest Remaining Processing Time) algorithm is optimal for nonbatched jobs [170]. Both algorithms are clairvoyant since at any time they select the job with the least remaining work to run. For the non-clairvoyant setting, Motwani, Phillips and Torng [142] showed that the RR (Round Robin) algorithm, which is equivalent to EQUI by giving each active job an equal share of processor at any time, is $(2 - \frac{2}{n+1})$ -competitive for batched jobs, and it is the best ratio possible in this setting. For jobs with arbitrary release time, however, RR performs poorly. Specifically, Matsumoto [138] showed that RR is $\Omega(n/\lg n)$ -competitive in this case, where n is the total number of jobs. Kalyanasundaram and Pruhs [105] further showed that even modest resource augmentation cannot make RR achieve $O(1)$ -competitive. In particular, they showed that with speed $s = 1 + \epsilon$, where $0 < \epsilon < 1$, RR is $\Omega(n^{1-\epsilon})$ -competitive. In addition, Motwani, Phillips and Torng [142] also gave a lower bound of $\Omega(n^{1/3})$ on the competitive ratio of any non-clairvoyant algorithm for nonbatched jobs. Finally, Kalyanasundaram and Pruhs [104] provided a fully scalable algorithm SETF (Shortest Elapsed Time First), which at any time shares the processor equally among those jobs that have received the least processing time so far, and they showed that SETF is $(1 + \epsilon)$ -speed $(1 + 1/\epsilon)$ -competitive. Berman and Coulston [27] improved this ratio for larger speed processor by showing that SETF is s -speed $2/s$ -competitive for $s \geq 2$.

We now turn to multiprocessor scheduling. For the clairvoyant setting, it was known that SJF, which always assigns the job with the least work to the next available processor, is still the optimal algorithm for batched jobs [57]. When jobs have arbitrary release time, Leonardi and Raz [121] showed that SRPT, which at any time selects P jobs with the least remaining work to run, is $O(\min(\log \Delta, \log n/P))$ -competitive, where P is the number of processors and Δ is the work ratio of the largest job over the smallest job in the job set. They also proved that this ratio matches asymptotically the lower bound, hence is tight. Phillips, et al. [152] showed using resource augmentation that SRPT is $(2 - 1/P)$ -speed 1-competitive, which was improved later by McCullough and Torng [140] to s -speed $1/s$ -competitive for any $s \geq 2 - 1/P$. Since SRPT requires migration of the jobs among the processors, it may not be possible to implement in

Table 2.6: Some results on sequential job scheduling that minimize total response time.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Proc#</i>	<i>Job release</i>	<i>Setting</i>
Smith [170]	SJF	1	1	batch	clairv.
Smith [170]	SRPT	1	1	arbitrary	clairv.
Motwani et al. [142]	RR	$2 - \frac{2}{n+1}$	1	batch	non-clairv.
Motwani et al. [142]	online	$2 - \frac{2}{n+1}$ (lower bound)	1	batch	non-clairv.
Matsumoto [138]	RR	$\Omega(n/\lg n)$	1	arbitrary	non-clairv.
Kalyanasundaram et al. [105]	RR	$(1 + \epsilon)$ -speed ($0 < \epsilon < 1$) $\Omega(n^{1-\epsilon})$ -comp.	1	arbitrary	non-clairv.
Motwani et al. [142]	online	$\Omega(n^{1/3})$	1	arbitrary	non-clairv.
Kalyanasundaram et al. [104]	SETF	$(1 + \epsilon)$ -speed ($\epsilon > 0$) $(1 + 1/\epsilon)$ -comp.	1	arbitrary	non-clairv.
Berman et al. [27]	SETF	s -speed ($s \geq 2$) $2/s$ -comp.	1	arbitrary	non-clairv.
Conway et al. [57]	SJF	1	P	batch	clairv.
Leonardi et al. [121]	SRPT	$O(\min(\log \Delta, \log n/P))$	P	arbitrary	clairv.
Leonardi et al. [121]	online	$\Omega(\min(\log \Delta, \log n/P))$	P	arbitrary	clairv.
Phillips, et al. [152]	SRPT	$(2 - 1/P)$ -speed 1-comp.	P	arbitrary	clairv.
McCullough et al. [140]	SRPT	s -speed ($s \geq 2 - 1/P$) $1/s$ -comp.	P	arbitrary	clairv.
Avrahami et al. [15]	IMD + SRPT	$O(\min(\log \Delta, \log n))$	P	arbitrary	clairv. w/o migration
Chekuri et al. [51]	IMD + SRPT	$(1 + \epsilon)$ -speed ($\epsilon > 0$) $O(1 + 1/\epsilon)$ -comp.	P	arbitrary	clairv. w/o migration
Motwani et al. [142]	RR	$\left(2 - \frac{2P}{n+P}\right)$	P	batch	non-clairv.
Motwani et al. [142]	online	$\left(2 - \frac{2P}{n+P}\right)$ (lower bound)	P	batch	non-clairv.

some systems. Avrahami and Azar [15] developed a non-migratory algorithm, called IMD (Immediate Dispatching) that dispatches an incoming job immediately to a processor with the least cumulative load in terms of the job's class. The jobs assigned to the same processor are then scheduled using SRPT on that processor without migration. They showed that IMD with SRPT is $O(\min(\log \Delta, \log n))$ -competitive. Using resource augmentation, Chekuri et al. [51] showed that IMD with SRPT also achieves fully scalable performance, that is, $(1 + \epsilon)$ -speed $O(1 + 1/\epsilon)$ -competitive. All the results above are based on SRPT, which is clairvoyant in nature. For the non-clairvoyant setting, Motwani, Phillips and Torng [142] showed that the RR algorithm, which at any time ensures that all active jobs have received the same amount of processing time, is $\left(2 - \frac{2P}{n+P}\right)$ -competitive for batched jobs and again this is the best ratio possible. For nonbatched jobs, however, no result on deterministic non-clairvoyant algorithms is known, to the best of our knowledge. Nevertheless, Becchetti and Leonardi [23] and Chekuri et al. [51] have developed efficient randomized algorithms that achieve respectively the optimal asymptotic performance and fully scalable performance.

Table 2.6 summarizes the results that minimize the total response time for sequential jobs under different settings.

2.2.2 Makespan

We consider the objective of minimizing makespan for a set of jobs in this subsection. It is obvious that on a single processor, any work-conserving algorithm, that is, any algorithm which does not idle the processor whenever there are jobs to run, is optimal with respect to the makespan, and this holds regardless of the jobs' arrival pattern.

Hence, we mainly focus on scheduling multiprocessors.

One of the most important results in multiprocessor scheduling was due to Graham [81] in 1966, which is now believed by many to be the first ever result on competitive analysis. He proposed a simple LIST scheduling algorithm, which chooses any available job and executes it till completion whenever a processor becomes idle. It was shown that LIST achieves $(2 - 1/P)$ -competitive with respect to the makespan, where P is the total number of processors. Note that this particular result can be applied to many different scenarios, including offline, online clairvoyant and non-clairvoyant settings, and even when jobs can have precedence constraints. In the non-clairvoyant setting, Shmoys, Wein and Williamson [168] showed that this ratio is the best possible. That is, no non-clairvoyant algorithm can achieve competitive ratio smaller than $(2 - 1/P)$, even if all jobs are batch released and preemptions of the jobs are allowed. In the clairvoyant setting, the problem becomes much easier to deal with and various improvements can be shown. It turns out that the ability to do preemption, that is, to interrupt a job at any time and later resume it possibly on a different processor, makes a big difference on the quality of the solutions. Specifically, if preemption is allowed, McNaughton [141] proposed a simple algorithm that successively fills the processors with the jobs in any order up to the maximum of the two obvious lower bounds. He showed that this algorithm, which we call SF (Successive Filling), is optimal with respect to the makespan when jobs are batched. Hong and Leung [98] later showed that a variant of SF that refills the processors whenever a new job arrives is also optimal for jobs with arbitrary release time. However, if preemption is not allowed, then optimal solutions cannot be guaranteed. The best algorithm in this case is the LPT (Longest Processing Time) algorithm, which is a variant of LIST that chooses the available job with the most work to run whenever a processor becomes idle. Graham [82] showed that LPT is $(\frac{4}{3} - \frac{1}{3P})$ -competitive for batched jobs, and this bound is tight for the algorithm. If jobs can have arbitrary release time, Chen and Vestjens [52] showed that LPT is $3/2$ -competitive. They also gave a lower bound of 1.3473 for any online algorithm in this setting.

Table 2.7 summarizes the results that minimize the makespan for sequential jobs under various settings.

2.2.3 Set Response Time

The objective of set response time for a collection of job sets was recently introduced by Robert and Schabanel [161] in 2007 as a subproblem for their study on pull-based broadcast scheduling. Since then, there have been only three paper [161, 160, 172] that address this objective, all of which concern parallel jobs on multiprocessors. Moreover, all three papers assumed that the jobs within each set are released at the same time. First of all, Robert and Schabanel [161] proposed the EQUIOY algorithm, which divides the total number of processors equally among all active job sets and within each set

Table 2.7: Some results on sequential job scheduling that minimize makespan.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Proc#</i>	<i>Job type</i>	<i>Setting</i>
Well-known	work-conserving	1	1	arbitrary	arbitrary
Graham [81]	LIST	$2 - 1/P$	P	arbitrary	non-clairv.
Shmoys et al. [168]	online	$2 - 1/P$ (lower bound)	P	batch	non-clairv.
McNaughton [141]	SF	1	P	batch	clairv. with preemption
Hong et al. [98]	SF variant	1	P	arbitrary	clairv. with preemption
Graham [82]	LPT	$\frac{4}{3} - \frac{1}{3P}$	P	batch	clairv. w/o preemption
Chen et al. [52]	LPT	3/2	P	arbitrary	clairv. w/o preemption
Chen et al. [52]	online	1.3473 (lower bound)	P	arbitrary	clairv. w/o preemption

uses any algorithm Y to schedule the jobs. The only requirement for Y is that it never idles the allocated processors, that is, it has to allocate all processors among the jobs at any time. They showed that when the job sets can have arbitrary release time and each job consists of only two phases, namely, a sequential phase followed by a fully-parallel one, which they call a SEQ-PAR job, $\text{EQUI} \circ Y$ achieves $O(1)$ -speed $O(1)$ -competitive with respect to the set response time. Robert and Schabanel [160] also considered the setting where all job sets are released at the same time and each job can have arbitrary parallelism profile. They proposed the $\text{EQUI} \circ \text{EQUI}$ algorithm, which at any time divides the total number of processors equally among all active job sets and within each set divides the allocated processors equally among the active jobs. It was shown that $\text{EQUI} \circ \text{EQUI}$ achieves $O(\frac{\ln n}{\ln \ln n})$ -competitive with respect to the set response time in this setting, where n is the maximum number of jobs among all sets. Moreover, they also proved an asymptotically matching lower bound, hence suggesting that $\text{EQUI} \circ \text{EQUI}$ is optimal up to constant factor. Sun, Cao and Hsu [172] combined the two-level adaptive scheduling algorithm AGDEQ with EQUI, and showed that $\text{EQUI} \circ \text{AGDEQ}$ achieves $O(1)$ -competitive for batched job sets and $O(1)$ -speed $O(1)$ -competitive for nonbatched job sets, hence improving upon the results of [161, 160] for sufficiently large jobs. Moreover, they also provided a generalized analysis framework for the $\text{EQUI} \circ Y$ algorithm family and showed that $\text{EQUI} \circ \text{ACDEQ}$ achieves similar performance as $\text{EQUI} \circ \text{AGDEQ}$ when jobs have smooth parallelism variations.

Table 2.8 summarizes the results that minimize the set response time for parallel jobs.

2.2.4 Energy-Related Objectives

We now review related results in the literature that optimize some energy-related objectives. For the purpose of this thesis, we only focus on those results that concern total response time and makespan for a set of jobs. Readers can refer to [189, 19, 20, 7, 8, 16] for sample results as well as surveys [99, 6] of related work with objectives

Table 2.8: Some results on parallel job scheduling that minimize set response time.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Proc#</i>	<i>Job type</i>	<i>Setting</i>
Robert et al. [161]	EQUIOY	$O(1)$ -speed $O(1)$ -comp.	P	SEQ-PAR	non-clairv.
Robert et al. [160]	EQUIOEQUI	$O(\frac{\ln n}{\ln \ln n})$	P	batch	non-clairv.
Robert et al. [160]	online	$\Omega(\frac{\ln n}{\ln \ln n})$	P	batch	non-clairv.
*Sun et al. [172]	EQUIOAGDEQ	$O(1)$ -speed $O(1)$ -comp.	P	sufficiently large	non-clairv.
*Sun et al. [172]	EQUIOAGDEQ	$O(1)$	P	sufficiently large, batch	non-clairv.
*Sun et al. [172]	EQUIOACDEQ	$O(1)$ -speed $O(1)$ -comp.	P	smooth parallelism	non-clairv.
*Sun et al. [172]	EQUIOACDEQ	$O(1)$	P	batch, smooth parallelism	non-clairv.

that involve temperature or deadlines.

We start with energy-efficient scheduling for total response time. Pruhs, Uthaisombut and Woeginger [156] first considered this problem where the objective is to minimize total response time of the jobs subject to a constraint on the total amount of energy consumed. They proposed an efficient offline algorithm that finds an optimal schedule up to a user desired precision for unit work jobs, as well as an $O(1)$ -approximation algorithm with $(1+\epsilon)$ times more energy than the optimal for arbitrary work jobs. Bunde [43] later showed that, even for unit work jobs, the exact solution cannot be expressed using standard arithmetic operations, including extraction of roots. Bunde also extended the algorithm of [156] to multiprocessors setting. Chen et al. [53] proposed an optimal offline solution to this problem for the case where all jobs are batch released and have arbitrary work.

Albers and Fujiwara [7] first proposed combining total response time and energy consumption into a single objective function. They analyzed a batched variation of the natural online speed scaling algorithm that always sets the power equal to the number of active jobs, and showed that this variation algorithm is $8.3e \left(\frac{3+\sqrt{5}}{2}\right)^\alpha$ -competitive with respect to total response time plus energy for unit work jobs, where $\alpha > 1$ denotes the power parameter. Bansal, Pruhs and Stein [21] improved the competitive ratio of this problem to 4 by using the natural speed scaling algorithm together with the SJF policy. They also proposed an $O\left(\frac{\alpha^2}{\log^2 \alpha}\right)$ -competitive algorithm for jobs with arbitrary work. Lam et al. [117] showed that the natural speed scaling algorithm, which they called AJC (Active Job Count), with the SRPT policy improves the competitive ratio for jobs with arbitrary work to $O\left(\frac{\alpha}{\log \alpha}\right)$. Bansal, Chan and Pruhs [18] later showed that the same algorithm can in fact achieve 3-competitive, i.e., independent of the power parameter α . Finally, Andrew, Wierman and Tang [10] improved the ratio to 2, which matches the lower bound given by Andrew, Lin and Wierman [9], hence no further improvement is possible. So far, all these results assume clairvoyance from the scheduling algorithms. For the non-clairvoyant setting, Lam et al. [117] showed that RR with AJC is $(2 - \frac{1}{\alpha})$ -competitive for batch released jobs. Chan et al. [48] showed that LAPS_β with AJC is $O(\alpha^3)$ -competitive for arbitrarily released jobs.

The above results are all about scheduling sequential jobs on a single processor.

We now review related work on multiprocessor scheduling. For scheduling sequential jobs on multiprocessors, Lam et al. [116] showed that by combining CRR (Classified Round Robin) for dispatching jobs to processors with the clairvoyant algorithms on a single processor, such as AJC plus SRPT shown in [21, 117], one can obtain a non-migratory scheduler that is $O(\log \Delta)$ -competitive with respect to the total response time plus energy, where Δ denotes the ratio of the maximum job size to the minimum job size. Lam et al. [115, 118] later improved this ratio to $O(1)$ -competitive through an enhanced analysis of the same algorithm. Recently, Greiner, Nonner and Souza [83] also presented an $O(1)$ -competitive algorithm with RD (Random Dispatching) of jobs, with a smaller competitive ratio than the ones in [115, 118]. Moreover, their analysis can be applied to the non-clairvoyant setting for the algorithm LAPS_β with AJC shown in [48] to obtain similar $O(1)$ -competitive result.

For scheduling parallel jobs on multiprocessors, both Chan, Edmonds and Pruhs [49] and Sun, Cao and Hsu [174] observed that any non-clairvoyant algorithm that allocates one set of uniform speed processors to a job performs poorly, or specifically $\Omega(P^{(\alpha-1)/\alpha^2})$ -competitive, where P is the total number of processors. Hence, to obtain better results, Chan, Edmonds and Pruhs [49] proposed the MULTILAPS_β algorithm, which generalized the LAPS_β algorithm given in [48] by allocating multiple groups of processors with different speeds to a job. They showed that for nonbatched jobs, MULTILAPS_β is $O(\log P)$ -competitive with respect to the total response time plus energy, and any non-clairvoyant algorithm is $\Omega(\log^{1/\alpha} P)$ -competitive. Sun, Cao and Hsu [174] assumed a different execution model and proposed the N-EQUI (Non-uniform Equi-partitioning) algorithm by allocating one group of non-uniform speed processors to a job. They showed that N-EQUI is $O(\ln^{1/\alpha} P)$ -competitive for batched jobs. Sun, He and Hsu [176] later showed that for nonbatched jobs N-EQUI achieves $O(\ln P)$ -competitive, and they also gave a lower bound of $\Omega(\ln^{1/\alpha} P)$ on the competitive ratio of any non-clairvoyant algorithm. Both results match asymptotically those given in [49]. This thesis shows that the two-level adaptive schedulers N-ACCEQ and N-AGCEQ achieve the same asymptotic bounds as N-EQUI. Finally, Sun, He and Hsu [176] proposed the semi-clairvoyant algorithm U-CEQ (Uniform Conservative Equi-partitioning) and showed that it is $O(1)$ -competitive with respect to the total response time plus energy even for nonbatched jobs.

Note that all of the results mentioned so far are based on the simple power function proposed by Yao, Demers and Shenker [189] in 1995, where a processor consumes power s^α when running at speed s and $\alpha > 1$. Besides this simple model, various results [17, 117, 115, 116, 118, 18, 114, 83, 174] have also been obtained based on more practical assumptions on the power function, such as imposing a maximum speed on the processors, or considering discrete speed processors, or including sleep modes, etc. Surprisingly, most of these results achieve (asymptotically) identical ratios compared with the corresponding results based on the simple power function, or with slight

Table 2.9: Some results that minimize total response time or makespan plus energy.

<i>Reference</i>	<i>Algorithm</i>	<i>Comp. ratio</i>	<i>Objective</i>	<i>Proc#</i>	<i>Job type</i>	<i>Setting</i>
Albers et al. [7]	AJC + SJF variant	$8.3e \left(\frac{3+\sqrt{5}}{2} \right)^\alpha$	response + energy	1	unit seq.	clairv.
Bansal et al. [21]	AJC + SJF	4	response+energy	1	unit seq.	clairv.
Bansal et al. [21]	AJC + SJF variant	$O \left(\frac{\alpha^2}{\log^2 \alpha} \right)$	response + energy	1	seq.	clairv.
Lam et al. [117]	AJC + SRPT	$O \left(\frac{\alpha}{\log \alpha} \right)$	response + energy	1	seq.	clairv.
Bansal et al. [18]	AJC + SRPT	3	response+energy	1	seq.	clairv.
Andrew et al. [10]	AJC + SRPT	2	response+energy	1	seq.	clairv.
Andrew et al. [9]	online	2 (lower bound)	response+energy	1	seq.	clairv.
Lam et al. [117]	AJC + RR	$(2 - \frac{1}{\alpha})$	response+energy	1	batch seq.	non-clairv.
Chan et al. [48]	AJC + LAPS _{β}	$O(\alpha^3)$	response+energy	1	seq.	non-clairv.
Lam et al. [116]	CRR + AJC + SRPT	$O(\log \Delta)$	response+energy	P	seq.	clairv.
Lam et al. [118]	CRR + AJC + SRPT	$O(1)$	response+energy	P	seq.	clairv.
Greiner et al. [83]	RD + AJC + LAPS _{β}	$O(1)$	response + energy	P	seq.	non-clairv.
Chan et al. [49]	MULTILAPS _{β}	$O(\log P)$	response+energy	P	arbitrary	non-clairv.
Sun et al. [174]	N-EQUI	$O(\ln^{1/\alpha} P)$	response+energy	P	batch	non-clairv.
*Sun [This thesis]	N-ACCEQ N-AGCEQ	$O(\ln^{1/\alpha} P)$	response + energy	P	batch	non-clairv.
Sun et al. [176]	N-EQUI	$O(\ln P)$	response+energy	P	arbitrary	non-clairv.
*Sun [This thesis]	N-ACCEQ N-AGCEQ	$O(\ln P)$	response + energy	P	arbitrary	non-clairv.
Chan et al. [49] *Sun et al. [176]	online	$\Omega(\ln^{1/\alpha} P)$	response+energy	P	arbitrary	non-clairv.
*Sun et al. [176]	U-CEQ	$O(1)$	response+energy	P	arbitrary	semi-clairv.
Sun et al. [176]	P-FIRST	$O(\ln^{1-1/\alpha} P)$	makespan+energy	P	batch (PAR-SEQ)	semi-clairv.
*Sun et al. [176]	online	$\Omega(\ln^{1-1/\alpha} P)$	makespan+energy	P	seq.	semi-clairv.

relaxation on the processor speeds.

We now turn to energy-efficient scheduling for makespan. Compared with the objectives that involve total response time, there are relatively fewer results concerning makespan. Pruhs, van Stee and Uthaisombut [157] considered scheduling batched sequential jobs with precedence constraints on multiprocessors to minimize their makespan subject to a given energy budget. They observed that for this problem the total power consumption from all processors is constant over time in the optimal schedule, and proposed an $O(\log^{1+2/\alpha} P)$ -approximation algorithm by reducing the problem to scheduling on uniformly related machines. Bunde [43] studied the same problem on both a single processor and multiprocessors for sequential jobs without precedence constraints but with arbitrary release time. He proposed a linear-time algorithm that finds the optimal schedule on a single processor. On multiprocessors, an optimal solution up to arbitrary precision is given for unit work jobs. However, for jobs with arbitrary work, NP-hardness result was proven even if they are batch released. Pruhs, van Stee and Uthaisombut [157] proposed a PTAS for the special case where all jobs are batch released. Różycki and Weglarz [164] also considered this problem by limiting the total power consumption at any time as well and proposed an optimal solution for another special case, where the number of jobs is not more than the number of processors. Sun, He and Hsu [176] considered combining makespan and

energy consumption into a single objective, and they proposed the semi-clairvoyant algorithm P-FIRST (Parallel-First) for any batched set of parallel jobs that consist of only fully-parallel and sequential phases, or (PAR-SEQ)* jobs. They showed that P-FIRST achieves $O(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy in this setting as well as a matching lower bound of $\Omega(\ln^{1-1/\alpha} P)$ on the competitive ratio of any semi-clairvoyant algorithm, using only sequential jobs.

Table 2.9 summarizes the online results that optimize a combination of total response time or makespan and energy consumption surveyed in this section.

Chapter 3

Two-Level Adaptive Scheduling: From A-GREEDY to ACDEQ

From this chapter onwards, we will present our two-level adaptive schedulers and analyze performances from various perspectives. In this chapter, we first motivate the presentation of our adaptive scheduler by describing an existing task scheduler A-GREEDY proposed by Agrawal et al. [3] under the same two-level scheduling framework. We show that despite its good theoretical performances, simple analysis of A-GREEDY reveals certain problems in the transient response of its processor desires. In particular, A-GREEDY suffers from desire instability and hence possible instability in processor allocation, even when the job's parallelism stays constant. Such instability problem can cause potential difficulties in the management of the processor resources, such as unnecessary processor waste and job execution delays as well as excessive reallocation overheads and loss of localities, etc., which tend to become worse with increased job parallelism. We then present a provably stable task scheduler A-CONTROL and combine it with the OS allocator DEQ (Dynamic Equi-partitioning) [139] to form a two-level scheduler ACDEQ. Unlike A-GREEDY, which responds to the job's parallelism variations in a discrete manner, A-CONTROL calculates processor desires continuously based on principles from *control theory*, which have been previously applied to scheduling real-time systems [130, 149] and designing computing applications [93]. In particular, we show that the processor desires calculated by A-CONTROL is able to achieve much improved transient and steady-state performances that A-GREEDY fails to attain. Furthermore, we also derive some algorithmic properties of ACDEQ and AGDEQ (which combines A-GREEDY with DEQ) when scheduling an individual job. Those properties form the foundation for the subsequent analysis of the two algorithms with respect to the total response time and makespan in Chapter 4 as well as their variants with respect to the set response time in Chapter 5. In these three chapters, we assume that all processors have fixed speed s , where $s > 0$. The content of this chapter was originally presented in [178, 175].

3.1 Revisit Adaptive Task Scheduler: A-GREEDY

We first review an existing adaptive task scheduler A-GREEDY to schedule individual malleable jobs. In particular, the processor desires of A-GREEDY are calculated using a multiplicative-increase multiplicative-decrease strategy based on the job's execution characteristics in the previous scheduling quantum.

Let t_q denote the time when a scheduling quantum q starts. The work $w_i(q)$ completed for job J_i in quantum q , or the *quantum work*, is given by $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q))dt$, where $\Gamma_i^{k_t}$ is the speedup function for the phase job J_i is executing at time t , $a_i(q)$ is the processor allocation of the job in quantum q and L is the quantum length. Job J_i is said to be *efficient* in quantum q if work $w_i(q)$ completed is at least δ fraction of the maximum amount of work that can be done in the quantum, i.e., $w_i(q) \geq \delta \cdot a_i(q)sL$, where $0 < \delta < 1$ is called the *utilization parameter* and $s > 0$ is the speed of the processors; otherwise it is *inefficient* if $w_i(q) < \delta \cdot a_i(q)sL$. Furthermore, the job is said to be *satisfied* in quantum q if its processor allocation $a_i(q)$ is at least the processor desire $d_i(q)$, i.e., $a_i(q) \geq d_i(q)$; otherwise, it is *deprived* if $a_i(q) < d_i(q)$. Based on the efficient-inefficient classification and the satisfied-deprived classification, the processor desire $d_i(q+1)$ of job J_i in the next quantum $q+1$ is calculated as shown in Algorithm 1, where $\rho > 1$ is the *responsiveness parameter*.

Algorithm 1 A-GREEDY

Require: quantum work $w_i(q)$, processor desire $d_i(q)$ and processor allocation $a_i(q)$ for job J_i in quantum q

Ensure: processor desire $d_i(q+1)$ for job J_i in quantum $q+1$

- 1: **if** $w_i(q) \geq \delta a_i(q)sL$ and $a_i(q) \geq d_i(q)$ **then**
 - 2: $d_i(q+1) = d_i(q) \cdot \rho$ //efficient and satisfied
 - 3: **else if** $w_i(q) < \delta a_i(q)sL$ **then**
 - 4: $d_i(q+1) = d_i(q)/\rho$ //inefficient
 - 5: **else**
 - 6: $d_i(q+1) = d_i(q)$ //efficient and deprived
-

In [3], a job is also said to be *accounted* if it is both satisfied and efficient in a quantum; otherwise, the job is *deductible* if it is either deprived or inefficient. For the first quantum when job J_i arrives at the system, its processor desire is simply set to 1. We can see from Algorithm 1 that the rationale of A-GREEDY is that it attempts to exploit the correlation between the parallelism of a job in two adjacent quanta, although the existence of such correlation is not explicitly assumed. Specifically, if the allocated processors in quantum q are not utilized efficiently, the parallelism of the job may not be as high. Therefore, the processor desire will be reduced by a factor of ρ for the next quantum $q+1$. If the allocated processors are utilized efficiently and the processor desire is satisfied, the parallelism of the job could be even higher. Thus, the processor desire will be increased by a factor of ρ to explore any potential parallelism. Lastly, if the allocated processors are utilized efficiently but the desire is deprived, it

is not known whether the processors could still be efficiently utilized had the desire been satisfied. Therefore, the processor desire is not changed for the next quantum.

It is not difficult to see, however, that if the parallelism of the job is indeed correlated, e.g., if the parallelism stays constant at a certain level for sufficiently long time, then the processor desires of A-GREEDY could become unstable as it oscillates around the target parallelism. The instability may cause much degradation in the actual performance of A-GREEDY due to the unnecessary processor waste and job execution delays. Furthermore, it also incurs extra context switching overheads and loss of locality, etc. In the next section, we present an improved task scheduling algorithm that will mitigate the problem.

3.2 Stable Two-Level Adaptive Scheduler: ACDEQ

We now propose our two-level adaptive scheduler ACDEQ, which combines a task scheduler A-CONTROL with the OS allocator DEQ [139]. In particular, A-CONTROL calculates the processor desires for a job using an adaptive controller that maintains the job's desire stability among other control-theoretic properties. In this section, we describe ACDEQ in detail.

Figure 3.1 shows the feedback control structure of ACDEQ for an individual job J_i . In each quantum q , the output of A-CONTROL is the processor desire $d_i(q)$. The desire is sent to OS allocator DEQ, which gives job J_i a processor allotment $a_i(q)$ based on the desires of all jobs as well as its allocation policy. During quantum q , job J_i is executed with $a_i(q)$ processors while information about the work $w_i(q)$ completed and the span $l_i(q)$ reduced for the job is collected. Specifically, let t_q denote the time quantum q starts, then the *quantum work* $w_i(q)$ is given by $w_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q)) dt$, and the *quantum span* $l_i(q)$ is given by $l_i(q) = \int_{t_q}^{t_q+L} \Gamma_i^{k_t}(a_i(q)) / h_i^{k_t} dt$, where $\Gamma_i^{k_t}$ and $h_i^{k_t}$ denote the speedup function and the parallelism for the phase job J_i is executing at time t respectively, and L is the quantum length. The *quantum average parallelism* $A_i(q)$ is therefore given by the quantum work divided by the quantum span, i.e., $A_i(q) = w_i(q)/l_i(q)$, which is a commonly used approach to represent the average parallelism for a job or a segment of the job [30, 32, 33]. The output $y_i(q)$ is defined as $y_i(q) = d_i(q)/A_i(q)$, and it is compared with the reference $f_i(q)$ to produce an error term $e_i(q) = f_i(q) - y_i(q)$, which is used by the adaptive controller to calculate the processor desire $d_i(q+1)$ for quantum $q+1$. The controller applies the following integral control law [93]:

$$d_i(q+1) = d_i(q) + K_i(q+1)e_i(q), \quad (3.1)$$

where $K_i(q+1)$ denotes the *controller gain* for quantum $q+1$ and it determines how aggressively the controller responds to the job's parallelism. Note that the controller adjusts the processor desire based on the desire and the error of the previous quantum,

Figure 3.1: Feedback control structure of ACDEQ.

and it is *adaptive* because the gain $K_i(q+1)$ is reset for each quantum based on the measurement $A_i(q)$ and some performance specifications. In the next section, we will present these specifications and show how to set the controller gain from control-theoretic perspective. As with A-GREEDY, the processor desire of job J_i in its first quantum is set to 1.

Also following the terminologies from A-GREEDY, we define some notions. Firstly, a job J_i is said to be *satisfied* in quantum q if its processor allocation is at least its processor desire, i.e., $a_i(q) \geq d_i(q)$; otherwise, if $a_i(q) < d_i(q)$, job J_i is *deprived*. In addition, job J_i is said to be *over-allocated* in quantum q if its processor allocation is more than its average parallelism in the quantum, i.e., $a_i(q) > A_i(q)$; otherwise, the job is *under-allocated* if $a_i(q) \leq A_i(q)$. Finally, job J_i is said to be *accounted* in a quantum if it is both deprived and under-allocated; otherwise, the job is *deductible* if it is either satisfied or over-allocated. Furthermore, we extend the concepts of accounted, deductible, etc. from quantum to time. For instance, job J_i is said to be accounted at time t if the job is accounted in the quantum which contains t .

Now, we describe the DEQ (Dynamic Equi-Partitioning) OS allocator [139], which is well known for its efficiency and fairness to allocate processors to jobs [39, 63, 88]. DEQ is a variants of EQUI (Equi-partitioning) [180] that divides the total number of processors equally among all active jobs at any time. In DEQ, however, if a job desires fewer processors than the equal share, it will not be allocated more processors than its desire, and the remaining processors will instead be given to the other jobs with higher desires. Let $\mathcal{J}(t)$ denote the set of active jobs at time t when quantum q begins. Based on the processor desires from all jobs in $\mathcal{J}(t)$, DEQ allocates the processors as shown in Algorithm 2.

As we can see that if a job's processor desire is not more than the equal share $P/|\mathcal{J}(t)|$ of processors, the job will be satisfied (line 3 and line 10). After that, the equal share will be recalculated excluding the jobs already satisfied and the processors already allocated. The remaining processors will then be allocated to the rest of the jobs by recursively calling the main procedure (line 11) until either all jobs are satisfied or all jobs' processor desires are more than the equal share. In the latter case, each remaining job will be deprived and get the current equal share of processors (lines 4-7). As was shown in [63, 88], if there are deprived jobs for a quantum, then all P

Algorithm 2 DEQ

Require: set $\mathcal{J}(t)$ of active jobs when quantum q begins, processor desire $d_i(q)$ of each active job J_i in quantum q , and total number P of processors

Ensure: processor allocation $a_i(q)$ of each active job J_i in quantum q

```
1: if  $\mathcal{J}(t) = \emptyset$  then
2:   return
3:  $S = \{J_i \in \mathcal{J}(t) : d_i(q) \leq P/|\mathcal{J}(t)|\}$ 
4: if  $S = \emptyset$  then
5:   for each  $J_i \in \mathcal{J}(t)$  do
6:      $a_i(q) = P/|\mathcal{J}(t)|$  //deprived jobs get current equal share
7:   return
8: else
9:   for each  $J_i \in S$  do
10:     $a_i(q) = d_i(q)$  //satisfied jobs get their desires
11:  DEQ( $\mathcal{J}(t) - S, P - \sum_{J_i \in S} a_i(q)$ )
```

processors must have been allocated by DEQ, and each deprived job will have the same number of allocated processors, which is higher than the initial equal share $P/|\mathcal{J}(t)|$.

3.3 Control-Theoretic Analysis

The adaptive controller shown in Figure 3.1 dynamically adjusts its controller gain based on the time-varying parallelism of the job and is referred to as *self-tuning regulator* [13] in control theory. In this section, we determine how the controller gain can be set for each scheduling quantum via control-theoretic analysis. Basically, we transform the system into z -domain, and employ pole placement strategy [93] by considering a set of transient and steady-state performance specifications, which directly apply to the scheduler when the job's average parallelism is constant (e.g., the job does not experience phase transitions). When the job's average parallelism changes (e.g., by making a transition from one phase to the next), our analysis will apply anew to the scheduler with respect to the job's new average parallelism. In case that the job's parallelism changes continuously, these specifications are unfortunately no longer applicable. In Section 3.5.1, we will address this issue by empirically studying the transient response of A-CONTROL using the set of parallelism variations shown in Section 1.6.2.

Now, we focus on the scenario with relatively constant average parallelism for a job. Assume that job J_i 's average parallelism is A_i at some time, and it will stay constant at A_i for sufficiently long. The controller gain K_i , which depends on the value of A_i , will also stay constant in the mean time. Ideally, the processor desire should match the job's average parallelism for a quantum to be efficient. Hence, the reference $f_i(q)$ of our scheduler is set to 1 in all quanta, which corresponds to a unit-step function. Thus, we can represent the REFERENCE, A-CONTROL and EXECUTION shown in

Figure 3.1 in z -domain as follows:

- REFERENCE: $F_i(z) = z/(z - 1)$.
- A-CONTROL: $G_i(z) = D_i(z)/E_i(z) = K_i/(z - 1)$.
- EXECUTION: $S_i(z) = Y_i(z)/D_i(z) = 1/A_i$.

The closed-loop transfer function of the system can be derived accordingly as

$$T_i(z) = \frac{Y_i(z)}{F_i(z)} = \frac{G_i(z)S_i(z)}{1 + G_i(z)S_i(z)} = \frac{K_i/A_i}{z - (1 - K_i/A_i)}. \quad (3.2)$$

Clearly, the closed-loop is a first-order system with single pole $p = 1 - K_i/A_i$. We adopt the following set of criteria [93] commonly used in control theory to place the pole by setting the value of controller gain K_i .

- *BIBO-Stability*. The system is said to be *bounded-input bounded-output (BIBO) stable* if given a bounded reference, the processor desire is also bounded.
- *Steady-State Error*. The steady-state error is the difference between the processor desire and the job's average parallelism after sufficiently long time, i.e., at steady state.
- *Maximum Overshoot*. The maximum overshoot is the maximal difference between the transient processor desire and its steady-state value.
- *Convergence Rate*. The convergence rate v is the speed at which the processor desire approaches the job's average parallelism, and is defined to be $v = \max_q (|d_i(q + 1) - A_i|/|d_i(q) - A_i|)$.

Note that these four criteria directly apply to the output $y_i(q)$ of the closed-loop system. However, since $y_i(q)$ is defined to be $y_i(q) = d_i(q)/A_i(q)$ and the average parallelism $A_i(q)$ of the job is assumed to be constant for the period of consideration, the criteria also specify the transient and steady-state performances of the processor desire. We show that A-CONTROL has good performance in terms of these criteria in the following theorem.

Theorem 3.1 *Suppose that A-CONTROL schedules a job J_i whose average parallelism stays constant at A_i for sufficiently long time. If the controller gain is set to $K_i = (1 - v)A_i$, where $v \in [0, 1)$, then the processor desires satisfy (1) BIBO-stability, (2) zero steady-state error, (3) zero overshoot, and (4) convergence rate v .*

Proof. Using rules established in control theory [93], we prove the theorem based on the pole position in z -plane, which is $p = 1 - K_i/A_i = 1 - (1 - v) = v$. For a first-order system to be BIBO-stable, the pole needs to be within the unit circle, which

is satisfied since $|p| = |v| < 1$. Applying Final Value Theorem [93] on output $y_i(p)$, we get $\lim_{q \rightarrow \infty} y_i(q) = \lim_{z \rightarrow 1} (z - 1)Y_i(z) = 1$. Hence, it has zero steady-state error. For a first-order system to have zero overshoot, the pole needs to be nonnegative. Finally, the convergence rate of a first-order system is exactly given by the value of its pole, which is independent of quantum q . $\square$

Since the average parallelism $A_i(q)$ is computed for each quantum $q \geq q_i$ based on the measured quantum work $w_i(q)$ and quantum span $l_i(q)$, where q_i is the first quantum when job J_i enters the system, the controller gain based on Theorem 3.1 is set to $K_i(q + 1) = (1 - v)A_i(q)$ for $q \geq q_i$. Substituting it into Equation (3.1) and simplifying, we essentially get the strategy of A-CONTROL to calculate the processor desire for each quantum $q > q_i$, as shown in Algorithm 3.

Algorithm 3 A-CONTROL

Require: quantum work $w_i(q)$, quantum span $l_i(q)$ and processor desire $d_i(q)$ for job J_i in quantum q

Ensure: processor desire $d_i(q + 1)$ for job J_i in quantum $q + 1$

- 1: $A_i(q) = w_i(q)/l_i(q)$
 - 2: $d_i(q + 1) = vd_i(q) + (1 - v)A_i(q)$
-

Note that v is a design parameter of A-CONTROL and can be configured differently for each job, but in this thesis we assume that it is set uniformly for all jobs. Of course, a special case is when the convergence rate is set to $v = 0$. This gives the fastest rate of convergence, or one-quantum convergence. The resulting processor desire for quantum $q + 1$ is then equal to the job's average parallelism in the previous quantum, i.e., $d_i(q + 1) = A_i(q)$.

Figure 3.2 compares the behaviors of A-CONTROL and A-GREEDY over several phases of a synthetic job. The quantum length in this case is set to 1, and is scaled in the figure to restore the original parallelism of the job. The convergence rate v of A-CONTROL is set to 0.2. The responsiveness parameter ρ of A-GREEDY is set to 2, and its utilization parameter δ is set to 0.8. As we can see, A-CONTROL satisfies good transient and steady-state performances in each phase of the job while A-GREEDY suffers from apparent desire oscillation, nonzero steady-state error and overshoot. Other choices of the parameter values will result in similar behaviors of the two algorithms. For example, varying the value of v will only change the convergence rate of A-CONTROL without affecting its other control-theoretic properties, such as stability. For A-GREEDY, different combinations of ρ and δ , such as reducing ρ and increasing δ will alleviate its desire oscillation but at the cost of slower convergence. In either case, the desire instability of A-GREEDY cannot be completely eliminated, and from practical point of view, this problem may cause potential oscillation in its processor allocations, especially when the load of the system is small and hence the desires tend to be satisfied. Such unnecessary processor reallocations will introduce extra overheads due to the context switching of some processors from one job to

(a)

(b)

Figure 3.2: Transient and steady-state behaviors of (a) A-CONTROL and (b) A-GREEDY over several phases of a synthetic job.

another as well as other associated issues, such as loss of localities, etc. In fact, the processor desire of A-GREEDY does satisfy BIBO-stability; its oscillation (referred to as *limit cycle* [93] in control theory) is a direct result of the non-linear behavior of the A-GREEDY algorithm.

3.4 Algorithmic Preliminaries

We now derive some properties of the ACDEQ and AGDEQ algorithms when scheduling an individual job, which lay the algorithmic foundations for our analysis in the next two chapters. In particular, we bound the response time and processor waste of any single job in a job set scheduled by ACDEQ and AGDEQ. We then compare ACDEQ with AGDEQ in terms of these bounds.

3.4.1 Properties of ACDEQ

For any job J_i scheduled by ACDEQ, given that its work completed in a quantum q is $w_i(q)$ and its span reduced in the quantum is $l_i(q)$, we define the *quantum work efficiency* to be $\alpha_i(q) = \frac{w_i(q)}{a_i(q)sL}$, and the *quantum span efficiency* to be $\beta_i(q) = \frac{l_i(q)}{sL}$, where $a_i(q)$ is the processor allocation to the job in quantum q , s is the processor speed and L is the quantum length. Obviously, we have $0 \leq \alpha_i(q), \beta_i(q) \leq 1$. The quantum average parallelism $A_i(q)$ is therefore

$$A_i(q) = \frac{w_i(q)}{l_i(q)} = a_i(q) \frac{\alpha_i(q)}{\beta_i(q)}. \quad (3.3)$$

Let t_q denote the time quantum q starts, then quantum q can be split into two portions depending on whether the processor allocation $a_i(q)$ is more than the parallelism $h_i^{k_t}$ of the job at any time $t \in [t_q, t_q + L]$. Specifically, let $L_1 = \int_{t_q}^{t_q+L} [a_i(q) \leq h_i^{k_t}]$ and $L_2 = \int_{t_q}^{t_q+L} [a_i(q) > h_i^{k_t}]$, where $[x]$ is 1 if proposition x is true and 0 otherwise. Hence, the quantum work $w_i(q)$ is at least $a_i(q)sL_1$ and the quantum span $l_i(q)$ is at least sL_2 . Thus, we have $L = L_1 + L_2 \leq \frac{w_i(q)}{a_i(q)s} + \frac{l_i(q)}{s} = \left(\frac{\alpha_i(q)}{\beta_i(q)} + 1 \right) \frac{l_i(q)}{s}$. Substituting in $l_i(q) = \beta_i(q)sL$, we get

$$\alpha_i(q) + \beta_i(q) \geq 1, \quad (3.4)$$

which is a lower bound on the sum of quantum work efficiency and quantum span efficiency. We will use this relationship as well as Equation (3.3) later in our analysis to bound response time and processor waste of a job.

We now define the concept of *transition factor* for job J_i . The transition factor, denoted by C_i , where $C_i \geq 1$, is the maximal ratio on the average parallelism of the job in any two consecutive quanta. Formally, the average quantum parallelism of the job satisfies

$$\frac{1}{C_i} \leq \frac{A_i(q)}{A_i(q-1)} \leq C_i, \quad (3.5)$$

for all $q \geq q_i$, and $A_i(q_i - 1)$ is defined to be 1, where q_i is the first quantum job J_i is scheduled. The transition factor, like the work and span, can be considered as an intrinsic job characteristic, and is independent of the task scheduler.¹ Since two malleable jobs with the same work and span can have very different parallelism variations, the transition factor indicates how fast the job's parallelism changes with time and thus suggests the level of difficulty to adaptively schedule it in a non-clairvoyant fashion. In contrast to [3], which does not consider the parallelism transitions of a job, we argue that the incorporation of transition factor better reflects the performance of

¹The transition factor, however, does depend on the quantum length and the processor speed, whose variation may yield different transition factor for the same job. For a given quantum length, processor speed, as well as the parallelism profile of a job, the transition factor can usually be derived based on the worst-case schedule. We will not be concerned about how the transition factor may be derived, much like the work and span of a job. We will just make use of these job characteristics to quantify the behavior of our scheduler in terms of performance bounds.

a scheduling algorithm.

In particular, given the transition factor of a job, we can show that the processor desire generated by A-CONTROL in any quantum is bounded from both above and below in terms of the average parallelism of the job in the same quantum.

Lemma 3.2 *Suppose that A-CONTROL schedules a job J_i with transition factor C_i . Then the processor desire $d_i(q)$ for each quantum q satisfies*

$$\frac{1-v}{C_i-v}A_i(q) \leq d_i(q) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q), \quad (3.6)$$

where $A_i(q)$ is the job's average parallelism in quantum q and v denotes the convergence rate of A-CONTROL. The inequality on the right only holds when $v < 1/C_i$.

Proof. We will prove the upper bound of $d_i(q)$ by induction on the scheduling quantum. The lower bound can be proven similarly, and is omitted.

Base case: For $q = q_i$, we have $d(q_i) = 1$. Because $A_i(q_i - 1) = 1$ by definition, according to Inequality (3.5), the quantum average parallelism satisfies $A_i(q_i) \geq 1/C_i$. Thus we have $\frac{d_i(q_i)}{A_i(q_i)} \leq C_i \leq \frac{C_i - C_iv}{1 - C_iv}$, since $C_i \geq 1$ and $C_iv < 1$. Therefore, we get $d_i(q_i) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q_i)$.

Induction: For $q \geq q_i + 1$, suppose that we have $d_i(q-1) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q-1)$. Because $A_i(q-1) \leq C_iA_i(q)$ from Inequality (3.5), then according to the A-CONTROL algorithm, we have for quantum q , $d_i(q) = vd_i(q-1) + (1-v)A_i(q-1) \leq \frac{C_iv(1-v)}{1-C_iv}A_i(q-1) + (1-v)A_i(q-1) = \frac{1-v}{1-C_iv}A_i(q-1) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q)$. $\square$

From the proof of Lemma 3.2, we can see that the assumption $v < 1/C_i$ is required for the upper bound of the processor desire to hold. Without this assumption, however, the ratio of processor desire and quantum average parallelism cannot be bounded. The worst case happens when the parallelism of the job reduces much faster than the responsiveness of A-CONTROL with the chosen convergence rate, resulting in the processor desire not reduced as quickly.

In the following, we start to analyze the response time and the processor waste of an individual job scheduled by ACDEQ. Agrawal et al. [3] studied the response time of a job scheduled by task scheduler A-GREEDY using *trim analysis*, which is a technique to limit the power of the OS allocator assuming that it can behavior like an adversary to the task scheduler. However, most practical OS allocators such as DEQ do work cooperatively with the task schedulers. Hence, in the following theorem, we show that the response time of any individual job scheduled by ACDEQ can be bounded in terms of the job's deserved equal share of processors and their speed.

Theorem 3.3 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on P processors of speed s , where $s > 0$. Then the response time $R_{AC}(J_i)$ of any individual job $J_i \in \mathcal{J}$ with*

work $w(J_i)$, span $l(J_i)$ and transition factor C_i is bounded by

$$R_{AC}(J_i) \leq \frac{2w(J_i)}{s\bar{P}} + \frac{C_i + 1 - 2v}{s(1-v)}l(J_i), \quad (3.7)$$

where $\bar{P} = P/|\mathcal{J}|$ is the equal share of processors for each job and v is the convergence rate of A-CONTROL.

Proof. To bound the response time $R_{AC}(J_i)$ of job J_i , we will bound separately its total accounted time and total deductible time, denoted by $R'_{AC}(J_i)$ and $R''_{AC}(J_i)$, respectively. Apparently, we have $R_{AC}(J_i) = R'_{AC}(J_i) + R''_{AC}(J_i)$.

The total accounted time can be bounded as follows. Since an accounted quantum q for job J_i is under-allocated by definition, we have $a_i(q) \leq A_i(q)$. According to Equation (3.3) and Inequality (3.4), we have $\alpha_i(q) \geq 1/2$ and therefore $w_i(q) \geq a_i(q)sL/2$. Since an accounted quantum is also deprived, based on DEQ policy, we have $a_i(q) \geq P/|\mathcal{J}(t)| \geq \bar{P}$. Let A denote the set of accounted quanta of job J_i . The total work done on accounted time thus satisfies $w(J_i) \geq \sum_{q \in A} w_i(q) \geq \sum_{q \in A} a_i(q)sL/2 \geq \bar{P}sR'_{AC}(J_i)/2$. The total accounted time is then $R'_{AC}(J_i) \leq \frac{2w(J_i)}{s\bar{P}}$.

To bound the total deductible time, we observe from definition of deductible quantum and Lemma 3.2 that $a_i(q) \geq \frac{1-v}{C_i-v}A_i(q)$ for job J_i in deductible quantum q . Substituting Equation (3.3) and Inequality (3.4) into it, we obtain $\beta_i(q) \geq \frac{1-v}{C_i+1-2v}$ and therefore $l_i(q) \geq \frac{1-v}{C_i+1-2v}sL$. Let D denote the set of deductible quanta of job J_i . The sum of the quantum span in deductible quanta thus satisfies $l(J_i) \geq \sum_{q \in D} l_i(q) \geq \sum_{q \in D} \frac{1-v}{C_i+1-2v}sL = \frac{1-v}{C_i+1-2v}sR''_{AC}(J_i)$. The total deductible time is $R''_{AC}(J_i) \leq \frac{C_i+1-2v}{s(1-v)}l(J_i)$. $\square$

We now bound the processor waste of the ACDEQ algorithm, which relies on the convergence rate of A-CONTROL to satisfy $v < 1/C_i$ for a job J_i with transition factor C_i . Since the characteristics of a job are usually unknown prior to its execution, we assume that the convergence rate is chosen based on some historical workload characterization, which ensures that it satisfies the requirement. The processor waste can then be bounded as stated in the following theorem.

Theorem 3.4 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on processors of speed s , where $s > 0$. Then the processor waste $X_{AC}(J_i)$ for any individual job $J_i \in \mathcal{J}$ with work $w(J_i)$ and transition factor C_i is bounded by*

$$X_{AC}(J_i) \leq \frac{C_i(1-v)}{1-C_iv}w(J_i), \quad (3.8)$$

where $v < 1/C_i$ is the convergence rate of A-CONTROL.

Proof. Since ACDEQ never allocates more processors than job J_i desires, that is, $a_i(q) \leq d_i(q)$ for any quantum q , from Lemma 3.2, we have $a_i(q) \leq \frac{C_i(1-v)}{1-C_iv}A_i(q)$. Substituting Equation (3.3) and Inequality (3.4) into it, we obtain $\alpha_i(q) \geq \frac{1-C_iv}{1+C_i-2C_iv}$, and

therefore $w_i(q) \geq \frac{1-C_iv}{1+C_i-2C_iv}a_i(q)sL$. Let $X_i(q)$ denote the processor waste of job J_i in quantum q , then we have $X_i(q) = a_i(q)sL - w_i(q) \leq a_i(q)sL - \frac{1-C_iv}{1+C_i-2C_iv}a_i(q)sL = \frac{C_i(1-v)}{1+C_i-2C_iv}a_i(q)sL \leq \frac{C_i(1-v)}{1-C_iv}w_i(q)$. The total processor waste is then bounded by $X_{AC}(J_i) = \sum_q X_i(q) \leq \sum_q \frac{C_i(1-v)}{1-C_iv}w_i(q) \leq \frac{C_i(1-v)}{1-C_iv}w(J_i)$. $\square$

Theorem 3.4 suggests that the processor waste for an individual job scheduled by ACDEQ is independent of the speed s of the processors. Let $X'_{AC}(J_i)$ denote the processor waste of job J_i on the set of accounted quanta. Then we can obtain a much improved bound for $X'_{AC}(J_i)$. Since an accounted quantum q for job J_i is under-allocated by definition, we have $a_i(q) \leq A_i(q)$. Following the proof of Theorem 3.4, we can get $X'_{AC}(J_i) \leq w(J_i)$.

3.4.2 Properties of AGDEQ

Agrawal et al. [3] derived the response time and the processor waste of any individual job scheduled by the AGDEQ algorithm on unit-speed processors. For completeness, we adopt their analysis and show in this section the bounds of AGDEQ on processors of arbitrary speed. Moreover, the response time bound is obtained without using trim analysis, as with that of ACDEQ.

Theorem 3.5 *Suppose that AGDEQ schedules a set $\mathcal{J}$ of jobs on P processors of speed s , where $s > 0$. Then the response time $R_{AG}(J_i)$ of any individual job $J_i \in \mathcal{J}$ with work $w(J_i)$ and span $l(J_i)$ is bounded by*

$$R_{AG}(J_i) \leq \frac{w(J_i)}{\delta s \bar{P}} + \frac{2}{s(1-\delta)}l(J_i) + L \log_\rho P + L, \quad (3.9)$$

where $\bar{P} = P/|\mathcal{J}|$ is the equal share of processors for each job, L is the quantum length, and δ and ρ are the utilization parameter and responsiveness parameter of A-GREEDY, respectively.

Proof. Let A and D denote the set of accounted quanta and the set of deductible quanta of job J_i , respectively. As with the proof for ACDEQ, we will bound the total accounted time $R'_{AG}(J_i)$ and total deductible time $R''_{AG}(J_i)$ of the job, separately.

Since an accounted quantum q for job J_i is efficient by definition, we have $w_i(q) \geq \delta a_i(q)sL$. Since an accounted quantum is also deprived, based on DEQ policy, we have $a_i(q) \geq P/|\mathcal{J}(t)| \geq \bar{P}$. The total work done on accounted time thus satisfies $w(J_i) \geq \sum_{q \in A} w_i(q) \geq \sum_{q \in A} \delta a_i(q)sL \geq \bar{P} \delta s R'_{AG}(J_i)$. The total accounted time is then $R'_{AG}(J_i) \leq \frac{w(J_i)}{\delta s \bar{P}}$.

Let I and ES denote the set of inefficient quanta and the set of efficient and satisfied quanta, respectively. By definition, we have $|D| = |ES| + |I|$. To bound the total deductible time, we first bound the number $|I|$ of inefficient quanta. In any inefficient quantum q , we have $w_i(q) < \delta a_i(q)sL$. Hence, the amount of time in the quantum when all $a_i(q)$ allocated processors are doing useful work is no more than $\frac{w_i(q)}{a_i(q)s} < \delta L$.

The amount of time when less than $a_i(q)$ processors are doing work is then at least $(1-\delta)L$, which contributes to reducing the span of the job. Therefore, the span $l_i(q)$ of job J_i reduced in quantum q satisfies $l_i(q) \geq s(1-\delta)L$. Given the total span $l(J_i)$ of the job, the number of inefficient quanta is thus bounded by $|I| \leq \frac{l(J_i)}{s(1-\delta)L}$. According to A-GREEDY, the processor desire starts at 1, increases by a factor of ρ after each efficient and satisfied quantum and decreases by a factor of ρ after each inefficient quantum, and the desire in any quantum cannot exceed $\rho \cdot P$ [3]. Thus, we have $\rho^{|ES|-|I|} \leq \rho \cdot P$, from which we can get $|ES| \leq |I| + \log_\rho P + 1$. Therefore, the total deductible time is given by $R''_{AG}(J_i) = |D|L = (|ES| + |I|)L \leq \frac{2}{s(1-\delta)}l(J_i) + L \log_\rho P + L$. $\square$

Theorem 3.6 *Suppose that AGDEQ schedules a set $\mathcal{J}$ of jobs on processors of speed s , where $s > 0$. Then the processor waste $X_{AG}(J_i)$ for any individual job $J_i \in \mathcal{J}$ with work $w(J_i)$ is bounded by*

$$X_{AG}(J_i) \leq \frac{1 + \rho - \delta}{\delta} w(J_i), \quad (3.10)$$

where δ and ρ are the utilization parameter and responsiveness parameter of A-GREEDY, respectively.

Proof. Let $X_i(q)$ denote the processor waste of job J_i in quantum q . In each efficient quantum q , we have $w_i(q) \geq \delta a_i(q)sL$. Hence, the processor waste in efficient quantum q is given by $X_i(q) = a_i(q)sL - w_i(q) \leq (1-\delta)a_i(q)sL \leq \frac{1-\delta}{\delta}w_i(q)$. According to the multiplicative-increase and multiplicative-decrease strategy of A-GREEDY, for each efficient and satisfied quantum q , there is a inefficient quantum q' , in which $d_i(q') = \rho d_i(q)$ [3, 5]. Hence, the processor waste in inefficient quantum q' is given by $X_i(q') \leq a_i(q')sL \leq d_i(q')sL = \rho d_i(q)sL = \rho a_i(q)sL \leq \frac{\rho}{\delta}w_i(q)$. Let E and I denote the set of efficient quanta and the set of inefficient quanta, respectively. The total processor waste is then bounded by $X_{AG}(J_i) = \sum_{q \in E} X_i(q) + \sum_{q' \in I} X_i(q') \leq \frac{1+\rho-\delta}{\delta} \sum_{q \in E} w_i(q) \leq \frac{1+\rho-\delta}{\delta} w(J_i)$. $\square$

From the proof of Theorem 3.6, we can see that if only the processor waste $X'_{AG}(J_i)$ in efficient quanta is of concern, we can get $X'_{AG}(J_i) \leq \frac{1-\delta}{\delta} w(J_i)$, which improves upon the total processor waste of AGDEQ.

Now, comparing ACDEQ and AGDEQ in terms of their performances for an individual job, we get that ACDEQ tends to have better bounds when the jobs have small transition factors; otherwise, the bounds of AGDEQ become better with appropriately chosen parameters. Note that AGDEQ is oblivious of the transition factor in the analysis because the symmetric structure of its multiplicative-increase multiplicative-decrease strategy allows it to bypass this difficulty.

It should be noted that although both algorithms attempt to exploit the parallelism correlation of the jobs, their theoretical performance bounds are established pessimistically based on the scenario where the future parallelism of a job is not correlated to its past. (In case of ACDEQ, a job's parallelism is only correlated by its

maximum transition.) Their performances in practice should therefore be much better than predicted by these theoretical bounds, especially when the parallelism of the jobs does exhibit strong correlation. In the next section, we will further evaluate and compare the performances of ACDEQ and AGDEQ through empirical studies.

3.5 Empirical Evaluations

We conduct simulations to study the empirical performance of ACDEQ and compare it with AGDEQ. To better understand these two-level adaptive schedulers, we primarily focus on the two task schedulers in this chapter by first studying their transient responses in terms of the processor desire calculations, which will directly shed light on the performances of the two-level schedulers in practice. We then evaluate the two task schedulers in terms of the individual job performances.

3.5.1 Transient Responses

In Section 3.3, we analyzed the control-theoretic performances of A-CONTROL when the parallelism of a job is constant. To better understand the behaviors of the two-level schedulers, we focus on both A-CONTROL and A-GREEDY in this section by studying their transient responses in terms of the generic parallelism variations given in Section 1.6.2, which will provide valuable insights on their performances in practice.

Figures 3.3 and 3.4 show the transient responses of A-CONTROL and A-GREEDY on four parallelism profiles. (The transient response of Impulse profile is similar to that of Step and hence is not shown.) In the figure, each profile has the same work, span, and average parallelism. The quantum length is set to $1/5$ of the segment length, which is scaled in the figure to restore the original parallelism variation. In addition, sequential phases are added before and after each profile such that the processor desires of both schedulers start and end at a steady state with value of 1, and are always satisfied by the OS allocator. The convergence rate of A-CONTROL is set to $v = 0$ in this case for faster response while the responsiveness parameter and the utilization parameter of A-GREEDY are set to $\rho = 2$ and $\delta = 0.8$, respectively. As can be seen, for Step profile, A-GREEDY is able to gradually catch up with the parallelism change but suffers from desire instability when the parallelism remains constant. In contrast, A-CONTROL rapidly approaches the parallelism within a quantum, and thereafter provides stable desires by directly utilizing the average parallelism of the job. For the other profiles, both A-GREEDY and A-CONTROL are able to respond gradually to the parallelism variations with A-CONTROL in general following more closely the changes of the parallelism and thus taking less processor reallocations. This is due to A-CONTROL's more effective processor desire calculation, which suggests that it probably performs better than A-GREEDY in practice.

(a)

(b)

Figure 3.3: Transient responses of A-CONTROL and A-GREEDY on (a) Step and (b) Poly(II) profiles.

3.5.2 Individual Job Performances

To verify the quality of feedbacks observed in the transient responses, we design one set of simulations to measure the individual job performances, that is, the response time and the processor waste of A-CONTROL and A-GREEDY. In the simulation, only one job with a single type of parallelism profile is run each time. This is specifically designed to study the impact of different parallelism variations on the task schedulers. In addition, the processor desires of both schedulers are always granted. This allows us to evaluate the performances of the task schedulers under a favorable circumstance, since otherwise the advantage of a more efficient processor desire calculation scheme can not be reflected. Figure 3.5 shows the effects of the five parallelism variations on the response time and the processor waste of both task schedulers. The response time is normalized in terms of the span of the job and the processor waste is normalized in

(a)

(b)

Figure 3.4: Transient responses of A-CONTROL and A-GREEDY on (a) Ramp and (b) Poly(I) profiles.

terms of the job's total work.

From the figure, we can see clearly that A-CONTROL indeed outperforms A-GREEDY with respect to the response time and the processor waste for all parallelism variations. Moreover, smoother parallelism variations with smaller transitions also lead to better performances of A-CONTROL, which matches our analysis in Section 3.4.1. For instance, A-CONTROL has better performance for the Step profile, which exhibits smaller parallelism transition. In contrast, the Ramp and Poly(I) profiles with steeper parallelism variations result in larger response time and processor waste for A-CONTROL. We should point out that although the Impulse profile has drastic parallelism variation, it occurs less frequently than the other profiles. Hence, A-CONTROL can easily capture its parallelism variation within one quantum under the unconstrained environment, thus has better response time. The performances of A-GREEDY, on the other hand, are relatively less sensitive to the different paral-

(a)

(b)

Figure 3.5: Individual job performances of A-CONTROL and A-GREEDY on the five parallelism profiles in terms of (a) response time and (b) processor waste.

lelism profiles, but are generally inferior to those of A-CONTROL even on jobs with larger parallelism transitions. In summary, the simulation results confirm our previous insight that a task scheduler with better transient responses should be able to achieve superior performances in terms of the response time and processor waste of an individual job.

Chapter 4

Total Response Time and Makespan on Fixed Speed Processors

In this chapter, we consider the performances of ACDEQ and AGDEQ with respect to the total response time and makespan for a set of jobs. As with the preceding chapter, we focus on the scenario where the speeds of all processors are identical and fixed throughout the execution of the jobs.

We show that minimizing the makespan is relatively straightforward for both algorithms, which achieve $O(1)$ -competitive regardless of the jobs' arrival patterns. For the total response time, however, the arrival patterns of the jobs do make a difference on the performances of the algorithms. In particular, we show that when jobs all arrive in a single batch, both ACDEQ and AGDEQ are $O(1)$ -competitive with respect to the total response time, even using the same amount of resources as the optimal offline scheduler. When jobs can have arbitrary arrival time, Motwani, Phillips and Torng [142] have shown that any non-clairvoyant algorithm using the same amount of resources as the optimal is at best $\Omega(n^{1/3})$ -competitive even for sequential jobs on a single processor, where n denotes the total number jobs in the job set. Hence, we focus on the resource augmentation analysis [104] in this case, and show that when using $O(1)$ times more resources than the optimal, both ACDEQ and AGDEQ achieve $O(1)$ -competitive with respect to the total response time.

Moreover, with respect to the total response time, we provide an analysis framework for any two-level scheduling algorithm. The framework is based on the *local competitive argument* and the *amortized local competitive argument* for batched and nonbatched jobs, respectively [154]. As we will see, these two general approaches play important roles in the rest of this thesis, and they will be used repeatedly in the next chapter and in Chapter 6 for the analysis of set response time and total response time plus energy of various scheduling algorithms. The content of this chapter was originally presented in [178, 173, 175].

4.1 Total Response Time of ACDEQ

In this section, we consider the total response time of the ACDEQ algorithm for a set of jobs. For simplicity, we assume that the convergence rate v of A-CONTROL is set to 0, and hence one-step convergence is achieved. We first derive two lower bounds on the total response time of any scheduler and define some preliminary concepts and notations. We then analyze the performance of ACDEQ for batched and nonbatched jobs, respectively.

4.1.1 Lower Bounds and Preliminaries

We first present two lower bounds on the total response time of any set of parallel jobs. One of them only applies to batched jobs and is known as the *squashed work* in the literature [181, 63, 71, 88]. The other lower bound, which is related to the total span of all jobs, applies to both batched or nonbatched cases. These two lower bounds are used to bound the performance of the optimal offline scheduler on unit-speed processors, and will help us conveniently derive the competitive ratios of the online algorithms through indirect comparison.

For any job set $\mathcal{J}$, we define the *total span* of all jobs in $\mathcal{J}$ to be $l(\mathcal{J}) = \sum_{i=1}^n l(J_i)$, and define the *squashed work* of the jobs in $\mathcal{J}$ to be $\hat{w}(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^n i \cdot w(J_{\pi(i)})$, where $\pi(\cdot)$ denotes a permutation of the jobs sorted in non-increasing work order, i.e., $w(J_{\pi(1)}) \geq w(J_{\pi(2)}) \geq \dots \geq w(J_{\pi(n)})$. The following lemma formally shows that the total span and the squashed work of the jobs in $\mathcal{J}$ can serve as two lower bounds for its total response time.

Lemma 4.1 *For any job set $\mathcal{J}$ to be scheduled on P processors of unit speed, the optimal total response time $R^*(\mathcal{J})$ satisfies*

$$\begin{aligned} R^*(\mathcal{J}) &\geq l(\mathcal{J}), \\ R^*(\mathcal{J}) &\geq \hat{w}(\mathcal{J}), \end{aligned}$$

where $l(\mathcal{J})$ and $\hat{w}(\mathcal{J})$ are the total span and the squashed work of all jobs in $\mathcal{J}$, respectively, and the second lower bound only applies to batched jobs.

Proof. As it takes at least the span of a job J_i , that is $l(J_i)$ time, to complete the job on unit-speed processors, completing job set $\mathcal{J}$ takes at least $R^*(\mathcal{J}) \geq \sum_{i=1}^n l(J_i) = l(\mathcal{J})$ time. Hence, the first lower bound is proved.

When all jobs in $\mathcal{J}$ are batch released, completing k jobs, where $1 \leq k \leq n$, takes at least $\frac{1}{P} \sum_{i=n-k+1}^n w(J_{\pi(i)})$ time on P processors of unit-speed. This is because no other schedule can produce better response time than executing k jobs that have the least amount of work on P processors without any waste of resources. The optimal total response time therefore satisfies $R^*(\mathcal{J}) \geq \frac{1}{P} \sum_{k=1}^n \sum_{i=n-k+1}^n w(J_{\pi(i)}) = \frac{1}{P} \sum_{i=1}^n i \cdot w(J_{\pi(i)}) = \hat{w}(\mathcal{J})$, proving the second lower bound. $\square$

We define some preliminary concepts and notations that will be used in my main analysis. Recall from Chapter 3 that at any time t , a job J_i scheduled by ACDEQ is either “accounted” or “deductible”. Let $\mathcal{J}(t)$ denote the set of active jobs at time t , and let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the sets of active jobs at time t that are accounted and deductible, respectively. Throughout the execution of job J_i , let $a_A(J_i)$ denote the processor allocation when job J_i is accounted, i.e., $a_A(J_i) = \int_0^\infty a_i(t)s \cdot [J_i(t) \in \mathcal{J}_A(t)]dt$, where s denotes the processor speed of the online algorithm and $[x]$ is 1 if proposition x is true and 0 otherwise, and let $t_B(J_i)$ denote the amount of time when job J_i is deductible, i.e., $t_B(J_i) = \int_0^\infty [J_i(t) \in \mathcal{J}_B(t)]dt$. To simplify our notations, we let $n_t = |\mathcal{J}(t)|$ denote the number of active jobs at time t , and let $n_t^A = |\mathcal{J}_A(t)|$ and $n_t^B = |\mathcal{J}_B(t)|$ denote the number of active jobs at time t that are accounted and deductible, respectively. Apparently, we have $n_t^A + n_t^B = n_t$.

Now, we introduce the concepts of *squashed accounted processor allocation* $\hat{a}_A(\mathcal{J})$ and *total deductible processing time* $t_B(\mathcal{J})$ for job set $\mathcal{J}$, which are defined as follows:

$$\hat{a}_A(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^n i \cdot a_A(J_{\gamma(i)}), \quad (4.1)$$

$$t_B(\mathcal{J}) = \sum_{i=1}^n t_B(J_i), \quad (4.2)$$

where $\gamma(\cdot)$ denotes a permutation of the jobs sorted in non-increasing order of accounted processor allocation, i.e., $a_A(\mathcal{J}_{\gamma(1)}) \geq a_A(\mathcal{J}_{\gamma(2)}) \geq \dots \geq a_A(\mathcal{J}_{\gamma(n)})$. It is not difficult to see that $\gamma(\cdot)$, among all permutations, gives the minimum value for the formulation of squashed accounted processor allocation, that is, $\sum_{i=1}^n i \cdot a_A(\mathcal{J}_{\gamma(i)}) \leq \sum_{i=1}^n i \cdot a_A(\mathcal{J}_{\pi(i)})$ for any permutation $\pi(\cdot)$ of the jobs. We will make use of this property shortly in the proof of Lemma 4.2, which derives the upper bounds for the squashed accounted processor allocation and the total deductible processing time respectively in terms of the squashed work and the total span of the jobs.

Lemma 4.2 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of n jobs on P processors of speed s , where $s > 0$. Then the squashed accounted processor allocation $\hat{a}_A(\mathcal{J})$ and the total deductible processing time $t_B(\mathcal{J})$ for the job set $\mathcal{J}$ satisfy*

$$\hat{a}_A(\mathcal{J}) \leq 2 \cdot \hat{w}(\mathcal{J}), \quad (4.3)$$

$$t_B(\mathcal{J}) \leq \frac{C+1}{s} \cdot l(\mathcal{J}), \quad (4.4)$$

where $\hat{w}(\mathcal{J})$ and $l(\mathcal{J})$ denote the squashed work and the total span of job set $\mathcal{J}$, and C denotes the maximum transition factor of all jobs in $\mathcal{J}$.

Proof. From the proof of Theorems 3.3 and 3.4 in Chapter 3, we get that for any job J_i scheduled by ACDEQ, the accounted processor allocation $a(J_i)$ and the deductible processing time $t_B(J_i)$ satisfy $a(J_i) \leq 2w(J_i)$ and $t_B(J_i) \leq \frac{C+1}{s} \cdot l(J_i)$.

For the job set $\mathcal{J}$ scheduled by ACDEQ, let $\gamma(\cdot)$ denote a permutation of the jobs sorted in non-increasing order of accounted processor allocation, and let $\pi(\cdot)$ denote a permutation of the jobs sorted in non-increasing order of work. The squashed accounted processor allocation is then given by $\hat{a}_A(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^n i \cdot a_A(J_{\gamma(i)}) \leq \frac{1}{P} \sum_{i=1}^n i \cdot a_A(J_{\pi(i)}) \leq \frac{1}{P} \sum_{i=1}^n i \cdot 2w(\mathcal{J}_{\pi(i)}) = 2\hat{w}(\mathcal{J})$. The total deductible processing time $t_B(\mathcal{J})$ for $\mathcal{J}$ is given by $t_B(\mathcal{J}) = \sum_{i=1}^n t_B(J_i) \leq \frac{C+1}{s} \sum_{i=1}^n l(J_i) = \frac{C+1}{s} \cdot l(\mathcal{J})$. $\square$

We also introduce the notions of *t-prefix* and *t-suffix* for jobs to ease our analysis. For the ACDEQ algorithm, define the *t-prefix* $J_i(\overleftarrow{t})$ of job J_i to be the portion of the job executed before and at time t , and define the *t-suffix* $J_i(\overrightarrow{t})$ to be the portion executed after time t . Specifically, if ACDEQ is executing the k -th phase of job J_i at time t , then $J_i(\overleftarrow{t})$ consists of the first $k-1$ phases $\langle J_i^1, J_i^2, \dots, J_i^{k-1} \rangle$ of job J_i , followed by part of the k -th phase with work $\int_{c_{i-1}}^t \Gamma_i^k(a_i(t))dt$; and $J_i(\overrightarrow{t})$ begins with the rest of the k -th phase with work $\int_t^{c_i^k} \Gamma_i^k(a_i(t))dt$, followed by the remaining phases $\langle J_i^{k+1}, J_i^{k+2}, \dots, J_i^{k_i} \rangle$ of job J_i . We can extend the notions of *t-prefix* and *t-suffix* from an individual job to a job set as follows: the *t-prefix* of job set $\mathcal{J}$ is given by $\mathcal{J}(\overleftarrow{t}) = \{J_i(\overleftarrow{t}) : J_i \in \mathcal{J} \text{ and } r_i \geq t\}$ and the *t-suffix* of $\mathcal{J}$ is $\mathcal{J}(\overrightarrow{t}) = \{J_i(\overrightarrow{t}) : J_i \in \mathcal{J} \text{ and } r_i \geq t\}$. We can also define $\mathcal{J}^*(\overleftarrow{t})$ and $\mathcal{J}^*(\overrightarrow{t})$ to be the *t-prefix* and the *t-suffix* in a similar way for the job set $\mathcal{J}$ executed by the optimal offline scheduler.

4.1.2 Analysis for Batched Jobs

In this section, we analyze the total response time of the ACDEQ algorithm when all jobs are released in a single batch. The analysis relies on the *local competitiveness argument* [154] by bounding the performance of an online algorithm at any time in terms of the optimal offline scheduler, or rather the two lower bounds given in Section 4.1.1, both of which represent the performance of the optimal in this scenario.

For any job set $\mathcal{J}$, our analysis focuses on its *t-prefix* $\mathcal{J}(\overleftarrow{t})$, which according to definition, always contains n jobs for any $t > 0$, since all jobs are batch released in this scenario. Recall that the squashed accounted processor allocation for $\mathcal{J}(\overleftarrow{t})$ is given by $\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P} \sum_{i=1}^n i \cdot a_A(J_{\gamma(i)}(\overleftarrow{t}))$, where $\gamma(\cdot)$ denotes a permutation of the jobs in $\mathcal{J}(\overleftarrow{t})$ sorted in non-increasing order of accounted processor allocation. At any time t , let $n_t(z)$ denote the number of jobs in $\mathcal{J}(\overleftarrow{t})$ whose accounted processor allocation is at least z under ACDEQ, i.e., $n_t(z) = \sum_{i=1}^n [a_A(J_i(\overleftarrow{t})) \geq z]$. Apparently, $n_t(z)$ is a staircase-like decreasing function of z , and Figure 4.1(a) shows an example of $n_t(z)$ at a particular time t . It is not difficult to see that an alternative formulation for the squashed accounted processor allocation $\hat{a}_A(\mathcal{J}(\overleftarrow{t}))$ is given by

$$\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P} \int_0^\infty \left(\sum_{i=1}^{n_t(z)} i \right) dz. \quad (4.5)$$

In the batched scenario, we give both the ACDEQ algorithm and the optimal offline

Figure 4.1: (a) An example of $n_t(z)$ at a particular time t . (b) The changes of $n_t(z)$ in an infinitesimal interval Δt in the worst case.

scheduler P processors of unit speed. The local performance of ACDEQ can then be shown in the following lemma.

Lemma 4.3 *Suppose that ACDEQ schedules a collection $\mathcal{J}$ of batched jobs on P processors of unit speed. Then the execution of the jobs satisfies the following running condition:*

$$n_t \leq 2 \left(\frac{d\hat{a}_A(\mathcal{J}(t))}{dt} + n_t^B \right), \quad (4.6)$$

where $\frac{d\hat{a}_A(\mathcal{J}(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\hat{a}_A(\mathcal{J}(t+\Delta t)) - \hat{a}_A(\mathcal{J}(t))}{\Delta t}$ denotes the rate of change for the squashed accounted processor allocation in an infinitesimal interval Δt during which no job completes.

Proof. Since an accounted job is also deprived by definition, according to the OS allocator DEQ, each deprived job gets at least P/n_t processors at time t . In the worst case, the n_t^A accounted jobs have the most accounted processor allocation so far among the n_t active jobs. As a result, in interval Δt during which no job set completes, each of the bottom n_t^A horizontal stripes from $n_t(z)$ grows by at least $\frac{P}{n_t} \Delta t$ as shown in Figure 4.1(b). The rate of change for the squashed accounted processor allocation can then be bounded by

$$\begin{aligned} \frac{d\hat{a}_A(\mathcal{J}(t))}{dt} &= \frac{1}{P\Delta t} \int_0^\infty \left[\binom{n_{t+\Delta t}(z)}{\sum_{i=1}^{n_t^A} i} - \binom{n_t(z)}{\sum_{i=1}^{n_t^A} i} \right] dz \\ &\geq \frac{1}{P\Delta t} \cdot \frac{n_t^A(n_t^A + 1)}{2} \cdot \frac{P}{n_t} \Delta t \\ &\geq \frac{x_t^2}{2} n_t, \end{aligned} \quad (4.7)$$

where $x_t = n_t^A/n_t$, and obviously $0 \leq x_t \leq 1$. Since a job set is either accounted or deductible, we have $n_t^B = (1 - x_t)n_t$. It can be easily verified that the running condition holds for all values of x_t by substituting Inequality (6.4) into it. $\square$

Lemma 4.3 essentially gives the local performance of the ACDEQ algorithm. We can combine it with the result of Lemma 4.2 for the total response time of ACDEQ in batched scenario.

Theorem 4.4 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of batched jobs on P processors of unit speed. Then the total response time $R_{\text{AC}}(\mathcal{J})$ of the job set satisfies*

$$R_{\text{AC}}(\mathcal{J}) \leq 2(C + 3) \cdot R^*(\mathcal{J}), \quad (4.8)$$

where $R^*(\mathcal{J})$ is the total response time of $\mathcal{J}$ under the optimal offline scheduler on unit-speed processors, and C denotes the maximum transition factor of all jobs in $\mathcal{J}$.

Proof. The total response time of any job set $\mathcal{J}$ scheduled by the ACDEQ algorithm can be expressed as $R_{\text{AC}}(\mathcal{J}) = \int_0^\infty n_t dt$. Similarly, the total deductible processing time of $\mathcal{J}$ under ACDEQ is given by $t_B(\mathcal{J}) = \int_0^\infty n_t^B dt$. Integrating the running condition in Lemma 4.3 over time, we have $R_{\text{AC}}(\mathcal{J}) \leq 2(\hat{a}_A(\mathcal{J}) + t_B(\mathcal{J}))$.

Substituting the bounds of the squashed accounted processor allocation $\hat{a}_A(\mathcal{J})$ and the total deductible processing time $t_B(\mathcal{J})$ from Lemma 4.2 into the above inequality, we get $R_{\text{AC}}(\mathcal{J}) \leq 2(2\hat{w}(\mathcal{J}) + (C + 1)l(\mathcal{J}))$. Based on Lemma 4.1, both squashed work $\hat{w}(\mathcal{J})$ and total span $l(\mathcal{J})$ are lower bounds for the total response time of batched jobs on unit-speed processors. The performance of ACDEQ satisfies $R_{\text{AC}}(\mathcal{J}) \leq 2(2R^*(\mathcal{J}) + (C + 1)R^*(\mathcal{J})) = 2(C + 3)R^*(\mathcal{J})$. $\square$

4.1.3 Analysis for Nonbatched Jobs

In this section, we analyze the total response time of ACDEQ when the jobs are nonbatched, i.e., have arbitrary release time. The analysis uses the *amortized local competitiveness argument* [154], which bounds the amortized performance of an online algorithm at any time in terms of the optimal scheduler through a potential function.

We adopt the potential function proposed by Lam et al. [117] in the context of online speed scaling. In this scenario, we focus on the t -suffix $\mathcal{J}(\vec{t})$ of job set $\mathcal{J}$, and redefine $n_t(z)$ to be the number of jobs in $\mathcal{J}(\vec{t})$ whose accounted processor allocation is at least $2z$ at time t under ACDEQ, i.e., $n_t(z) = \sum_{i=1}^n [a_A(J_i(\vec{t})) \geq 2z]$. Moreover, let $n_t^*(z)$ denote the number of job sets in $\mathcal{J}^*(\vec{t})$ whose work is at least z under the optimal, i.e., $n_t^*(z) = \sum_{i=1}^n [w(J_i^*(\vec{t})) \geq z]$. Hence, both $n_t(z)$ and $n_t^*(z)$ are staircase-like decreasing functions of z , and Figure 4.2(a) shows an example of $n_t(z)$ and $n_t^*(z)$ at a particular time t . For nonbatched job sets, we give the ACDEQ algorithm P processors of speed s , where $s = 4 + \epsilon$ for any $\epsilon > 0$, while the optimal scheduler uses unit-speed processors. The potential function is defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i \right) - n_t(z)n_t^*(z) \right] dz, \quad (4.9)$$

where $\eta = \frac{4}{\epsilon P}$. We will now prove the amortized local performance of ACDEQ in the following lemma.

Figure 4.2: (a) An example of $n_t(z)$ and $n_t^*(z)$ at a particular time t . (b) The changes of $n_t(z)$ and $n_t^*(z)$ after a new job with work w' and accounted processor allocation a'_A arrives. (c) The changes of $n_t(z)$ and $n_t^*(z)$ in an infinitesimal interval Δt in the worst case.

Lemma 4.5 Suppose that ACDEQ schedules a set $\mathcal{J}$ of job sets on P processors of speed s , where $s = 4 + \epsilon$ for any $\epsilon > 0$. Then given the potential function defined in Equation (4.9), the execution of the job set satisfies the following

- Boundary condition: $\Phi(0) \leq 0$ and $\Phi(\infty) \geq 0$;
- Arrival condition: $\Phi(t)$ does not increase when a new job arrives;
- Completion condition: $\Phi(t)$ does not increase when an existing job completes under either ACDEQ or the optimal;
- Running condition:

$$n_t + \frac{d\Phi(t)}{dt} \leq \frac{2s}{\epsilon} (n_t^* + n_t^B), \quad (4.10)$$

where $\frac{d\Phi(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Phi(t+\Delta t) - \Phi(t)}{\Delta t}$ denotes the rate of change for the potential function in an infinitesimal interval Δt during which no job arrives or completes under both ACDEQ and the optimal.

Proof. We check each of the four conditions in the following.

- *Boundary Condition:* at time 0, no job exists in the system. The terms $n_t(z)$ and $n_t^*(z)$ are both 0 for all z . Therefore, we have $\Phi(0) = 0$. At time ∞ , no job remains in the system, so again we have $\Phi(\infty) = 0$. Hence, the boundary condition holds.

- *Arrival Condition:* suppose that a new job with work w' arrives at time t . Let t^- and t^+ denote the instances right before and after the job arrives. Hence, we have $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w'$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w'$. Similarly, $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq a'_A/2$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > a'_A/2$, where a'_A is the accounted processor allocation to the job. Figure 4.2(b) illustrates the changes of $n_t(z)$ and $n_t^*(z)$ in this case. Note that $a'_A/2 \leq w'$ from Theorem 3.4. For convenience, let us define $\phi_t(z) = \left(\sum_{i=1}^{n_t(z)} i\right) - n_t(z)n_t^*(z)$. Thus, it is obvious that for $z > w'$, we

have $\phi_{t+}(z) = \phi_{t-}(z)$. For $z \leq w'$, we consider two cases.

Case 1: for $z \leq a'_A/2$, we have $\phi_{t+}(z) - \phi_{t-}(z) = \left(\sum_{i=1}^{n_{t-}(z)+1} i\right) - (n_{t-}(z) + 1)(n_{t-}^*(z) + 1) - \left(\sum_{i=1}^{n_{t-}(z)} i\right) + n_{t-}(z)n_{t-}^*(z) = -n_{t-}^*(z) \leq 0$.

Case 2: for $a'_A/2 \leq z \leq w'$, we have $\phi_{t+}(z) - \phi_{t-}(z) = \left(\sum_{i=1}^{n_{t-}(z)} i\right) - n_{t-}(z)(n_{t-}^*(z) + 1) - \left(\sum_{i=1}^{n_{t-}(z)} i\right) + n_{t-}(z)n_{t-}^*(z) = -n_{t-}(z) \leq 0$.

Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t+}(z) dz \leq \eta \int_0^\infty \phi_{t-}(z) dz = \Phi(t^-)$, and the arrival condition holds.

- *Completion Condition*: when a job completes under ACDEQ or the optimal, the potential function $\Phi(t)$ remains unchanged, because in such cases, neither $n_t(z)$ nor $n_t^*(z)$ changes for $z \geq 0$. Hence, the completion condition also holds.

- *Running condition*: Again, an accounted job is deprived by definition. According to the OS allocator DEQ, each deprived job gets at least P/n_t processors at time t . In the worst case, the n_t^A accounted jobs have the most remaining accounted processor allocation, while the optimal executes the job with the least remaining work using all P processors. Hence, as Figure 4.2(c) shows, each of the bottom n_t^A horizontal stripes from $n_t(z)$ shrinks by $\frac{sP}{2n_t}\Delta t$, and the top horizontal stripe from $n_t^*(z)$ shrinks by $P\Delta t$ in interval Δt . The rate of change for the potential function can then be bounded by

$$\begin{aligned}
\frac{d\Phi(t)}{dt} &= \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i \right) - n_{t+\Delta t}(z)n_{t+\Delta t}^*(z) - \left(\sum_{i=1}^{n_t(z)} i \right) + n_t(z)n_t^*(z) \right] dz \\
&\leq \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i \right) - \left(\sum_{i=1}^{n_t(z)} i \right) \right] dz \\
&\quad + \frac{\eta}{\Delta t} \int_0^\infty [n_t(z)(n_t^*(z) - n_{t+\Delta t}^*(z)) + n_t^*(z)(n_t(z) - n_{t+\Delta t}(z))] dz \\
&\leq \frac{4}{\epsilon P \Delta t} \left(-\frac{n_t^A(n_t^A + 1)}{2} \cdot \frac{sP}{2n_t}\Delta t + n_t P \Delta t + n_t^* \frac{sP n_t^A}{2n_t} \Delta t \right) \\
&\leq \frac{4}{\epsilon} \left(1 - \frac{s x_t^2}{4} \right) n_t + \frac{2s}{\epsilon} n_t^*, \tag{4.11}
\end{aligned}$$

where $x_t = n_t^A/n_t$, and $0 \leq x_t \leq 1$. Since a job set is either accounted or deductible, we have $n_t^B = (1 - x_t)n_t$. We can again verify that the running condition holds for all values of x_t by substituting Inequality (4.11) into Inequality (4.10). $\square$

Lemma 4.5 essentially gives the amortized local performance of the ACDEQ algorithm. We can combine it with the result of Lemma 4.2 for the total response time of ACDEQ in nonbatched scenario.

Theorem 4.6 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on P processors of speed s , where $s = 4 + \epsilon$ for any $\epsilon > 0$. Then the total response time $R_{AC}(\mathcal{J})$ for the job set satisfies*

$$R_{AC}(\mathcal{J}) \leq \left(2 + \frac{2C + 10}{\epsilon} \right) R^*(\mathcal{J}), \tag{4.12}$$

where $R^*(\mathcal{J})$ is the total response time of $\mathcal{J}$ under the optimal offline scheduler on unit-speed processors, and C denotes the maximum transition factor of all jobs in $\mathcal{J}$.

Proof. As the total response time of ACDEQ is given by $R_{AC}(\mathcal{J}) = \int_0^\infty n_t dt$, and the total response time of the optimal is given by $R^*(\mathcal{J}) = \int_0^\infty n_t^* dt$, integrating the running condition in Lemma 4.5 over time, we have $R_{AC}(\mathcal{J}) + \Phi(\infty) - \Phi(0) + \sum_{t \in T} (\Phi(t^-) - \Phi(t^+)) \leq \frac{2s}{\epsilon} (R^*(\mathcal{J}) + t_B(\mathcal{J}))$, where T denotes the set of time instances when a new job arrives or an existing job completes under either ACDEQ or the optimal. Applying the boundary, the arrival and the completion conditions, we get $R_{AC}(\mathcal{J}) \leq \frac{2s}{\epsilon} (R^*(\mathcal{J}) + t_B(\mathcal{J}))$. Note that $t_B(\mathcal{J}) = \int_0^\infty n_t^B dt$ is the total deductible processing time for $\mathcal{J}$ under ACDEQ. From Lemma 4.2, the total deductible processing time for $\mathcal{J}$ is also given by $t_B(\mathcal{J}) \leq \frac{C+1}{s} \cdot l(\mathcal{J})$. The total response time of $\mathcal{J}$ scheduled by ACDEQ thus satisfies $R_{AC}(\mathcal{J}) \leq \frac{2s}{\epsilon} (R^*(\mathcal{J}) + \frac{C+1}{s} \cdot l(\mathcal{J}))$. Since the total span $l(\mathcal{J})$ is the only lower bound on the total response time of $\mathcal{J}$ on unit-speed processors now, the theorem follows by substituting $R^*(\mathcal{J}) \geq l(\mathcal{J})$ into the above inequality and simplifying. $\square$

4.2 Analysis Framework for Total Response Time and Performance of AGDEQ

In this section, we provide a generalized framework for the total response time analysis of any algorithm that can be formulated in the two-level adaptive scheduling paradigm. Similarly to the analysis of ACDEQ, the key to the analysis framework is to study the efficiency of an algorithm when scheduling an individual job, which is specifically reflected in two performance bounds. In particular, for any two-level algorithm XY, which consists of the task scheduler X and the OS allocator Y, we need to identify two properties A and B , such as “accounted” and “deductible” in the case of ACDEQ. Then for any job J_i scheduled by XY on speed- s processors, let $a_A(J_i) = \int_0^\infty a_i(t)s \cdot [J_i(t) \in \mathcal{J}_A(t)]dt$ denote the job’s processor allocation with property A and let $t_B(J_i) = \int_0^\infty [J_i(t) \in \mathcal{J}_B(t)]dt$ denote the job’s processing time with property B , where $J_i(t)$ denotes job J_i at time t , and $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the sets of active jobs in $\mathcal{J}$ that satisfy properties A and B respectively at time t . We need to bound both the processor allocation $a_A(J_i)$ of a job with property A and the processing time $t_B(J_i)$ of a job with property B . The two bounds should be given in terms of the work and the span of the job for any $s > 0$ in the following form:¹

$$a_A(J_i) \leq \gamma_1 \cdot w(J_i), \quad (4.13)$$

$$t_B(J_i) \leq \frac{\gamma_2}{s} \cdot l(J_i). \quad (4.14)$$

¹If there are any additive constants for these bounds, such as $t_B(J_i) \leq \frac{\gamma_2}{s} \cdot l(J_i) + \gamma_3$ in the case of AGDEQ to be seen shortly, then these constants will be absorbed by the performance of the optimal scheduler when the jobs under consideration are sufficiently large.

The two properties A and B need not be disjoint. However, they are usually chosen to cover the whole set of active jobs, i.e., $\mathcal{J}_A(t) \cup \mathcal{J}_B(t) = \mathcal{J}(t)$. Hence, we have $n_t^A + n_t^B = n_t$. We will now show that as long as the OS allocator Y ensures fairness among the active jobs that satisfy property A at any time t by allocating at least P/n_t processors to each of these jobs, then the total response time of the jobs under XY can be bounded. The results are stated in the following lemma.

Lemma 4.7 *Suppose that XY schedules a set $\mathcal{J}$ of jobs. For any individual job $J_i \in \mathcal{J}$, if its processor allocation with a chosen property A satisfies Inequality (4.13), and its processing time with a chosen property B satisfies Inequality (4.14), then XY achieves*

- $2(\gamma_1 + \gamma_2)$ -competitive with respect to the total response time of batched jobs;
- $(2\gamma_1 + \epsilon)$ -speed $\left(2 + \frac{2(2\gamma_1 + \gamma_2)}{\epsilon}\right)$ -competitive for nonbatched jobs, where $\epsilon > 0$,

provided that the OS allocator Y allocates at least P/n_t processors to each active job that satisfies property A at any time t , and the two properties are chosen such that $\mathcal{J}_A(t) \cup \mathcal{J}_B(t) = \mathcal{J}(t)$.

Proof. Based on the derivation of Lemma 4.2, the squashed processor allocation $\hat{a}_A(\mathcal{J})$ with property A and the total processing time $t_B(\mathcal{J})$ with property B for any job set $\mathcal{J}$ scheduled by XY satisfy $\hat{a}_A(\mathcal{J}) \leq \gamma_1 \cdot \hat{w}(\mathcal{J})$ and $t_B(\mathcal{J}) \leq \gamma_2 \cdot l(\mathcal{J})$, where $\hat{w}(\mathcal{J})$ and $l(\mathcal{J})$ denote the squashed work and the total span of all job sets in $\mathcal{J}$, respectively. Following the lead of Section 4.1.2, we can get $R_{XY}(\mathcal{J}) \leq 2(\gamma_1 + \gamma_2) \cdot R^*(\mathcal{J})$ for any set of batched job set $\mathcal{J}$, and following the potential function as well as the analysis of Section 4.1.3, we get $R_{XY}(\mathcal{J}) \leq \frac{2s}{\epsilon}(1 + \frac{\gamma_2}{s}) \cdot R^*(\mathcal{J})$ when using $(2\gamma_1 + \epsilon)$ times more speed processors than the optimal for nonbatched jobs. Hence, the lemma is proved. $\square$

Now, based on the results of Lemma 4.7 and the properties of the AGDEQ algorithm, we prove the performance of AGDEQ with respect to the total response time. Specifically, we choose properties A and B to be “accounted” and “deductible” for AGDEQ as well, and according to Theorems 3.5 and 3.6, each individual job J_i scheduled by the AGDEQ algorithm satisfies ²

$$a_A(J_i) \leq \frac{1}{\delta} \cdot w(J_i), \quad (4.15)$$

$$t_B(J_i) \leq \frac{2}{s(1 - \delta)} \cdot l(J_i) + o(1), \quad (4.16)$$

on speed- s processors, where $s > 0$. The performance of AGDEQ is then given in the following theorem.

²As mentioned previously, the additive constant for the deductible processing time, which is represented as $o(1)$ in Inequality (4.16), will be absorbed by the performance of the optimal scheduler when the job is sufficiently large.

Theorem 4.8 *Suppose that AGDEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then AGDEQ achieves*

- $\frac{2(1+\delta)}{\delta(1-\delta)}$ -competitive with respect to the total response time if jobs in $\mathcal{J}$ are batched;
- $(\frac{2}{\delta} + \epsilon)$ -speed $(2 + \frac{4}{\delta(1-\delta)\epsilon})$ -competitive for nonbatched jobs, where $\epsilon > 0$,

and δ is the utilization parameter of A-GREEDY.

Proof. The theorem is directly implied by applying Lemma 4.7 with the properties of AGDEQ given in Inequalities (4.15) and (4.16). $\square$

4.3 Makespan of ACDEQ and AGDEQ

In this section, we analyze the makespan of ACDEQ and AGDEQ on any set of parallel jobs. Specifically, we show that the makespan depends on both the running time and the processor waste of an individual job, and can be obtained by combining both bounds. We first derive two lower bound on the makespan of any job set under the optimal offline scheduler, and then compare the performance of the two-level adaptive algorithms with the two lower bounds.

Lemma 4.9 *For any job set $\mathcal{J}$ to be scheduled on P processors of unit speed, the optimal makespan $M^*(\mathcal{J})$ satisfies*

$$\begin{aligned} M^*(\mathcal{J}) &\geq \max_{i=1 \dots n} (l(J_i) + r_i), \\ M^*(\mathcal{J}) &\geq \frac{1}{P} \sum_{i=1}^n w(J_i), \end{aligned}$$

where $l(J_i)$ and $w(J_i)$ are the span and the work of any job $J_i \in \mathcal{J}$, respectively, and r_i denotes the release time of job J_i .

Proof. As it takes at least the span of a job J_i , that is $l(J_i)$ time, to complete the job on unit-speed processors since its release time, completing job set $\mathcal{J}$ takes at least $\max_{i=1 \dots n} (l(J_i) + r_i)$ time. Hence, the first lower bound is proved.

For the second lower bound, since there is no better schedule than executing all jobs on P unit-speed processors without any waste of resources, the optimal makespan satisfies $M^*(\mathcal{J}) \geq \frac{1}{P} \sum_{i=1}^n w(J_i)$. Hence, the second lower bound is also proved. $\square$

In the following theorem, we first give the makespan performance of the ACDEQ algorithm.

Theorem 4.10 *Suppose that ACDEQ schedules a set $\mathcal{J}$ of jobs on unit-speed processors. Then the makespan $M_{AC}(\mathcal{J})$ of the job set is bounded by*

$$M_{AC}(\mathcal{J}) \leq 2(C + 1) \cdot M^*(\mathcal{J}), \quad (4.17)$$

where $M^*(\mathcal{J})$ is the makespan of the optimal offline scheduler on unit-speed processors, and C is the maximum transition factor of all jobs in the job set.

Proof. Suppose that J_k is the last completed job in $\mathcal{J}$ scheduled by ACDEQ. Then the makespan $M_{AC}(\mathcal{J})$ of job set $\mathcal{J}$ is given by the completion time c_k of the job J_k . Let $R'(J_k)$ and $R''(J_k)$ denote the total satisfied time and the total deprived time of J_k , respectively. Then the completion time of the job is given by $c_k = r_k + R'(J_k) + R''(J_k)$.

We bound the total satisfied time and the total deprived time of job J_k , separately. Since a satisfied job is also deductible by definition, from Theorem 3.3, the total satisfied time of J_k is bounded by $R'(J_k) \leq (C + 1) \cdot l(J_k)$, where C is the maximum transition factor from all jobs in $\mathcal{J}$. Since the OS allocator DEQ allocates all P processors when job J_k is deprived, the amount of processing power $R''(J_k) \cdot P$ is either spent on work or wasted. According to Theorem 3.4, we have $R''(J_k) \cdot P \leq \sum_{i=1 \dots n} (w(J_i) + X(J_i)) \leq \sum_{i=1 \dots n} (w(J_i) + Cw(J_i)) = (C + 1) \sum_{i=1 \dots n} w(J_i)$. The total deprived time of J_k is bounded by $R''(J_k) \leq \frac{C+1}{P} \sum_{i=1 \dots n} w(J_i)$. The makespan $M_{AC}(\mathcal{J})$ of job set $\mathcal{J}$, which is equal to the completion time c_k of job J_k , is then $M_{AC}(\mathcal{J}) = c_k = r_k + R'(J_k) + R''(J_k) \leq r_k + (C + 1) \cdot l(J_k) + \frac{C+1}{P} \sum_{i=1 \dots n} w(J_i) \leq (C + 1) (l(J_k) + r_k) + \frac{C+1}{P} \sum_{i=1 \dots n} w(J_i)$. Since both $l(J_k) + r_k \leq \max_{i=1 \dots n} (l(J_i) + r_i)$ and $\frac{1}{P} \sum_{i=1 \dots n} w(J_i)$ are lower bound on the makespan of job set $\mathcal{J}$ according to Lemma 4.9, we have $M_{AC}(\mathcal{J}) \leq 2(C + 1) \cdot M^*(\mathcal{J})$, hence proving the theorem. $\square$

Now, based on the prove of Theorem 4.10, we provide a generalized framework for the makespan analysis of any two-level adaptive algorithm. As mentioned previously, the key is to study the performance of an algorithm when scheduling an individual job. In particular, for algorithm XY, we need to identify a properties B , such as “satisfied” in the case of ACDEQ. Then for any job J_i scheduled by XY on unit-speed processors, let $a(J_i) = \int_0^\infty a_i(t)dt$ denote the job’s total processor allocation and let $t_B(J_i) = \int_0^\infty [J_i(t) \in \mathcal{J}_B(t)]dt$ denote the job’s processing time with property B , where $J_i(t)$ denotes job J_i at time t , and $\mathcal{J}_B(t)$ denote the sets of active jobs in $\mathcal{J}$ that satisfy property B at time t . We need to bound both the total processor allocation $a(J_i)$ of a job and the processing time $t_B(J_i)$ of a job with property B . The two bounds should be given in terms of the work and the span of the job in the following form:

$$a(J_i) \leq \gamma_1 \cdot w(J_i), \quad (4.18)$$

$$t_B(J_i) \leq \gamma_2 \cdot l(J_i). \quad (4.19)$$

We will now show that as long as the OS allocator Y does not idle any processor, that is, it allocates all available processors to jobs, when at least one job does not satisfy property B , then the makespan of the jobs under XY can be bounded. The results are stated in the following lemma.

Lemma 4.11 *Suppose that XY schedules a set $\mathcal{J}$ of jobs. For any individual job $J_i \in \mathcal{J}$, if its total processor allocation satisfies Inequality (4.18), and its processing*

time with a chosen property B satisfies Inequality (4.19), then XY achieves $(\gamma_1 + \gamma_2)$ -competitive with respect to the makespan, provided that the OS allocator Y does not idle any processor when at least one job does not satisfy property B .

Proof. Following the proof of Theorem 4.10, let J_k denote the last completed job in $\mathcal{J}$ scheduled by XY . Then the makespan of job set $\mathcal{J}$ is given by $M_{XY}(\mathcal{J}) = r_k + R'(J_k) + R''(J_k)$, where $R'(J_k)$ denotes the time when J_k satisfies property B and $R''(J_k)$ denotes the time when J_k does not satisfy property B .

According to Inequality (4.19), the time when J_k satisfies property B is given by $R'(J_k) = t_B(J_k) \leq \gamma_2 \cdot l(J_k)$. Since the OS allocator Y does not idle any processor when J_k does not satisfy property B , we have that $R''(J_k)P \leq \sum_{i=1 \dots n} a(J_i) \leq \sum_{i=1 \dots n} \gamma_1 \cdot w(J_i)$. Hence, the time when J_k does not satisfy property B is given by $R''(J_k) \leq \frac{\gamma_1}{P} \sum_{i=1 \dots n} w(J_i)$. The makespan thus satisfies $M_{XY}(\mathcal{J}) = r_k + R'(J_k) + R''(J_k) \leq \gamma_2 \cdot (l(J_k) + r_k) + \frac{\gamma_1}{P} \sum_{i=1 \dots n} w(J_i)$, which proves the theorem since both $l(J_k) + r_k$ and $\frac{1}{P} \sum_{i=1 \dots n} w(J_i)$ are lower bound on the makespan of job set $\mathcal{J}$ according to Lemma 4.9. $\square$

With the help of Lemma 4.11, we can prove the makespan of the AGDEQ algorithm, which is stated in the following theorem.

Theorem 4.12 *Suppose that AGDEQ schedules a set $\mathcal{J}$ of jobs on unit-speed processors. Then the makespan $M_{AG}(\mathcal{J})$ of the job set is bounded by*

$$M_{AG}(\mathcal{J}) \leq \left(\frac{1+\rho}{\delta} + \frac{2}{1-\delta} \right) \cdot M^*(\mathcal{J}), \quad (4.20)$$

where $M^*(\mathcal{J})$ is the makespan of the optimal offline scheduler on unit-speed processors, and δ and ρ are the utilization parameter and the responsiveness parameter of A-GREEDY, respectively.

Proof. Since for any individual job J_i scheduled by AGDEQ, the total processor allocation $a(J_i)$ and the total satisfied time $t_B(J_i)$, according to Theorems 3.5 and 3.6, are given by $a(J_i) \leq \frac{1+\rho}{\delta} \cdot w(J_i)$ and $t_B(J_i) \leq \frac{2}{1-\delta} \cdot l(J_i) + o(1)$, the theorem is proved by directly applying Lemma 4.11. $\square$

4.4 Discussions

We have analyzed the performances of the ACDEQ and AGDEQ algorithms with respect to both total response time and makespan for a set of jobs. From the various bounds, we can see that the performances of ACDEQ are closely related to the transition factor of the jobs, which can be considered as constant if the parallelism of the jobs does not change frequently, hence exhibits smooth transition. On the other hand, the performances of AGDEQ largely depend upon the choice of its parameters, or

specifically the utilization parameter and the responsiveness parameter of A-GREEDY. These parameters can also be considered as constant once chosen. Hence, both ACDEQ and AGDEQ achieve competitive performances with respect to the total response time and makespan. In case that the jobs are nonbatched, ACDEQ and AGDEQ also achieve competitive performances using $O(1)$ times more resources than the optimal offline scheduler.

It is worth noting that as far as the total response time performance is concerned, both ACDEQ and AGDEQ have larger competitive ratios than the EQUI scheduler [71, 70], which implies that the two-level adaptive schedulers exhibit inferior total response time performance for a set of jobs in the worst case. The reason is because two-level adaptive schedulers only utilize the history of the jobs to generate feedbacks and we do not assume that the job’s future parallelism is correlated to its past. Hence, in the worst case, the adversary can always make the future parallelism of the job deviate from its processor desire, e.g., make the job have high parallelism when its processor desire is low and vice versa. Thus, the OS allocator DEQ can be tricked into making poorer decisions, resulting in worse processor distributions. In practice, however, such worst-case scenario is not likely to occur. Therefore, we expect that ACDEQ and AGDEQ should perform comparably to or even better than EQUI, especially when the parallelism of the jobs does not change frequently, hence the correlation between the future and the past parallelism can be well explored by the adaptive strategies. We will confirm this observation in the next section by comparing the performances of ACDEQ and AGDEQ with that of EQUI through simulations.

4.5 Empirical Evaluations

We now empirically evaluate the performances of ACDEQ and AGDEQ, and compare them with EQUI in terms of the total response time and makespan for a set of jobs. We set the system load proportionally to the number of jobs as well as their arrival rate based on Downey’s model [67] for evaluating total response time. For makespan, we use batched job set and define the load to be proportional to the number of jobs (more precisely, the load is defined to be the number of jobs divided by 100 in this case), since otherwise the makespan could be dominated by the release time of the last job from a large job stream.

As can be observed in Figure 4.3, ACDEQ and AGDEQ generally achieve better total response time and makespan performances than EQUI. Only when the system has light loads with a small number of jobs, EQUI performs better because in this case all jobs can be easily satisfied on the given processors. With increased loads, however, both ACDEQ and AGDEQ outperform EQUI, and eventually tend to converge to EQUI at heavy loads, where each job gets very few processors most of the time and hence the advantage of adaptive scheduling is diminished. This suggests that two-level adaptive

(a)

(b)

Figure 4.3: The performance comparisons of ACDEQ and AGDEQ with the EQUI scheduler in terms of (a) makespan and (b) total response time.

scheduling is more effective under moderate loads with many parallel jobs competing for but not overwhelmed by the limited processor resources.

Moreover, we also specifically compare the performances of the two-level schedulers. In particular, we capture the number of processor reallocations both schedulers incur during running time, which are used to measure practical scheduling overheads. The response time of a job is then assumed to increase by an additive factor $\gamma \cdot \chi$, where χ denotes its total number of processor reallocations under a particular scheduler and γ depends on the system's physical overhead for context switching.

Figure 4.4 shows the number of processor reallocations of ACDEQ and AGDEQ with different system loads. We can see from Figure 4.4(a) that under light loads both schedulers produce a large number of processor reallocations on the last completed job, which contributes to the makespan of the job set. With increased loads, however, the processor desires of both schedulers cannot be easily granted, and this eventually

(a)

(b)

Figure 4.4: The number of processor reallocations of ACDEQ and AGDEQ on (a) the last completed job and (b) all jobs in the job set.

leads to less number of reallocations. On the other hand, the total number of reallocations of all jobs is shown in Figure 4.4(b), and it is related to the total response time of the job set. As can be seen, the total number of reallocations first increases because of more jobs joining the system before it starts to decrease, which is also due to the depression of the processor desires under higher system loads. Moreover, the number of processor reallocations of ACDEQ is always less than that of AGDEQ under all system loads because of A-CONTROL's more effective processor desire calculation. Furthermore, we can see from Figure 4.5 that ACDEQ performs better than AGDEQ under light to medium loads in terms of both makespan and total response time, especially when the cost of processor reallocations becomes high. This again demonstrates that A-CONTROL is able to provide more effective processor desires. Under heavy

(a)

(b)

Figure 4.5: The performance comparison of ACDEQ and AGDEQ with different cost for reallocation overhead in terms of (a) makespan and (b) total response time.

system loads, however, the processor desires tend to be deprived and the advantage of A-CONTROL disappears since neither schedulers have direct control over the processor allocations. Therefore, the performances of both schedulers are similar in such case.

Chapter 5

Set Response Time on Fixed Speed Processors

In this chapter, we consider scheduling for sets of parallel jobs, again on fixed speed processors, and the objective is to minimize the total response time of all job sets, or the *set response time*. This metric was first introduced by Robert and Schabanel [160, 161] to measure the overall fairness and efficiency of a scheduling algorithm. In particular, we consider a collection of job sets, where each job set contains one or more parallel jobs. We assume that all jobs within a set are released at the same time and the job set is considered complete when all jobs in the set are completed. The response time of a job set is defined to be the duration between its release time and completion. This model can represent a typical real-life scenario, where many users share a large cluster of processor resources, and each user may simultaneously submit one or more parallel jobs into the system. The set response time measures the overall waiting time of all users. To schedule a collection of job sets, an algorithm needs to decide the processor allocations at both the *job set level* and the *job level*. That is, it needs to specify at any time the number of processors allocated to each job set, as well as the processor allocation for each job within the job set.

In fact, the objective of set response time in a sense combines both total response time and makespan considered in the preceding chapter [160]. Suppose that each job set in the collection only contains a single job, then the set response time becomes the total response time of all jobs in the collection. On the other hand, if the collection only contains a single job set, then the set response time is simply the makespan of all jobs. In [160], Robert and Schabanel applied the EQUI scheduler [180] to both job set level and job level by equally dividing the total number of processors to all active job sets and within each job set equally dividing the allocated processors to the active jobs. They showed that the resulting algorithm $\text{EQUI} \circ \text{EQUI}$ is $\Theta(\frac{\ln n}{\ln \ln n})$ -competitive with respect to the set response time for batched job sets, where n is the maximum number of jobs in any set. Since it is known that EQUI is $O(1)$ -competitive with respect to the total response time [71] for batched jobs and $O(\frac{\ln n}{\ln \ln n})$ -competitive with respect

to the makespan [160] for a set of n jobs, the performance of $\text{EQUI} \circ \text{EQUI}$, confirms the observation that set response time combines the objectives of total response time and makespan. Hence, it is important to retain both fairness among the job sets and efficiency within each job set when minimizing set response time.

In this chapter, we combine EQUI with the two-level adaptive schedulers ACDEQ and AGDEQ to form $\text{EQUI} \circ \text{ACDEQ}$ and $\text{EQUI} \circ \text{AGDEQ}$ algorithms. We show that with the fairness of EQUI and the efficiency of the two-level schedulers, our proposed algorithms exhibit competitive performances with respect to the set response time. Specifically, we show that both $\text{EQUI} \circ \text{ACDEQ}$ and $\text{EQUI} \circ \text{AGDEQ}$ achieve $O(1)$ -competitive for batched job sets, and using $O(1)$ times more resources than the optimal, they also achieve $O(1)$ -competitive for nonbatched job sets. As mentioned in Chapter 4, the analysis presented in this chapter borrows heavily from the framework developed when we analyze the total response time performance. In this chapter, however, instead of first analyzing ACDEQ -based algorithm, we first analyze the set response time of $\text{EQUI} \circ \text{AGDEQ}$, and then present a generalized analysis for the $\text{EQUI} \circ Y$ family of algorithms, followed by its application to $\text{EQUI} \circ \text{ACDEQ}$. The content of this chapter was originally presented in [172].

5.1 Preliminaries and Lower Bounds

In this section, we give a formal description of the problem and the metric of set response time. Following the analysis of the total response time, we also develop two lower bounds on the set response time of any scheduling algorithm, and introduce some preliminary concepts and notations.

We consider a collection $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_m\}$ of m job sets, where each job set $\mathcal{J}_i = \{J_{i1}, J_{i2}, \dots, J_{in_i}\}$ has n_i parallel jobs with time-varying parallelism as described in Chapter 1. For each job J_{ij} , let $w(J_{ij})$ denote the *work* of the job, and let $l(J_{ij})$ denote the *span* of the job. Moreover, for each job set $\mathcal{J}_i$, we define its *set work* to be $w(\mathcal{J}_i) = \sum_{j=1}^{n_i} w(J_{ij})$ and define its *set span* to be $l(\mathcal{J}_i) = \max_{j=1 \dots n_i} l(J_{ij})$.

At any time t , given a total number P of processors, a scheduling algorithm needs to decide the processor allocation $a_i(t)$ for each job set $\mathcal{J}_i$, as well as the processor allocation $a_{ij}(t)$ for each job J_{ij} within job set $\mathcal{J}_i$, where $1 \leq i \leq m$ and $1 \leq j \leq n_i$. Of course, we require that the total processor allocation cannot exceed the total number of available processors at any time, i.e., $\sum_{i=1}^m \sum_{j=1}^{n_i} a_{ij}(t) \leq P$. Let r_{ij} denote the *release time* of job J_{ij} and let r_i denote the *release time* of job set $\mathcal{J}_i$. We assume that all jobs in a job set are released at the same time, i.e., $r_{ij} = r_i$ for all $1 \leq j \leq n_i$. Moreover, if all job sets are released in a single *batch*, then their release times are equal to 0. Otherwise, we can assume without loss of generality that the first released job set arrives at time 0. Let c_{ij} denote the *completion time* of job J_{ij} . The *completion time* c_i of job set $\mathcal{J}_i$ is therefore given by $c_i = \max_{j=1 \dots n_i} c_{ij}$. The *response time* $F(\mathcal{J}_i)$

of job set $\mathcal{J}_i$ is given by $F(\mathcal{J}_i) = c_i - r_i$.

Our objective is to minimize the total response time of all job sets, or the *set response time* $F(\mathcal{J})$, which is given by $F(\mathcal{J}) = \sum_{i=1}^m F(\mathcal{J}_i)$. Recall that a job J_{ij} is said to be *active* at time t if it has been released but not completed at t . We say that a job set $\mathcal{J}_i$ is *active* at time t if it contains at least one active job at t . An alternative expression for the set response time is thus given by $F(\mathcal{J}) = \int_0^\infty m_t dt$, where m_t denotes the number of active job sets at time t . As pointed out in [160], the set response time of a job set collection $\mathcal{J}$ has the following interesting property: if $\mathcal{J} = \{\mathcal{J}_1\}$ contains a single job set, then the set response time is simply the makespan of all jobs in $\mathcal{J}$; if $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_m\}$ contains a collection of singleton job sets, where for each $i = 1 \dots m$, we have $\mathcal{J}_i = \{J_{i1}\}$, then the set response time is the total response time of all jobs in $\mathcal{J}$. Hence, the objective of set response time in a sense combines both makespan and total response time of the jobs in $\mathcal{J}$.

We now present two lower bounds on the set response time of any job set collection. These lower bounds resemble the corresponding lower bounds on the total response time of a single job set presented in Chapter 4. Again, one of them applies to the batched scenario, where all job sets are released at the same time, and the other lower bound applies to both batched or nonbatched scenarios. For any job set collection $\mathcal{J}$, we define the *total span* of all job sets in $\mathcal{J}$ to be $l(\mathcal{J}) = \sum_{i=1}^m l(\mathcal{J}_i)$, and define the *squashed work* of the job sets in $\mathcal{J}$ to be $\hat{w}(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^m i \cdot w(\mathcal{J}_{\pi(i)})$, where $\pi(\cdot)$ denotes a permutation of the job sets sorted in non-increasing work order, i.e., $w(\mathcal{J}_{\pi(1)}) \geq w(\mathcal{J}_{\pi(2)}) \geq \dots \geq w(\mathcal{J}_{\pi(m)})$. The following lemma shows the two lower bounds.

Lemma 5.1 *For any job set collection $\mathcal{J}$ to be scheduled on P processors of unit speed, the optimal set response time is at least the total span of the job sets in $\mathcal{J}$, i.e., $F^*(\mathcal{J}) \geq l(\mathcal{J})$; and if all job sets are batched, the optimal set response time also satisfies $F^*(\mathcal{J}) \geq \hat{w}(\mathcal{J})$, where $\hat{w}(\mathcal{J})$ denotes the squashed work of the job sets in $\mathcal{J}$.*

Proof. As it takes at least the span of job J_{ij} , that is $l(J_{ij})$ time, to complete the job on unit-speed processors, according to the definition of set span, completing job set $\mathcal{J}_i$ takes at least $l(\mathcal{J}_i)$ time. Thus, the optimal set response time of any job set collection $\mathcal{J}$ satisfies $F^*(\mathcal{J}) \geq \sum_{i=1}^m l(\mathcal{J}_i) = l(\mathcal{J})$.

When all job sets in $\mathcal{J}$ are batched, completing k sets of jobs, where $1 \leq k \leq m$, takes at least $\frac{1}{P} \sum_{i=m-k+1}^m w(\mathcal{J}_{\pi(i)})$ time on P processors of unit-speed. This is because no other schedule can produce better completion time than executing k job sets that have the least amount of work on P processors without any waste of resources. The optimal set response time therefore satisfies $F^*(\mathcal{J}) \geq \frac{1}{P} \sum_{k=1}^m \sum_{i=m-k+1}^m w(\mathcal{J}_{\pi(i)}) = \frac{1}{P} \sum_{i=1}^m i \cdot w(\mathcal{J}_{\pi(i)}) = \hat{w}(\mathcal{J})$. $\square$

Finally, recall the notions of t -prefix and t -suffix for an individual job as well as for a job set defined in Chapter 4. We can extend these notions from a job set to the

entire job set collection as follows: the t -prefix of job set collection $\mathcal{J}$ is defined to be $\mathcal{J}(\overleftarrow{t}) = \{\mathcal{J}_i(\overleftarrow{t}) : \mathcal{J}_i \in \mathcal{J} \text{ and } r_i \geq t\}$ and the t -suffix of $\mathcal{J}$ is $\mathcal{J}(\overrightarrow{t}) = \{\mathcal{J}_i(\overrightarrow{t}) : \mathcal{J}_i \in \mathcal{J} \text{ and } r_i \geq t\}$. We can also define $\mathcal{J}^*(\overleftarrow{t})$ and $\mathcal{J}^*(\overrightarrow{t})$ to be the t -prefix and the t -suffix in a similar way for the job set collection $\mathcal{J}$ executed by the optimal offline scheduler.

5.2 EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ

In this section, we present our adaptive algorithms EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ for scheduling sets of parallel jobs. Both algorithms employ the well-known scheduler EQUI [180] to partition the total number of processors among the job sets. Within each job set, the two-level algorithms ACDEQ and AGDEQ are used to distribute the processors among the jobs.

EQUI (Equi-partition) scheduler was first introduced by Tucker and Gupta [180] as a process control policy for the parallel applications on multiprocessors. From algorithmic perspective, it simply divides the total number of processors evenly among all active job set at any time. Suppose that at time t there are m_t active job sets. Then for each active job set $\mathcal{J}_i$, the number $a_i(t)$ of processors allocated to the job set by EQUI is shown in Algorithm 4.

Algorithm 4 EQUI

Require: number m_t of active job sets at time t and total number P of processors

Ensure: processor allocation $a_i(t)$ for each active job set $\mathcal{J}_i$ at time t

1: $a_i(t) = P/m_t$.

The EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ algorithms then simply combine the EQUI scheduler that allocate the processors at the job set level with the two-level schedulers ACDEQ and AGDEQ presented in Chapter 3 that distribute the allocated processors of each job set among its active jobs. From Algorithm 4, we can see that EQUI only reallocates processors among the job sets when a new set is released into the system or an existing set completes execution. To synchronize the scheduling decisions made by EQUI and the two-level algorithms, we apply the concept of scheduling quanta to EQUI as well. Hence, if a new job set is released in the middle of a quantum, it is not scheduled until the beginning of the next quantum, and if an existing job set is completed in the middle of a quantum, it is not considered complete until the end of the quantum.

In the subsequent two sections, we will first analyze the set response time of the EQUI $\circ$ AGDEQ algorithm. In Section 5.5, we provide a generalized analysis for the EQUI $\circ$ Y family of algorithms and apply it to the EQUI $\circ$ ACDEQ algorithm. The rest of this section is devoted to proving a couple of properties of the EQUI $\circ$ AGDEQ algorithm, which will be useful to the later analysis.

Recall the notions of satisfied and deprived for a job scheduled by AGDEQ as shown in Chapter 3. We can extend these notions from an individual job to a job set as follows. A job set $\mathcal{J}_i$ is said to be *satisfied* at time t if *all* jobs in $\mathcal{J}_i$ are satisfied at t ; otherwise, $\mathcal{J}_i$ is said to be *deprived* if it contains *at least one* deprived job at t . Let $\mathcal{J}_i(t)$ denote job set $\mathcal{J}_i$ at time t , and let $\mathcal{J}(t)$ denote the set of all active job sets at t . Moreover, let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the set of deprived job sets and the set of satisfied job sets in $\mathcal{J}(t)$, respectively. Throughout the execution of job set $\mathcal{J}_i$, we define $a_A(\mathcal{J}_i)$ to be the *deprived processor allocation*, i.e., $a_A(\mathcal{J}_i) = \int_0^\infty a_i(t) \cdot [\mathcal{J}_i(t) \in \mathcal{J}_A(t)] dt$, and define $t_B(\mathcal{J}_i)$ to be the *satisfied processing time*, i.e., $t_B(\mathcal{J}_i) = \int_0^\infty [\mathcal{J}_i(t) \in \mathcal{J}_B(t)] dt$, where $[x]$ is 1 if proposition x is true and 0 otherwise. To simplify notations, let $m_t^A = |\mathcal{J}_A(t)|$ and $m_t^B = |\mathcal{J}_B(t)|$ denote the number of deprived job sets and the number of satisfied job sets at time t , respectively. Since an active job set is either satisfied or deprived, we have $m_t^A + m_t^B = m_t$.

We also define the concepts of *squashed deprived processor allocation* $\hat{a}_A(\mathcal{J})$ and *total satisfied processing time* $t_B(\mathcal{J})$ for the entire job set collection $\mathcal{J}$, which are given by $\hat{a}_A(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\gamma(i)})$ and $t_B(\mathcal{J}) = \sum_{i=1}^m t_B(\mathcal{J}_i)$, where $\gamma(\cdot)$ denotes a permutation of the job sets sorted in non-increasing order of deprived processor allocation, i.e., $a_A(\mathcal{J}_{\gamma(1)}) \geq a_A(\mathcal{J}_{\gamma(2)}) \geq \dots \geq a_A(\mathcal{J}_{\gamma(m)})$. Again, we can see that $\gamma(\cdot)$, among all permutations of the job sets, gives the minimum value for the formulation of squashed deprived processor allocation, that is, $\sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\gamma(i)}) \leq \sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\pi(i)})$ for any permutation $\pi(\cdot)$ of the job sets. Now, as with the total response time analysis, we derive the upper bounds for the squashed deprived processor allocation and the total satisfied processor time in terms of the squashed work and the total span for a job set collection.

Lemma 5.2 *Suppose that EQUIOAGDEQ schedules a collection $\mathcal{J}$ of m job sets on P processors of speed s , where $s > 0$. Then the squashed deprived processor allocation $\hat{a}_A(\mathcal{J})$ and the total satisfied processor time $t_B(\mathcal{J})$ for the collection $\mathcal{J}$ satisfy*

$$\hat{a}_A(\mathcal{J}) \leq \frac{1 + \rho}{\delta} \cdot \hat{w}(\mathcal{J}), \quad (5.1)$$

$$t_B(\mathcal{J}) \leq \frac{2}{s(1 - \delta)} \cdot l(\mathcal{J}) + o(1), \quad (5.2)$$

where $\hat{w}(\mathcal{J})$ and $l(\mathcal{J})$ denote the squashed work and the total span of $\mathcal{J}$, and δ and ρ denote A-GREEDY's utilization parameter and responsiveness parameter.

Proof. According to Theorems 3.5 and 3.6, for any job J_{ij} scheduled by AGDEQ on P processors of speed s , its total processor allocation $a(J_{ij})$ and overall satisfied processing time $t_B(J_{ij})$ satisfy¹ $a(J_{ij}) \leq \frac{1+\rho}{\delta} \cdot w(J_{ij})$ and $t_B(J_{ij}) \leq \frac{2}{s(1-\delta)} \cdot l(J_{ij}) + o(1)$.

¹As with the total response time analysis of AGDEQ in Chapter 4, the additive constant for the satisfied processing time is considered as $o(1)$, which will be absorbed by the performance of the optimal scheduler when the job is sufficiently large.

Now, according to definition, the satisfied processing time for job set $\mathcal{J}_i$ is given by $t_B(\mathcal{J}_i) \leq \max_{j=1 \dots n_i} t_B(J_{ij}) \leq \frac{2}{s(1-\delta)} \max_{j=1 \dots n_i} l(J_{ij}) + o(1) = \frac{2}{s(1-\delta)} \cdot l(\mathcal{J}_i) + o(1)$. When job set $\mathcal{J}_i$ is deprived, which means that at least one job within $\mathcal{J}_i$ is deprived, according to the DEQ allocator, all allocated processors to the job set have been distributed to its jobs. Hence, the deprived processor allocation for $\mathcal{J}_i$ satisfies $a_A(\mathcal{J}_i) \leq \sum_{j=1}^{n_i} a(J_{ij}) \leq \frac{1+\rho}{\delta} \sum_{j=1}^{n_i} w(J_{ij}) = \frac{1+\rho}{\delta} \cdot w(\mathcal{J}_i)$. For the entire job set collection $\mathcal{J}$ scheduled by EQUI $\circ$ AGDEQ, let $\gamma(\cdot)$ denote a permutation of the job sets sorted in non-increasing order of deprived processor allocation, and let $\pi(\cdot)$ denote a permutation of the job sets sorted in non-increasing order of work. The squashed deprived processor allocation is then given by $\hat{a}_A(\mathcal{J}) = \frac{1}{P} \sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\gamma(i)}) \leq \frac{1}{P} \sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\pi(i)}) \leq \frac{1}{P} \sum_{i=1}^m i \cdot \frac{1+\rho}{\delta} \cdot w(\mathcal{J}_{\pi(i)}) = \frac{1+\rho}{\delta} \cdot \hat{w}(\mathcal{J})$. The total satisfied processor time for $\mathcal{J}$ can be obtained by simply summing up the satisfied processor time over all job sets. $\square$

5.3 Analysis of EQUI $\circ$ AGDEQ for Batched Job sets

In this section, we analyze the set response time of the EQUI $\circ$ AGDEQ algorithm when all job sets are released in a single batch. The analysis adopts that of the total response time shown in Chapter 4 and relies on the *local competitiveness argument* [154], which bounds the performance of the online algorithm at any local time in terms of the two lower bounds presented in Section 5.1.

Recall that the squashed deprived processor allocation for $\mathcal{J}(\overleftarrow{t})$ is given by $\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P} \sum_{i=1}^m i \cdot a_A(\mathcal{J}_{\gamma(i)}(\overleftarrow{t}))$, where $\gamma(\cdot)$ denotes a permutation of the job sets in $\mathcal{J}(\overleftarrow{t})$ sorted in non-increasing order of deprived processor allocation. At any time t , let $m_t(z)$ denote the number of job sets in $\mathcal{J}(\overleftarrow{t})$ whose deprived processor allocation is at least z under EQUI $\circ$ AGDEQ, i.e., $m_t(z) = \sum_{i=1}^m [a_A(\mathcal{J}_i(\overleftarrow{t})) \geq z]$. Apparently, $m_t(z)$ is a staircase-like decreasing function of z , and Figure 5.1(a) shows an example of $m_t(z)$ at a particular time t . An alternative formulation for the squashed deprived processor allocation $\hat{a}_A(\mathcal{J}(\overleftarrow{t}))$ is given by

$$\hat{a}_A(\mathcal{J}(\overleftarrow{t})) = \frac{1}{P} \int_0^\infty \left(\sum_{i=1}^{m_t(z)} i \right) dz. \quad (5.3)$$

We give both EQUI $\circ$ AGDEQ and the optimal offline scheduler P processors of unit speed. The performance of EQUI $\circ$ AGDEQ can be shown in the following lemma and theorem.

Lemma 5.3 *Suppose that EQUI $\circ$ AGDEQ schedules a collection $\mathcal{J}$ of batched job sets on P processors of unit speed. Then the execution of the job sets satisfies the following*

Figure 5.1: (a) An example of $m_t(z)$ at a particular time t . (b) The changes of $m_t(z)$ in an infinitesimal interval Δt in the worst case.

running condition:

$$m_t \leq 2 \left(\frac{d\hat{a}_A(\mathcal{J}(t))}{dt} + m_t^B \right), \quad (5.4)$$

where $\frac{d\hat{a}_A(\mathcal{J}(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\hat{a}_A(\mathcal{J}(t+\Delta t)) - \hat{a}_A(\mathcal{J}(t))}{\Delta t}$ denotes the rate of change for the squashed deprived processor allocation in an infinitesimal interval Δt during which no job set completes.

Proof. According to the EQUI algorithm, each of the m_t active job sets gets P/m_t processors at time t . In the worst case, the m_t^A deprived job sets have the most deprived processor allocation so far among the m_t active job sets. As a result, in interval Δt during which no job set completes, each of the bottom m_t^A horizontal stripes from $m_t(z)$ grows by $\frac{P}{m_t} \Delta t$ (cf. Figure 1(b)). The rate of change for the squashed deprived processor allocation can then be bounded by

$$\begin{aligned} \frac{d\hat{a}_A(\mathcal{J}(t))}{dt} &= \frac{1}{P\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{m_t+\Delta t(z)} i \right) - \left(\sum_{i=1}^{m_t(z)} i \right) \right] dz \\ &\geq \frac{1}{P\Delta t} \cdot \frac{m_t^A(m_t^A + 1)}{2} \cdot \frac{P}{m_t} \Delta t \\ &\geq \frac{x_t^2}{2} m_t, \end{aligned} \quad (5.5)$$

where $x_t = m_t^A/m_t$, and obviously $0 \leq x_t \leq 1$. Since a job set is either satisfied or deprived, we have $m_t^B = (1-x_t)m_t$. It can be easily verified that the running condition holds for all values of x_t by substituting Inequality (5.5) into it. $\square$

Theorem 5.4 Suppose that EQUI $\circ$ AGDEQ schedules a collection $\mathcal{J}$ of m batched job sets on P processors of unit speed. Then the set response time $F_{\text{E}\circ\text{AG}}(\mathcal{J})$ for collection $\mathcal{J}$ satisfies

$$F_{\text{E}\circ\text{AG}}(\mathcal{J}) \leq 2 \left(\frac{1+\rho}{\delta} + \frac{2}{1-\delta} \right) \cdot F^*(\mathcal{J}) + o(1), \quad (5.6)$$

where $F^*(\mathcal{J})$ denotes the set response time for collection $\mathcal{J}$ under the optimal offline scheduler on unit-speed processors, and δ and ρ denote A-GREEDY's utilization parameter and responsiveness parameter.

Proof. As mentioned in Section 5.1, the set response time of any job set collection $\mathcal{J}$ scheduled by the EQUI $\circ$ AGDEQ algorithm can be expressed as $F_{\text{E}\circ\text{AG}}(\mathcal{J}) = \int_0^\infty m_t dt$. Similarly, the total satisfied processing time for $\mathcal{J}$ under EQUI $\circ$ AGDEQ is given by $t_B(\mathcal{J}) = \int_0^\infty m_t^B dt$. Integrating the running condition in Lemma 5.3 over time and applying the boundary condition, we have $F_{\text{E}\circ\text{AG}}(\mathcal{J}) \leq 2(\hat{a}_A(\mathcal{J}) + t_B(\mathcal{J}))$.

Substituting the bounds for the squashed deprived processor allocation $\hat{a}_A(\mathcal{J})$ and the total satisfied processing time $t_B(\mathcal{J})$ from Lemma 5.2 into the above inequality, we get $F_{\text{E}\circ\text{AG}}(\mathcal{J}) \leq 2\left(\frac{1+\rho}{\delta} \cdot \hat{w}(\mathcal{J}) + \frac{2}{1-\delta} \cdot l(\mathcal{J})\right) + o(1)$. Based on Lemma 5.1, both squashed work $\hat{w}(\mathcal{J})$ and total span $l(\mathcal{J})$ are lower bounds for the set response time of $\mathcal{J}$. The performance of EQUI $\circ$ AGDEQ thus satisfies $F_{\text{E}\circ\text{AG}}(\mathcal{J}) \leq 2\left(\frac{1+\rho}{\delta} + \frac{2}{1-\delta}\right) \cdot F^*(\mathcal{J}) + o(1)$. $\square$

5.4 Analysis of EQUI $\circ$ AGDEQ for Nonbatched Job sets

We now analyze the set response time of EQUI $\circ$ AGDEQ when the job sets are non-batched. Note that the squashed work is no longer a lower bound for the set response time in this scenario. The analysis here continues to adopt that of the total response time in Chapter 4 and uses the *amortized local competitiveness argument* [154].

We use the same potential function as given in Equation (4.9) for the total response time analysis of ACDEQ. Instead, we focus on the t -suffix $\mathcal{J}(\vec{t})$ of job set collection $\mathcal{J}$, and redefine $m_t(z)$ to be the number of job sets in $\mathcal{J}(\vec{t})$ whose deprived processor allocation is at least $\frac{1+\rho}{\delta} \cdot z$ at time t under EQUI $\circ$ AGDEQ, i.e., $m_t(z) = \sum_{i=1}^m [a_A(\mathcal{J}_i(\vec{t})) \geq \frac{1+\rho}{\delta} \cdot z]$. Moreover, let $m_t^*(z)$ denote the number of job sets in $\mathcal{J}^*(\vec{t})$ whose work is at least z under the optimal, i.e., $m_t^*(z) = \sum_{i=1}^m [w(\mathcal{J}_i^*(\vec{t})) \geq z]$. Hence, both $m_t(z)$ and $m_t^*(z)$ are staircase-like decreasing functions of z , and Figure 5.2(a) shows an example of $m_t(z)$ and $m_t^*(z)$ at a particular time t . For non-batched job sets, we give the EQUI $\circ$ AGDEQ algorithm P processors of speed s , where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$, while the optimal scheduler uses unit-speed processors. The potential function is given by

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{m_t(z)} i \right) - m_t(z)m_t^*(z) \right] dz, \quad (5.7)$$

where $\eta = \frac{2(1+\rho)}{\delta \epsilon P}$. We will prove the performance of EQUI $\circ$ AGDEQ in the following lemma and theorem for nonbatched job sets.

Lemma 5.5 *Suppose that EQUI $\circ$ AGDEQ schedules a collection $\mathcal{J}$ of job sets on P processors of speed s , where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$. Then given the potential function defined in Equation (5.7), the execution of the job set collection satisfies the following*

Figure 5.2: (a) An example of $m_t(z)$ and $m_t^*(z)$ at a particular time t . (b) The changes of $m_t(z)$ and $m_t^*(z)$ after a new job set with work w' and deprived processor allocation a' arrives. (c) The changes of $m_t(z)$ and $m_t^*(z)$ in an infinitesimal interval Δt in the worst case.

- *Boundary condition:* $\Phi(0) \leq 0$ and $\Phi(\infty) \geq 0$;
- *Arrival condition:* $\Phi(t)$ does not increase when a new job set arrives;
- *Completion condition:* $\Phi(t)$ does not increase when an existing job set completes under either ACDEQ or the optimal;
- *Running condition:*

$$m_t + \frac{d\Phi(t)}{dt} \leq \frac{2s}{\epsilon} (m_t^* + m_t^B), \quad (5.8)$$

where $\frac{d\Phi(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Phi(t+\Delta t) - \Phi(t)}{\Delta t}$ denotes the rate of change for the potential function in an infinitesimal interval Δt during which no job set arrives or completes under both EQUIOAGDEQ and the optimal.

Proof. As the potential function given in Equation (5.7) is identical to that given in Equation (4.9), the same argument in the proof of Lemma 4.5 suggests that the boundary, the arrival and the completion conditions still hold. In particular, when a new job set with work w' arrives at time t , let t^- and t^+ denote the instances right before and after the job set arrives. Hence, we have $m_{t^+}^*(z) = m_{t^-}^*(z) + 1$ for $z \leq w'$ and $m_{t^+}^*(z) = m_{t^-}^*(z)$ for $z > w'$. Similarly, $m_{t^+}(z) = m_{t^-}(z) + 1$ for $z \leq \frac{\delta}{1+\rho} \cdot a'_A$ and $m_{t^+}(z) = m_{t^-}(z)$ for $z > \frac{\delta}{1+\rho} \cdot a'_A$, where a'_A is the deprived processor allocation for the job set. Figure 5.2(b) illustrates the changes of $m_t(z)$ and $m_t^*(z)$ in this case. Note that $\frac{\delta}{1+\rho} \cdot a'_A \leq w'$ from the proof of Lemma 5.2. For convenience, let us define $\phi_t(z) = \left(\sum_{i=1}^{m_t(z)} i \right) - m_t(z)m_t^*(z)$. It is obvious that for $z > w'$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w'$, we consider two cases.

Case 1: for $z \leq \frac{\delta}{1+\rho} \cdot a'_A$, we have $\phi_{t^+}(z) - \phi_{t^-}(z) = \left(\sum_{i=1}^{m_{t^-}(z)+1} i \right) - (m_{t^-}(z) + 1)(m_{t^-}^*(z) + 1) - \left(\sum_{i=1}^{m_{t^-}(z)} i \right) + m_{t^-}(z)m_{t^-}^*(z) = -m_{t^-}^*(z) \leq 0$.

Case 2: for $\frac{\delta}{1+\rho} \cdot a'_A \leq z \leq w'$, we have $\phi_{t+}(z) - \phi_{t-}(z) = \left(\sum_{i=1}^{m_{t-}(z)} i\right) - m_{t-}(z)(m_{t-}^*(z) + 1) - \left(\sum_{i=1}^{m_{t-}(z)} i\right) + m_{t-}(z)m_{t-}^*(z) = -m_{t-}(z) \leq 0$.

Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t+}(z) dz \leq \eta \int_0^\infty \phi_{t-}(z) dz = \Phi(t^-)$, and the arrival condition holds.

We now prove the running condition. Since no job set arrives or completes during Δt , each of the m_t active job sets gets P/m_t processors according to EQUI. In the worst case, the m_t^A deprived job sets have the most remaining deprived processor allocation, while the optimal executes the job set with the least remaining work using all P processors. Hence, as Figure 5.2(c) shows, each of the bottom m_t^A horizontal stripes from $m_t(z)$ shrinks by $\frac{\delta s P}{(1+\rho)m_t} \Delta t$, and the top horizontal stripe from $m_t^*(z)$ shrinks by $P \Delta t$ in interval Δt . The rate of change for the potential function can then be bounded by

$$\begin{aligned}
\frac{d\Phi(t)}{dt} &= \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{m_{t+\Delta t}(z)} i \right) - m_{t+\Delta t}(z)m_{t+\Delta t}^*(z) - \left(\sum_{i=1}^{m_t(z)} i \right) + m_t(z)m_t^*(z) \right] dz \\
&\leq \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{m_{t+\Delta t}(z)} i \right) - \left(\sum_{i=1}^{m_t(z)} i \right) \right] dz \\
&\quad + \frac{\eta}{\Delta t} \int_0^\infty [m_t(z)(m_t^*(z) - m_{t+\Delta t}^*(z)) + m_t^*(z)(m_t(z) - m_{t+\Delta t}(z))] dz \\
&\leq \frac{2(1+\rho)}{\delta \epsilon P \Delta t} \left(-\frac{m_t^A(m_t^A + 1)}{2} \cdot \frac{\delta s P}{(1+\rho)m_t} \Delta t + m_t P \Delta t + m_t^* \frac{\delta s P m_t^A}{(1+\rho)m_t} \Delta t \right) \\
&\leq \frac{2(1+\rho)}{\delta \epsilon} \left(1 - \frac{\delta s x_t^2}{2(1+\rho)} \right) m_t + \frac{2s x_t}{\epsilon} m_t^*, \tag{5.9}
\end{aligned}$$

where $x_t = m_t^A/m_t$, and $0 \leq x_t \leq 1$. Since a job set is either satisfied or deprived, we have $m_t^B = (1 - x_t)m_t$. We can again verify that the running condition holds for all values of x_t by substituting Inequality (5.9) into Inequality (5.8). $\square$

Theorem 5.6 *Suppose that EQUI $\circ$ AGDEQ schedules a collection $\mathcal{J}$ of m job sets on P processors of speed s , where $s = \frac{2(1+\rho)}{\delta} + \epsilon$ for any $\epsilon > 0$. Then the set response time $F_{\text{E}\circ\text{AG}}(\mathcal{J})$ for collection $\mathcal{J}$ satisfies*

$$F_{\text{E}\circ\text{AG}}(\mathcal{J}) \leq \left(2 + \frac{4(1+\rho-\rho\delta)}{\delta(1-\delta)\epsilon} \right) F^*(\mathcal{J}) + o(1), \tag{5.10}$$

where $F^*(\mathcal{J})$ is the set response time for collection $\mathcal{J}$ under the optimal offline scheduler on unit-speed processors, and δ and ρ denote A-GREEDY's utilization parameter and responsiveness parameter.

Proof. As the set response time of EQUI $\circ$ AGDEQ is given by $F_{\text{E}\circ\text{AG}}(\mathcal{J}) = \int_0^\infty m_t dt$, and the set response time of the optimal is given by $F^*(\mathcal{J}) = \int_0^\infty m_t^* dt$, integrating the running condition in Lemma 5.5 over time, we have $F_{\text{E}\circ\text{AG}}(\mathcal{J}) + \Phi(\infty) - \Phi(0) +$

$\sum_{t \in T} (\Phi(t^-) - \Phi(t^+)) \leq \frac{2s}{\epsilon} (F^*(\mathcal{J}) + t_B(\mathcal{J}))$, where T denotes the set of times when a new job set arrives, and $t_B(\mathcal{J}) = \int_0^\infty m_t^B dt$ is the total satisfied processing time for $\mathcal{J}$ under EQUI $\circ$ AGDEQ. From Inequality (5.2), the total satisfied processing time for $\mathcal{J}$ is also given by $t_B(\mathcal{J}) \leq \frac{2}{s(1-\delta)} \cdot l(\mathcal{J}) + o(1)$, and applying the boundary condition and the arrival condition in Lemma 5.5, the set response time of $\mathcal{J}$ scheduled by EQUI $\circ$ AGDEQ satisfies $F_{\text{EoAG}}(\mathcal{J}) \leq \frac{2s}{\epsilon} \cdot F^*(\mathcal{J}) + \frac{4}{(1-\delta)\epsilon} \cdot l(\mathcal{J}) + o(1)$. Since the total span is a lower bound for the set response time of $\mathcal{J}$ on unit-speed processors, the theorem follows by substituting $F^*(\mathcal{J}) \geq l(\mathcal{J})$ into the above inequality and simplifying. $\square$

5.5 Analysis Framework for Set Response Time and Performance of EQUI $\circ$ ACDEQ

In this section, we present a generalized framework for the analysis of the EQUI $\circ$ Y family of algorithms, which uses EQUI to allocate processors among the sets, and uses any algorithm Y to schedule the jobs within each set. We then apply this analysis framework to the EQUI $\circ$ ACDEQ algorithm and show its performance.

Robert and Schabanel [161] showed that, for nonbatched job sets, EQUI $\circ$ Y is $O(1)$ -speed $O(1)$ -competitive with respect to the set response time for jobs with special parallelism structures, i.e., a sequential phase followed by a fully-parallel phase. This result holds independent of the choice of algorithm Y, with the only requirement that Y needs to distribute all the allocated processors of a set to the active jobs within the set at all time, i.e., Y never idles processors.

Here, we provide a generalized analysis framework of the EQUI $\circ$ Y family of algorithms for both batched and nonbatched job sets with any parallelism profile. Similar to the analysis of EQUI $\circ$ AGDEQ, the key to the analysis of EQUI $\circ$ Y is to study the efficiency of algorithm Y when scheduling an individual job, which is specifically reflected in two performance bounds. First of all, we need to bound the total processor allocation of a job in terms of the job's work. We then need to choose an appropriate property B , and bound the overall processor time of the job with property B in terms of the job's span. For any job J_{ij} scheduled by Y on speed- s processors, let $a(J_{ij}) = \int_0^\infty a(J_{ij}, t) s dt$ denote the job's total processor allocation and let $t_B(J_{ij}) = \int_0^\infty [J_{ij}(t) \in \mathcal{J}_i^B(t)] dt$ denote its overall processor time with property B , where $J_{ij}(t)$ denotes job J_{ij} at time t and $\mathcal{J}_i^B(t)$ denotes the set of active jobs in $\mathcal{J}_i$ that satisfy property B at t . The two bounds are given in terms of the work and span

of the job for any $s > 0$ in the following form:²

$$a(J_{ij}) \leq \gamma_1 \cdot w(J_{ij}), \quad (5.11)$$

$$t_B(J_{ij}) \leq \frac{\gamma_2}{s} \cdot l(J_{ij}). \quad (5.12)$$

For job set $\mathcal{J}_i$, we say that it satisfies property B at time t if all jobs in $\mathcal{J}_i$ satisfy property B at t . Otherwise, $\mathcal{J}_i$ is said to satisfy property A if it contains at least one active job that does not satisfy property B at t . In order for our analysis to hold, we require that at any time t when a job set $\mathcal{J}_i$ satisfies property A , algorithm Y never idles processors, that is, all the P/m_t processors allocated to $\mathcal{J}_i$ by $EQUI$ must be distributed to the active jobs in $\mathcal{J}_i$ by Y . This requirement is necessary to bound the squashed processor allocation, and eventually ensures the main claim.

Lemma 5.7 *Suppose that $EQUI \circ Y$ schedules a collection $\mathcal{J}$ of job sets. For any individual job $J_{ij} \in \mathcal{J}$, if its total processor allocation satisfies Inequality (5.11), and its overall processor time with a chosen property B satisfies Inequality (5.12), then $EQUI \circ Y$ achieves*

- $2(\gamma_1 + \gamma_2)$ -competitive with respect to the set response time for batched job sets;
- $(2\gamma_1 + \epsilon)$ -speed $\left(2 + \frac{2(2\gamma_1 + \gamma_2)}{\epsilon}\right)$ -competitive for nonbatched job sets, where $\epsilon > 0$,

provided that algorithm Y never idles the processors allocated to a set whenever at least one job in the set does not satisfy property B .

Proof. Based on the derivation of Lemma 5.2, the squashed processor allocation $\hat{a}_A(\mathcal{J})$ with property A and the total processor time $t_B(\mathcal{J})$ with property B for the entire job set collection $\mathcal{J}$ satisfy $\hat{a}_A(\mathcal{J}) \leq \gamma_1 \cdot \hat{w}(\mathcal{J})$ and $t_B(\mathcal{J}) \leq \gamma_2 \cdot l(\mathcal{J})$, where $\hat{w}(\mathcal{J})$ and $l(\mathcal{J})$ denote the squashed work and the total span of all job sets in $\mathcal{J}$, respectively. The claim can then be derived easily by following the analysis in Sections 5.3 and 5.4. $\square$

Note that as long as the scheduler at the job set level, which need not be $EQUI$, can guarantee fairness by allocating at least P/m_t processors at any time t to the sets that satisfy property A , Lemma 5.7 will continue to hold. Therefore, from the analysis in this chapter as well as the results of [160, 161], we can see that both fairness among the job sets and efficiency within each job set play important roles in contributing to the set response time performance of a scheduling algorithm.

We now apply the analysis framework presented above to the $EQUI \circ ACDEQ$ algorithm and show its performance. Again, we choose property B to be “satisfied” in this case, and for simplicity, we assume that the convergence rate v of A -CONTROL is set to 0. According to Theorems 3.3 and 3.4, the total processor allocation $a(J_{ij})$

²As with the total response time analysis of any two-level algorithms shown in Chapter 4, the additive constants for these bounds will be absorbed by the performance of the optimal scheduler for sufficiently large jobs.

for each individual job J_{ij} and the satisfied processing time $t_B(J_{ij})$ for J_{ij} scheduled by EQUI $\circ$ ACDEQ are bounded by

$$a(J_{ij}) \leq (C + 1) \cdot w(J_{ij}), \quad (5.13)$$

$$t_B(J_{ij}) \leq \frac{C + 1}{s} \cdot l(J_{ij}). \quad (5.14)$$

on processors of any speed s , where $s > 0$ and C denotes the maximum transition factor of all jobs in $\mathcal{J}$. The following theorem then gives the performance of EQUI $\circ$ ACDEQ.

Theorem 5.8 *Suppose that EQUI $\circ$ ACDEQ schedules a collection $\mathcal{J}$ of job sets. Then EQUI $\circ$ ACDEQ achieves*

- $4(C + 1)$ -competitive with respect to the set response time if job sets in $\mathcal{J}$ are batched;
- $(2C + 2 + \epsilon)$ -speed $(2 + \frac{6C+6}{\epsilon})$ -competitive for nonbatched job sets, where $\epsilon > 0$,

and C denotes the maximum transition factor of all jobs in $\mathcal{J}$.

Proof. The theorem follows by directly applying Lemma 5.7 with the bounds given in Inequalities (5.13) and (5.14). $\square$

We can see that the analysis framework for the set response time is similar to that for the total response time presented in Section 4.2 of Chapter 4. However, the requirements on the performances of an individual job is more stringent in the case of set response time than in total response time. In particular, the total processor allocation of a job, instead of the total processor allocation with a chosen property A , needs to be bounded in terms of the work of the job. It is such a stronger requirement that makes the theoretical bounds larger in set response time compared with total response time, as can be observed from the corresponding performances of the ACDEQ and AGDEQ based algorithms.

5.6 Discussions

From the analysis of the preceding two sections, we have that both EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ achieve competitive performances with respect to the set response time for batched job sets, since the various parameters in both algorithms can be considered as constants. In case that the job sets are nonbatched, EQUI $\circ$ ACDEQ and EQUI $\circ$ AGDEQ also achieve competitive performances using $O(1)$ times more resources than the optimal offline scheduler. The batched result improves the corresponding performance of the EQUI $\circ$ EQUI algorithm obtained in [160], which is $\Theta(\frac{\ln n}{\ln \ln n})$ -competitive, where n is the maximum number of jobs in any set. The nonbatched result extends the same performance of the EQUI $\circ$ Y family of algorithms obtained in

[161] for jobs with specific parallelism structure, i.e., a sequential phase followed by a fully parallel phase.

Compared with the results in [160, 161], the performances of our algorithms are based on the assumption that the jobs under consideration are either sufficiently large or have smooth parallelism variations. This may explain why our algorithms beat the lower bound shown in [160], which states that any non-clairvoyant scheduler is $\omega(1)$ -competitive with respect to the set response time for jobs with arbitrary sizes. Hence, in our case, those scenarios in which the jobs are too small to amortize the initial high cost of quantum-based schedulers are ignored. Moreover, the lower bound given in [160] is also based on the assumption that a non-clairvoyant algorithm is not allowed to access any information regarding the past execution of the jobs, such as those provided by the feedback mechanisms of AGDEQ and ACDEQ, hence it is only applicable to a stricter class of algorithms. Relaxing this assumption to allow learning from history, we can see that the improvement of our algorithms over EQUI◊EQUI is essentially due to the fact that the processors within each job set are utilized more efficiently than EQUI, which obviously allocates the processors and potentially incurs a large waste of resources.

Moreover, comparing the set response time bounds of the EQUI◊ACDEQ algorithm with that of EQUI◊AGDEQ, we can again see that EQUI◊ACDEQ tends to have better performances if the jobs have smoother parallelism variations, and hence smaller transition factors. However, the simulations conducted in Chapters 3 and 4 have shown that, in a wide range of parallelism variations and workloads, ACDEQ outperforms AGDEQ for a single set of jobs. In the following section, we will further confirm the advantage of the ACDEQ-based algorithm by comparing its set response time with that of EQUI◊AGDEQ and EQUI◊EQUI through simulations.

5.7 Empirical Evaluations

In this section, we conduct simulations to compare the set response time of three algorithms, namely, EQUI◊ACDEQ, EQUI◊AGDEQ, and EQUI◊EQUI. In addition, we evaluate the efficiency of the three algorithms in terms of their processor utilizations.

Following Downey’s model, the arrival rate of the jobs are set proportionally to the number of sets, and hence is tied to the load of the system. We group consecutively released jobs together to form job sets. In our simulation, the number of jobs in each job set is the same and varied from 5 to 100 at an increment of 5 each time. The number of sets also increases from 5 to 100. Moreover, to make sure that all jobs within a set are batched, we adjust the release time of each job in a job set to when the first job of the set is released. Because of such adjustment of the job release time, the gaps between the release of consecutive sets are also correspondingly increased. Hence, it turns out that the system load is still light when the number of sets reaches

Figure 5.3: The performance comparison of EQUI°ACDEQ and EQUI°AGDEQ with EQUI°EQUI in terms of set response time for (a) light loads and (b) heavy loads.

100. However, due to some limitations of our simulator, we are not able to further increase the number of sets. Thus, to simulate heavy loads, we also run batched job sets, and the number of sets is increased from 5 to 50 in this case.

We first focus on the set response time of the three scheduling algorithms, which are shown in Figure 5.3. From the simulation results, we can see that when the number of sets is fixed, the set response time of all algorithms closely resembles the performances of the corresponding schedulers for a single job set (see Figure 4.3), and hence is related to the number of jobs within each set. In general, EQUI°ACDEQ and EQUI°AGDEQ outperform EQUI°EQUI when there is a moderate number of jobs in each set. However, when each set contains relatively smaller or larger number of jobs, the performance of EQUI becomes better. Moreover, we can see that the set response time of the three scheduling algorithms are also significantly impacted by the number of sets, or the system loads. In particular, the set response time of both

(a)

(b)

Figure 5.4: The performance comparison of $\text{EQUI} \circ \text{ACDEQ}$ and $\text{EQUI} \circ \text{AGDEQ}$ with $\text{EQUI} \circ \text{EQUI}$ in terms of processor utilization for (a) light loads and (b) heavy loads.

$\text{EQUI} \circ \text{ACDEQ}$ and $\text{EQUI} \circ \text{AGDEQ}$ are much better than $\text{EQUI} \circ \text{EQUI}$ under light loads while $\text{EQUI} \circ \text{EQUI}$ performs comparably well when the loads become heavy. This is again similar to the performances of the three schedulers for a single job set. In addition, we can see from Figure 5.3 that the performance of $\text{EQUI} \circ \text{ACDEQ}$ is always better than that of $\text{EQUI} \circ \text{AGDEQ}$, which is due to the more effective and stable desire calculation strategy of the A-CONTROL scheduler.

We now turn to the processor utilizations of the three algorithms, which are shown in Figure 5.4. The results show that the utilizations of the two-level schedulers are always better than that of $\text{EQUI} \circ \text{EQUI}$ under all system loads. Especially when there are very few jobs in each set, $\text{EQUI} \circ \text{EQUI}$ has the worst processor utilization, since it is blind to the jobs' parallelism variations by equally allocating all processors, which eventually leads to large waste of resources. The simulation results also reveal that

EQUI◦EQUI actually achieves better set response time in this case at the cost of poor system utilization. On the other hand, with increases in job number and set number, the utilization advantage of the two-level schedulers gradually becomes smaller. This is because nearly all processors are well utilized by all three schedulers under heavy system loads. In addition, Figure 5.4 also shows that the utilization of EQUI◦ACDEQ is slightly better than that of EQUI◦AGDEQ, which is again because of the more effective parallelism feedback strategy of A-CONTROL. The simulation results confirm that the two-level scheduling algorithms that take advantage of parallelism correlations of the jobs indeed achieve better overall system efficiency than EQUI◦EQUI.

Chapter 6

Total Response Time plus Energy on Variable Speed Processors

In the preceding three chapters, we focused on the scenario where the speeds of all processors are identical and fixed. Hence, performance was the only concern. In this chapter and the following one, we present scheduling algorithms on processors whose speeds can be dynamically scaled. In this case, both performance and energy consumption become our concerns and they are apparently conflicting objectives that are closely related to the speeds of the processors. Our goal is to design non-clairvoyant scheduling algorithms that minimize a combination of performance and energy.

We adopt the power function proposed in [189] and focus on multiprocessors with per-processor speed scaling capability [94, 112], that is, the speed of each processor can be individually scaled and each processor consumes power s^α when running at speed s , where $\alpha > 1$. The objective is to minimize total response time plus energy for a set of jobs. Compared with scheduling on fixed speed processors, where we need only design a *processor allocation policy* that decides at any time the number of processors allocated to each job, scheduling on variable speed processors needs to have an additional *speed scaling policy* that decides the speed of each allocated processor. Since the parallel jobs we consider can have time-varying parallelism, the challenge for a non-clairvoyant scheduling algorithm is to ensure that it does not incur large amount of energy waste when the jobs have relatively low parallelism, or cause severe execution delays when the parallelism of the jobs is high.

We first show that any non-clairvoyant algorithm, which employs uniform speed scaling, performs poorly, or more specifically is $\Omega(P^{(\alpha-1)/\alpha^2})$ -competitive, with respect to the total response time plus energy, where P is the total number of processors. We then present two adaptive scheduling algorithms N-ACCEQ and N-AGCEQ, which combine variants of the ACDEQ and AGDEQ schedulers with a non-uniform speed scaling policy. We show that both N-ACCEQ and N-AGCEQ achieve $O(\ln P)$ -competitive for nonbatched jobs and $O(\ln^{1/\alpha} P)$ -competitive for batched jobs, which are the best asymptotic bounds to date. Moreover, we also provide a matching lower bound for

batched jobs, hence suggesting that both algorithms are asymptotically optimal in this setting. Again, the upper bounds are obtained based on the technique we developed when analyzing the total response time in Chapter 4. In this chapter, however, we first present the general analysis framework before deriving the specific bounds for N-ACCEQ and N-AGCEQ. The content of this chapter was originally presented in [174, 176].

6.1 A Lower Bound on Non-clairvoyant Uniform Speed Scaling

In this section, we present a lower bound on the competitive ratio of any non-clairvoyant algorithm that employs uniform speed scaling. Recall that a scheduling algorithm employs uniform speed scaling if it restricts all processors allocated to a job to run at the same speed at any given time; otherwise, if the algorithm allows the allocated processors to a job to run at different speeds, we say that it employs non-uniform speed scaling. The following theorem gives the lower bound for any non-clairvoyant uniform speed scaling algorithm.

Theorem 6.1 *Any non-clairvoyant scheduling algorithm that employs uniform speed scaling is $\Omega(P^{(\alpha-1)/\alpha^2})$ -competitive with respect to the total response time plus energy, where P is the total number of processors.*

Proof. The proof is based on a simple construction with a single job having constant parallelism $h \leq P$ and work $w > 0$. Without loss of generality, we can assume that any non-clairvoyant uniform speed scaling algorithm A allocates a fixed number a of processors with speed s to the job throughout its execution. The adversary, upon knowing the allocation a , can set the parallelism h of the job in such a way that it forces algorithm A to have a large competitive ratio.

The optimal algorithm on the other hand always allocates h processors to the job, each with speed $\left(\frac{1}{(\alpha-1)h}\right)^{1/\alpha}$. Thus, it has total response time plus energy $G^* = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{h^{1-1/\alpha}}$. We consider two cases depending on the allocation of A.

Case 1: Suppose that $a \geq P^{1-1/\alpha}$. Then setting $h = 1$ gives the total response time plus energy $G_A = \frac{w}{s} + \frac{w}{s}as^\alpha \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot wa^{1/\alpha}$. The competitive ratio in this case is given by $G_A/G^* \geq a^{1/\alpha} \geq P^{(\alpha-1)/\alpha^2}$.

Case 2: Suppose that $a < P^{1-1/\alpha}$. Then setting $h = P$ gives the total response time plus energy $G_A = \frac{w}{as} + \frac{w}{as}as^\alpha \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{a^{1-1/\alpha}}$. The competitive ratio in this case is given by $G_A/G^* \geq (h/a)^{1-1/\alpha} > P^{(\alpha-1)/\alpha^2}$. $\square$

Theorem 6.1 shows that any uniform speed scaling algorithm performs poorly in the absence of clairvoyance. Intuitively, the poor performance is because a non-clairvoyant algorithm may in the worst case allocate a “wrong” number of processors

to a job compared to its parallelism, thus either incurring excessive waste of energy or causing severe execution delay for the job. Hence, in the rest of this chapter, we will only consider non-uniform speed scaling algorithms.

However, in non-uniform speed scaling, setting processors to run at different speeds can have complicated effects on a parallel job's execution, subject to the underlying dependencies among the tasks (or threads) of the job as well as the task scheduling strategy. In this thesis, we assume that the *maximum utilization policy* [100, 24] is used in task scheduling as described in Section 1.4, which always utilizes faster processors before slower ones. Under this policy, the execution rate of job J_i at any time t is given by the sum of the speeds from the $\min\{a_i(t), h_i^k\}$ fastest processors, where $a_i(t)$ is the number of processors allocated to J_i at time t and h_i^k is the instantaneous parallelism of the job at t . Such rate can be achieved by executing the tasks of the job in an ideal round-robin manner if the tasks are sufficiently long and independent of each other, as in many data-parallel programs [96].

Intuitively, maximum utilization policy brings positive impacts on the energy waste and the execution rate of a job when the processor allocation is more than the job's instantaneous parallelism. Specifically, under this policy, the job is able to utilize the faster processors from the set of all allocated ones while leaving only the slower processors idle. Hence, the execution rate for the job, i.e., aggregate speed from the utilized processors, is now increased while the energy waste, i.e., aggregate power from the idle processors, is reduced. However, when the processor allocation is not more than the instantaneous parallelism of the job, non-uniform speed scaling performs worse than its uniform counterpart, since all allocated processors are utilized in this case and generally less energy will be consumed with uniform speed [189]. Nevertheless, as we will see in the rest of this chapter, the benefit of non-uniform speed scaling outweighs its deficiency under the maximum utilization policy, and therefore it will lead to much improved performance compared to the lower bound on uniform speed scaling.

6.2 Analysis Framework for Total Response Time plus Energy

We now present a general analysis framework for total response time plus energy of any online algorithm N-X, which uses a specific non-uniform speed scaling policy and any processor allocation policy X. We give the analysis of N-X for batched jobs and nonbatched jobs, respectively.

6.2.1 Preliminaries and the N-X algorithm

As with the analysis in the preceding two chapters, we need to find two lower bounds on the total response time plus energy of any job set $\mathcal{J}$. Let $\pi(\cdot)$ denote a permutation

of the jobs in $\mathcal{J}$ sorted in non-increasing work order, i.e., $w(J_{\pi(1)}) \geq w(J_{\pi(2)}) \geq \dots \geq w(J_{\pi(n)})$. The following lemma shows the two lower bounds.

Lemma 6.2 *To schedule any set $\mathcal{J}$ of n parallel jobs on P processors, the total response time plus energy $G^*(\mathcal{J})$ of the job set scheduled by an optimal algorithm is lower-bounded by the following two equations*

$$\begin{aligned} G_1^*(\mathcal{J}) &= \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^n \sum_{k=1}^{k_i} \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}, \\ G_2^*(\mathcal{J}) &= \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot w(J_{\pi(i)}), \end{aligned}$$

and the second lower bound only applies to batched jobs.

Proof. To derive the first lower bound, consider any phase J_i^k of a particular job J_i . The optimal offline scheduler will only perform better in terms of total response time plus energy if there is an unlimited number of processors at its disposal. In this case, it will allocate a processors of the same speed, say s , to the phase throughout its execution, since by the convexity of the power function, if different speeds are used, then averaging the speeds will result in the same execution rate but less energy consumed [189]. Moreover, we also have $a \leq h_i^k$, since allocating more processors to a phase than its parallelism will only incur more energy without improving its response time. The response time plus energy introduced by the execution of J_i^k is then given by $\frac{w_i^k}{as} + \frac{w_i^k}{as} \cdot as^\alpha = w_i^k \left(\frac{1}{as} + s^{\alpha-1} \right) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{a^{1-1/\alpha}} \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$. Extending this property over all phases and all jobs gives the lower bound.

For the second lower bound, the optimal offline scheduler will do no worse in terms of total response time plus energy if each job J_i in the job set is replaced by a simpler job that contains a single fully-parallel phase with work w_i , since the original optimal schedule is also a valid schedule for the new job set. Now, because the jobs in the new job set are fully-parallel and batch released, it is well-known that the optimal offline scheduler will execute them using the SJF policy, since otherwise the total response time can be reduced by swapping the jobs without affecting the energy consumption. Moreover, for each job J_i , the optimal offline scheduler will allocate all P processors of the same speed, say s_i , throughout its execution according to the same argument as in the proof of the first lower bound. The response time plus energy introduced by the execution of J_i is given by $\frac{w(J_i)}{Ps} \cdot i + \frac{w(J_i)}{Ps} \cdot Ps^\alpha = w(J_i) \left(\frac{i}{Ps} + s^{\alpha-1} \right) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{i^{1-1/\alpha} \cdot w(J_i)}{P^{1-1/\alpha}}$. Summing the inequality over all jobs gives the lower bound. $\square$

We now present the N-X algorithm, in which X is a processor allocation policy that at any time t decides the processor allocation $a_i(t)$ for each job J_i in the set $\mathcal{J}(t)$ of active jobs. The speeds of the allocated processors are then set in a non-uniform manner based on a scaled version of harmonic series as shown in Algorithm 5. For convenience, we assume that the number $a_i(t)$ of processors allocated to each job J_i

by X is always an integer, otherwise by rounding it to $\lfloor a_i(t) \rfloor$, the bounds derived will increase by at most a constant factor.¹ Our analysis framework applies to any such algorithm N-X with the requirement that X never allocates more than the equal share P/n_t of processors to a job at any time t .

Algorithm 5 N-X

Require: set $\mathcal{J}(t)$ of active jobs at time t and total number P of processors

Ensure: processor allocation $a_i(t)$ and speed $s_{ij}(t)$ of each allocated processor for each active job J_i at time t , where $1 \leq j \leq a_i(t)$

1: decide $a_i(t)$ using algorithm X

2: $s_{ij}(t) = \left(\frac{1}{(\alpha-1)H_P \cdot j} \right)^{1/\alpha}$, where $H_P = 1 + \frac{1}{2} + \dots + \frac{1}{P}$ is the P -th harmonic number

Since N-X is non-clairvoyant, it does not know a priori the parallelism of the jobs when allocating processors. Hence, if the speeds of the processors are not properly assigned, it may either waste a large amount of energy or cause severe execution slowdown for the jobs. However, we will show in the rest of this chapter that by assigning the speeds of the allocated processors in this manner, N-X is able to strike a good balance between the energy waste and the execution rate. First of all, we bound the processing power and the power consumption of N-X for any job at time t in the following lemma. These bounds will be useful to our later analysis.

Lemma 6.3 *Suppose that N-X schedules a set $\mathcal{J}$ of jobs. Then at any time t , when any job $J_i \in \mathcal{J}$ is allocated $a_i(t)$ processors, the processing power for the job satisfies $\left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{2^{1/\alpha}} \leq \sum_{j=1}^{a_i(t)} s_{ij}(t) \leq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{1-1/\alpha}$, and the power consumption for the job satisfies $u_i(t) \leq \frac{1}{\alpha-1}$.*

Proof. According to the N-X algorithm, the processing power for any job J_i at time t is given by $\sum_{j=1}^{a_i(t)} s_{ij}(t) = \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \sum_{j=1}^{a_i(t)} \frac{1}{j^{1/\alpha}}$. We can approximate summation with integration as follows: $\sum_{j=1}^{a_i(t)} s_{ij}(t) \leq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \int_0^{a_i(t)} \frac{1}{j^{1/\alpha}} dj = \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{1-1/\alpha}$, and $\sum_{j=1}^{a_i(t)} s_{ij}(t) \geq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \int_1^{a_i(t)+1} \frac{1}{j^{1/\alpha}} dj = \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{(a_i(t)+1)^{1-1/\alpha}-1}{1-1/\alpha} \geq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha} a_i(t)^{1-1/\alpha} \geq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{2^{1/\alpha}}$. The second last inequality is because $\frac{(a+1)^{1-1/\alpha}-1}{a^{1-1/\alpha}}$ is an increasing function of a for all $a > 0$.

The power consumption for any job J_i at time t satisfies $u_i(t) = \sum_{j=1}^{a_i(t)} s_{ij}(t)^\alpha = \frac{1}{(\alpha-1)H_P} \sum_{j=1}^{a_i(t)} \frac{1}{j} = \frac{H_{a_i(t)}}{(\alpha-1)H_P} \leq \frac{1}{\alpha-1}$, where $H_{a_i(t)}$ is $a_i(t)$ -th harmonic number and apparently we have $H_{a_i(t)} \leq H_P$ since $a_i(t) \leq P$ at any time. $\square$

We can see from Lemma 6.3 that at any time t the overall power consumption u_t of all active jobs under N-X is given by $u_t \leq \frac{n_t}{\alpha-1}$, which is proportional to the

¹In case that the number of jobs at any time is more than the number of processors, we can set the speed of all processors to $\left(\frac{n_t}{(\alpha-1)H_P \cdot P} \right)^{1/\alpha}$ and equally share the processors among all active jobs, i.e., allocate a fraction $a_i(t) = P/n_t$ of processors to each job J_i . It can be verified that our subsequent analysis will not be affected.

response time contribution of the job set at time t . This strategy provides an intuitive solution to optimize the total response time plus energy for any online scheduling algorithm and it has been a heuristic commonly accepted in the speed scaling literature [7, 21, 117, 48, 49, 18] and in system designs [22].

To simplify our analysis, we define a *scaling factor* $\beta_i(t)$ for each job J_i at time t such that it satisfies $\beta_i(t) \cdot u_i(t) = \frac{1}{\alpha-1}$. In addition, we need to identify two properties A and B , and let $\mathcal{J}_A(t)$ and $\mathcal{J}_B(t)$ denote the sets of active jobs in $\mathcal{J}$ that satisfy properties A and B respectively at time t . Then for each job J_i , let $g_A(J_i) = \int_0^\infty \left(\sum_{j=1}^{a_i(t)} s_{ij}(t) \right) \cdot [J_i(t) \in \mathcal{J}_A(t)] dt$ denote the job's *processing power* when it satisfies property A and let $g_B(J_i) = \int_0^\infty (1 + \beta_i(t) \cdot u_i(t)) \cdot [J_i(t) \in \mathcal{J}_B(t)] dt = \frac{\alpha}{\alpha-1} \int_0^\infty [J_i(t) \in \mathcal{J}_B(t)] dt$ denote the job's *response time plus scaled energy* when it satisfies property B . For convenience, let $n_t^A = |\mathcal{J}_A(t)|$ and $n_t^B = |\mathcal{J}_B(t)|$. Our analysis requires that properties A and B are chosen such that they cover the whole set of active jobs at any time t , i.e., $\mathcal{J}_A(t) \cup \mathcal{J}_B(t) = \mathcal{J}(t)$ or $n_t^A + n_t^B \geq n_t$, where $\mathcal{J}(t)$ denotes the set of all active jobs at t . In addition, we require that each active job that satisfies property A at time t must get exactly the equal share P/n_t of processors. The analysis framework relies on finding γ_1 and γ_2 such that the following inequalities hold:

$$g_A(J_i) \leq \gamma_1 \cdot w(J_i), \quad (6.1)$$

$$g_B(J_i) \leq \gamma_2 \cdot g_1^*(J_i), \quad (6.2)$$

where $g_1^*(J_i) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{k=1}^{k_i} \frac{w_i^k}{(h_i^k)^{1-1/\alpha}}$. Now, we define the concepts of *squashed processing power* $G_A(\mathcal{J})$ with property A and *total response time plus scaled energy* $G_B(\mathcal{J})$ with property B for a set of jobs as follows:

$$G_A(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot g_A(J_{\gamma(i)}),$$

$$G_B(\mathcal{J}) = \sum_{i=1}^n g_B(J_i),$$

where $\gamma(\cdot)$ denotes a permutation of the jobs sorted in non-increasing order of processing power with property A , i.e., $g_A(\mathcal{J}_{\gamma(1)}) \geq g_A(\mathcal{J}_{\gamma(2)}) \geq \dots \geq g_A(\mathcal{J}_{\gamma(n)})$. In the following lemma, we derive the upper bounds for $G_A(\mathcal{J})$ and $G_B(\mathcal{J})$ respectively in terms of the two lower bounds given in Lemma 6.2. These bounds play essential roles in our analysis framework.

Lemma 6.4 *Suppose that N-X schedules a set $\mathcal{J}$ of jobs. Then the squashed processing power $G_A(\mathcal{J})$ with property A and the total response time plus scaled energy*

$$\begin{aligned} G_A(\mathcal{J}) &\leq \gamma_1 \cdot G_2^*(\mathcal{J}), \\ G_B(\mathcal{J}) &\leq \gamma_2 \cdot G_1^*(\mathcal{J}), \end{aligned}$$

where $G_1^*(\mathcal{J})$ and $G_2^*(\mathcal{J})$ are the two lower bounds on the total response time plus energy of $\mathcal{J}$ given in Lemma 6.2.

Proof. For the job set $\mathcal{J}$ scheduled by N-X, let $\gamma(\cdot)$ denote a permutation of the jobs sorted in non-increasing order of processing power with property A , and let $\pi(\cdot)$ denote a permutation of the jobs sorted in non-increasing order of work. Then according to the definitions of $G_A(\mathcal{J})$, $G_2^*(\mathcal{J})$ and Inequality (6.1), we have $G_A(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot g_A(J_{\gamma(i)}) \leq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot g_A(J_{\pi(i)}) \leq \gamma_1 \cdot \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot w(J_{\pi(i)}) = \gamma_1 \cdot G_1^*(\mathcal{J})$.

For the second bound, according to the definitions of $G_B(\mathcal{J})$, $G_1^*(\mathcal{J})$ and Inequality (6.2), we have $G_B(\mathcal{J}) = \sum_{i=1}^n g_B(J_i) \leq \gamma_2 \cdot \sum_{i=1}^n g_1^*(J_i) = \gamma_2 \cdot G_1^*(\mathcal{J})$. $\square$

Finally, recall the notions of t -prefix and t -suffix for a job set $\mathcal{J}$. At time t , let $\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{G_{\text{NX}}(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_{\text{NX}}(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ denote the rate of change for the total response time plus energy scheduled by N-X, and let $\frac{dG^*(\mathcal{J}^*(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{G^*(\mathcal{J}^*(\overleftarrow{t+\Delta t})) - G^*(\mathcal{J}^*(\overleftarrow{t}))}{\Delta t}$ denote the rate of change for the total response time plus energy scheduled by an optimal offline scheduler. Apparently, we have $\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} = n_t + u_t \leq \frac{\alpha}{\alpha-1} n_t$, and $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + u_t^*$, where n_t^* and u_t^* denote the number of active jobs and the power consumption respectively under the optimal at time t . Moreover, we let $\frac{dG_A(\mathcal{J}(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{G_A(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_A(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ and $\frac{dG_B(\mathcal{J}(t))}{dt} = \lim_{\Delta t \rightarrow 0} \frac{G_B(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_B(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ denote respectively the rates of change for the total processing power with property A and the total response time plus scaled energy with property B in terms of the execution of the job set under N-X at time t . Note that Δt represents an infinitesimally small interval of time during which no job arrives, completes or experiences any phase transition under both N-X and the optimal.

6.2.2 Analysis of N-X for Batched Jobs

In this section, we analyze the total response time plus energy of the N-X algorithm when all jobs are batch released. The analysis again uses the *local competitiveness argument* [154] by bounding the performance of an online algorithm at any time in terms of the optimal offline scheduler.

For any job set $\mathcal{J}$, we focus on its t -prefix $\mathcal{J}(\overleftarrow{t})$, which according to definition, always contains n jobs for any $t > 0$, since all jobs are batch released in this scenario. Recall that the squashed processing power with property A for $\mathcal{J}(\overleftarrow{t})$ is given by $G_A(\mathcal{J}(\overleftarrow{t})) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n i^{1-1/\alpha} \cdot g_A(J_{\gamma(i)}(\overleftarrow{t}))$, where $\gamma(\cdot)$ denotes a permutation of the jobs in $\mathcal{J}(\overleftarrow{t})$ sorted in non-increasing order of processing power with

Figure 6.1: (a) An example of $n_t(z)$ at a particular time t . (b) The changes of $n_t(z)$ in an infinitesimal interval Δt in the worst case.

property A . At any time t , let $n_t(z)$ denote the number of jobs in $\mathcal{J}(\overleftarrow{t})$ whose processing power with property A is at least z under N-X, i.e., $n_t(z) = \sum_{i=1}^n [g_A(J_i(\overleftarrow{t})) \geq z]$, which is a staircase-like decreasing function of z as shown in Figure 6.1(a). It is not difficult to see that an alternative formulation for the squashed processing power with property A is given by

$$G_A(\mathcal{J}(\overleftarrow{t})) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \int_0^\infty \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) dz.$$

The following lemma gives the local performance of the N-X algorithm.

Lemma 6.5 *Suppose that N-X schedules a set $\mathcal{J}$ of batched jobs on P processors. Then the execution of the jobs at any time t satisfies the following running condition:*

$$\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} \leq 2^{1+1/\alpha} H_P^{1/\alpha} \cdot \frac{dG_A(\mathcal{J}(t))}{dt} + 2 \cdot \frac{dG_B(\mathcal{J}(t))}{dt}. \quad (6.3)$$

Proof. We know that $\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} \leq \frac{\alpha}{\alpha-1} n_t$, and according to definition, we have $\frac{dG_B(\mathcal{J}(t))}{dt} = \frac{\alpha}{\alpha-1} n_t^B \geq \frac{\alpha}{\alpha-1} (1-x_t) n_t$, where $x_t = n_t^A/n_t$.

We assumed that algorithm X at any time t allocates exactly $a_i(t) = P/n_t$ processors to each active job J_i that satisfies property A . In the worst case, the n_t^A such jobs have the most processing power with property A so far among the n_t active jobs. Therefore, as shown in Figure 6.1(b), in interval Δt during which no job set completes, each of the bottom n_t^A horizontal stripes from $n_t(z)$ grows by $(\sum_{j=1}^{a_i(t)} s_{ij}(t)) \Delta t \geq \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{2^{1/\alpha}} \Delta t$, where the inequality is due to Lemma 6.3. Also, we can get that $\sum_{i=1}^{n_t^A} i^{1-1/\alpha} \geq \int_0^{n_t^A} i^{1-1/\alpha} di = \frac{(n_t^A)^{2-1/\alpha}}{2-1/\alpha} \geq \frac{x_t^2 n_t^{2-1/\alpha}}{2}$. The rate of change for the

squashed processing power with property A can then be bounded by

$$\begin{aligned}
\frac{dG_A(\mathcal{J}(t))}{dt} &= \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i^{1-1/\alpha} \right) - \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) \right] dz \\
&\geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha} \Delta t} \cdot \sum_{i=1}^{n_t^A} i^{1-1/\alpha} \cdot \left(\sum_{j=1}^{a_i(t)} s_{ij}(t) \right) \Delta t \\
&\geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{1}{P^{1-1/\alpha}} \cdot \frac{x_t^2 n_t^{2-1/\alpha}}{2} \cdot \left(\frac{1}{(\alpha-1)H_P} \right)^{1/\alpha} \frac{a_i(t)^{1-1/\alpha}}{2^{1/\alpha}} \\
&= \frac{\alpha}{\alpha-1} \cdot \frac{x_t^2}{2^{1+1/\alpha} H_P^{1/\alpha}} n_t.
\end{aligned}$$

Now, substituting the bounds of $\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt}$, $\frac{dG_B(\mathcal{J}(t))}{dt}$ and $\frac{dG_A(\mathcal{J}(t))}{dt}$ into Inequality (6.3), we can verify that the running condition holds for all values of x_t . $\square$

Lemma 4.3 basically gives the local performance of the N-X algorithm in terms of the squashed processing power with property A and the total response time plus scaled energy with property B . We can combine it with the result of Lemma 6.4 for the overall performance of N-X in batched scenario.

Theorem 6.6 *Suppose that N-X schedules a set $\mathcal{J}$ of batched jobs on P processors. Then the total response time plus energy $G_{\text{NX}}(\mathcal{J})$ of the job set satisfies*

$$G_{\text{NX}}(\mathcal{J}) \leq \left(2^{1+1/\alpha} H_P^{1/\alpha} \cdot \gamma_1 + 2 \cdot \gamma_2 \right) \cdot G^*(\mathcal{J}), \quad (6.4)$$

where $G^*(\mathcal{J})$ is the total response time plus energy of $\mathcal{J}$ under the optimal offline scheduler, and γ_1 and γ_2 are the multipliers given in Inequalities (6.1) and (6.2).

Proof. Integrating the running condition in Lemma 6.5 over time, we have $G_{\text{NX}}(\mathcal{J}) \leq \left(2^{1+1/\alpha} H_P^{1/\alpha} \cdot G_A(\mathcal{J}) + 2 \cdot G_B(\mathcal{J}) \right)$. We can then prove the theorem by substituting the properties of $G_A(\mathcal{J})$ and $G_B(\mathcal{J})$ given in Lemma 6.4 into the above inequality. $\square$

6.2.3 Analysis of N-X for Nonbatched Jobs

We now analyze the total response time plus energy of the N-X algorithm when the jobs are nonbatched. The analysis uses the *amortized local competitiveness argument* [154], which bounds the amortized performance of an online algorithm at any time in terms of the optimal offline scheduler through a potential function.

We adopt the potential function as given in Equation (4.9) for the total response time analysis of ACDEQ, but tailor it to suit the analysis for total response time plus energy. Specifically, given the t -suffix $\mathcal{J}(\vec{t})$ of job set $\mathcal{J}$, we redefine $n_t(z)$ to be the number of jobs in $\mathcal{J}(\vec{t})$ whose processing power with property A is at least $\gamma_1 \cdot z$ at time t under N-X, i.e., $n_t(z) = \sum_{i=1}^n [g_A(J_i(\vec{t})) \geq \gamma_1 \cdot z]$. Moreover, let $n_t^*(z)$

Figure 6.2: (a) An example of $n_t(z)$ and $n_t^*(z)$ at a particular time t . (b) The changes of $n_t(z)$ and $n_t^*(z)$ after a new job with work w' and processing power g'_A with property A arrives. (c) The changes of $n_t(z)$ and $n_t^*(z)$ in an infinitesimal interval Δt in the worst case.

denote the number of jobs in $\mathcal{J}^*(\vec{t})$ whose work is at least z under the optimal, i.e., $n_t^*(z) = \sum_{i=1}^n [w(J_i^*(\vec{t})) \geq z]$. Hence, both $n_t(z)$ and $n_t^*(z)$ are staircase-like decreasing functions of z as shown in Figure 6.2(a). The potential function is then defined to be

$$\Phi(t) = \eta \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) - n_t(z)^{1-1/\alpha} n_t^*(z) \right] dz, \quad (6.5)$$

where $\eta = \eta' \frac{H_P^{1/\alpha}}{P^{1-1/\alpha}}$ and η' is a constant to be specified later. In addition, we define $\frac{d\Phi(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{\Phi(t+\Delta t) - \Phi(t)}{\Delta t}$ to be the rate of change for the potential function at time t . The following lemma proves the amortized local performance of N-X.

Lemma 6.7 *Suppose that N-X schedules a set $\mathcal{J}$ of job sets on P processors. Then given the potential function defined in Equation (6.5), the execution of the job set satisfies the following*

- *Boundary condition:* $\Phi(0) \leq 0$ and $\Phi(\infty) \geq 0$;
- *Arrival condition:* $\Phi(t)$ does not increase when a new job arrives;
- *Completion condition:* $\Phi(t)$ does not increase when an existing job completes under either N-X or the optimal;
- *Running condition:*

$$\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \leq c_1 \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + 2\alpha \cdot \frac{dG_B(\mathcal{J}(t))}{dt}, \quad (6.6)$$

where $c_1 = \max\{\frac{4\alpha^3}{(\alpha-1)^2}, (4\gamma_1)^\alpha \alpha H_P\}$ and γ_1 is the multiplier given in Inequality (6.1).

Proof. We will examine each of the conditions in the following.

- *Boundary condition*: at time 0, no jobs exist in the system. The terms $n_t(z)$ and $n_t^*(z)$ are both 0 for all z . Therefore, we have $\Phi(0) = 0$. At time ∞ , all jobs have completed execution, i.e., $\mathcal{J}(\vec{t}) = \emptyset$ and $\mathcal{J}^*(\vec{t}) = \emptyset$, so again we have $\Phi(\infty) = 0$. Hence, the boundary condition holds.

- *Arrival condition*: suppose that a new job with work w' arrives at time t . Let t^- and t^+ denote the instances right before and after the job arrives. Hence, we have $n_{t^+}^*(z) = n_{t^-}^*(z) + 1$ for $z \leq w'$ and $n_{t^+}^*(z) = n_{t^-}^*(z)$ for $z > w'$, and similarly, $n_{t^+}(z) = n_{t^-}(z) + 1$ for $z \leq g'_A/\gamma_1$ and $n_{t^+}(z) = n_{t^-}(z)$ for $z > g'_A/\gamma_1$, where g'_A is the job's processing power with property A. Figure 6.2(b) illustrates the changes of $n_t(z)$ and $n_t^*(z)$ in this case. Note that $g'_A/\gamma_1 \leq w'$ from Inequality (6.1). For convenience, let us define $\phi_t(z) = \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha}\right) - n_t(z)^{1-1/\alpha} n_t^*(z)$. Thus, it is obvious that for $z > w'$, we have $\phi_{t^+}(z) = \phi_{t^-}(z)$. For $z \leq w'$, we consider two cases.

Case 1: for $z \leq g'_A/\gamma_1$, we have $\phi_{t^+}(z) - \phi_{t^-}(z) = \left(\sum_{i=1}^{n_{t^-}(z)+1} i^{1-1/\alpha}\right) - (n_{t^-}(z) + 1)^{1-1/\alpha} \cdot (n_{t^-}^*(z) + 1) - \left(\sum_{i=1}^{n_{t^-}(z)} i^{1-1/\alpha}\right) + n_{t^-}(z)^{1-1/\alpha} n_{t^-}^*(z) = (n_{t^-}(z)^{1-1/\alpha} - (n_{t^-}(z) + 1)^{1-1/\alpha}) \cdot n_{t^-}^*(z) \leq 0$.

Case 2: for $g'_A/\gamma_1 \leq z \leq w'$, we have $\phi_{t^+}(z) - \phi_{t^-}(z) = \left(\sum_{i=1}^{n_{t^-}(z)} i^{1-1/\alpha}\right) - n_{t^-}(z)^{1-1/\alpha} (n_{t^-}^*(z) + 1) - \left(\sum_{i=1}^{n_{t^-}(z)} i^{1-1/\alpha}\right) + n_{t^-}(z)^{1-1/\alpha} n_{t^-}^*(z) = -n_{t^-}(z)^{1-1/\alpha} \leq 0$.

Hence, $\Phi(t^+) = \eta \int_0^\infty \phi_{t^+}(z) dz \leq \eta \int_0^\infty \phi_{t^-}(z) dz = \Phi(t^-)$, and the arrival condition holds.

- *Completion condition*: when a job completes under either N-X or the optimal, the potential function $\Phi(t)$ is unchanged, because in such case, $n(t)$ or $n^*(t)$ is unchanged for all $z \geq 0$. Hence, the completion condition holds.

- *Running condition*: At any time t , suppose that the optimal offline scheduler sets the speed of the j -th processor to $s_j^*(t)$. We have $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + u_t^* = n_t^* + \sum_{j=1}^P (s_j^*(t))^\alpha$. Also, from the proof of Lemma 6.5, we know that $\frac{dG_{\text{NX}}(\mathcal{J}(t))}{dt} \leq \frac{\alpha}{\alpha-1} n_t$, and $\frac{dG_B(\mathcal{J}(t))}{dt} \geq \frac{\alpha}{\alpha-1} (1 - x_t) n_t$, where $x_t = n_t^A/n_t$.

Now, we find an upper bound on the rate of change $\frac{d\Phi(t)}{dt}$ for the potential function $\Phi(t)$ at time t by focusing on the set $\mathcal{J}_A(t)$ of active jobs that satisfy property A. We assumed that algorithm X at any time t allocates exactly $a_i(t) = P/n_t$ processors to each active job J_i that satisfies property A and the other jobs gets processor allocations not more than P/n_t . In the worst case, the n_t^A jobs that satisfy property A have the most remaining processing power with property A while the optimal offline scheduler gives all processor resources to the job with the least remaining work. Hence, as Figure 6.2(c) shows, each of the bottom n_t^A horizontal stripes from $n_t(z)$ shrinks by $\frac{\sum_{j=1}^{a_i(t)} s_{ij}(t)}{\gamma_1} \Delta t$, and the top horizontal stripe from $n_t^*(z)$ shrinks by $\left(\sum_{j=1}^P s_j^*(t)\right) \Delta t$ in interval Δt . The change of the potential function over an infinitesimally small interval

Δt is then bounded by

$$\begin{aligned}
\frac{d\Phi(t)}{dt} &= \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i^{1-1/\alpha} \right) - n_{t+\Delta t}(z)^{1-1/\alpha} n_{t+\Delta t}^*(z) \right] dz \\
&\quad - \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) - n_t(z)^{1-1/\alpha} n_t^*(z) \right] dz \\
&\leq \frac{\eta}{\Delta t} \int_0^\infty \left[\left(\sum_{i=1}^{n_{t+\Delta t}(z)} i^{1-1/\alpha} \right) - \left(\sum_{i=1}^{n_t(z)} i^{1-1/\alpha} \right) \right] dz \\
&\quad + \frac{\eta}{\Delta t} \int_0^\infty [n_t(z)^{1-1/\alpha} (n_t^*(z) - n_{t+\Delta t}^*(z)) + n_t^*(z) (n_t(z)^{1-1/\alpha} - n_{t+\Delta t}(z)^{1-1/\alpha})] dz \\
&\leq \frac{\eta' H_P^{1/\alpha}}{P^{1-1/\alpha}} \left(- \sum_{i=1}^{n_t^A} i^{1-1/\alpha} \cdot \frac{\sum_{j=1}^{a_i(t)} s_{ij}(t)}{\gamma_1} \right) \\
&\quad + \frac{\eta' H_P^{1/\alpha}}{P^{1-1/\alpha}} \left(n_t^{1-1/\alpha} \sum_{j=1}^P s_j^*(t) + n_t^* \sum_{i=1}^{n_t} (i^{1-1/\alpha} - (i-1)^{1-1/\alpha}) \cdot \frac{\sum_{j=1}^{a_i(t)} s_{ij}(t)}{\gamma_1} \right).
\end{aligned}$$

By telescoping, we have $\sum_{i=1}^{n_t} (i^{1-1/\alpha} - (i-1)^{1-1/\alpha}) = n_t^{1-1/\alpha}$, and according to Corollary 6.9, which is based on Young's Inequality stated in Lemma 6.8, we have $n_t^{1-1/\alpha} \sum_{j=1}^P s_j^*(t) \leq \frac{\lambda(H_P \cdot P)^{1-1/\alpha}}{\alpha} \sum_{j=1}^P s_j^*(t)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}(H_P \cdot P)^{1/\alpha}} P n_t$, where λ is a constant to be specified later. Substituting these bounds as well as the upper and lower bounds of $\left(\sum_{j=1}^{a_i(t)} s_{ij}(t) \right)$ given in Lemma 6.3 into $\frac{d\Phi(t)}{dt}$ and simplify, we have

$$\frac{d\Phi(t)}{dt} \leq \eta' \left(- \frac{x_t^2}{4(\alpha-1)^{1/\alpha} \gamma_1} n_t + \frac{\lambda H_P}{\alpha} \sum_{j=1}^P (s_j^*(t))^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)} n_t} + \frac{\alpha}{(\alpha-1)^{1+1/\alpha} \gamma_1} n_t^* \right).$$

Now, we set $\eta' = \frac{4\alpha^2 \gamma_1}{(\alpha-1)^{1-1/\alpha}}$ and $\lambda = (4(\alpha-1)^{1/\alpha} \gamma_1)^{\alpha-1}$. Substituting $\frac{d\Phi(t)}{dt}$ as well as the rates of change $\frac{dG_{NX}(\mathcal{J}(t))}{dt}$, $\frac{dG^*(\mathcal{J}^*(t))}{dt}$ and $\frac{dG_B(\mathcal{J}(t))}{dt}$ into Inequality (6.6), we can see that in order to satisfy the running condition for all values of x_t , the multiplier c_1 can be set to $\max\{\frac{4\alpha^3}{(\alpha-1)^2}, (4\gamma_1)^\alpha \alpha H_P\}$. $\square$

Lemma 6.8 (Young's Inequality [87]) *If f is a continuous and strictly increasing function on $[0, c]$ with $c > 0$, $f(0) = 0$, $a \in [0, c]$ and $b \in [0, f(c)]$, then $ab \leq \int_0^a f(x)dx + \int_0^b f^{-1}(x)dx$, where f^{-1} the inverse function of f .* $\square$

Corollary 6.9 *For any $n_t \geq 0$, $s_j^*(t) \geq 0$ and $\lambda > 0$, we have that $n_t^{1-1/\alpha} s_j^*(t) \leq \frac{\lambda(H_P \cdot P)^{1-1/\alpha}}{\alpha} s_j^*(t)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}(H_P \cdot P)^{1/\alpha}} n_t$.*

Proof. Setting $f(x) = \lambda(H_P \cdot P)^{1-1/\alpha} x^{\alpha-1}$, $a = s_j^*(t)$ and $b = n_t^{1-1/\alpha}$, the corollary is directly implied by Young's Inequality. $\square$

Lemma 6.7 gives the amortized local performance of the N-X algorithm. We can

combine it with the result of Lemma 6.4 for the total response time plus energy of N-X in nonbatched scenario.

Theorem 6.10 *Suppose that N-X schedules a set $\mathcal{J}$ of jobs on P processors. Then the total response time plus energy $G_{\text{NX}}(\mathcal{J})$ of the job set satisfies*

$$G_{\text{NX}}(\mathcal{J}) \leq (c_1 + 2\alpha \cdot \gamma_2) \cdot G^*(\mathcal{J}), \quad (6.7)$$

where $G^*(\mathcal{J})$ is the total response time plus energy of $\mathcal{J}$ under the optimal offline scheduler, $c_1 = \max\{\frac{4\alpha^3}{(\alpha-1)^2}, (4\gamma_1)^\alpha \alpha H_P\}$, and γ_1 and γ_2 are the multipliers given in Inequalities (6.1) and (6.2).

Proof. Let T denote the set of time instances when a new job arrives or when an existing job completes under either N-X or an optimal offline scheduler. Integrating the running condition in Lemma 6.7 over time, we get $G_{\text{NX}}(\mathcal{J}) + \Phi(\infty) - \Phi(0) + \sum_{t \in T} (\Phi(t^-) - \Phi(t^+)) \leq c_1 \cdot G^*(\mathcal{J}) + 2\alpha \cdot G_B(\mathcal{J})$, where t^- and t^+ denote the time instances right before and after time t . Now, applying the boundary condition, the arrival condition and the completion condition to the above inequality, we can get that $G_{\text{NX}}(\mathcal{J}) \leq c_1 \cdot G^*(\mathcal{J}) + 2\alpha \cdot G_B(\mathcal{J})$. Substituting the property of $G_B(\mathcal{J})$ given in Lemma 6.4 into the above inequality, we prove the theorem. $\square$

6.3 Performances of N-AGCEQ and N-ACCEQ

In this section, we will apply the analysis framework presented in the preceding section to two scheduling algorithms, namely, N-AGCEQ and N-ACCEQ, and show their performances with respect to the total response time plus energy. In particular, N-AGCEQ uses a variant of the AGDEQ algorithm and N-ACCEQ uses a variant of the ACDEQ algorithm presented in Chapter 3.

We first define the algorithms AGCEQ and ACCEQ, which combine respectively the A-GREEDY and the A-CONTROL task schedulers with the CEQ (Conservative Equi-partitioning) algorithm. In both cases, the processor desire $d_i(q)$ for job J_i in each quantum q is still calculated as shown in Algorithms 1 and 3. Since our analysis framework requires that the OS allocator can not give more processors than the equal share P/n_t to any job at any time t , CEQ achieves this by allocating the minimum of the equal share P/n_t and the processor desire $d_i(q)$ of job J_i at the beginning of each quantum q , as shown in Algorithm 6.

We now prove the performances of N-AGCEQ and N-ACCEQ. As shown in the analysis framework, the key to bounding the total response time plus energy of a scheduling algorithm is to find multipliers γ_1 and γ_2 in Inequalities (6.1) and (6.2).

We start with the N-AGCEQ algorithm. Note that job J_i is now said to be efficient in quantum q if we have $w_i(q) \geq \delta \cdot \left(\sum_{j=1}^{a_i(q)} s_{ij}(q) \right) L$. Otherwise, job J_i is said to be

Algorithm 6 CEQ

Require: number n_t of active jobs when quantum q begins, processor desire $d_i(q)$ of each active job J_i in quantum q , and total number P of processors

Ensure: processor allocation $a_i(q)$ of each active job J_i in quantum q

1: $a_i(q) = \min\{P/n_t, d_i(q)\}$

inefficient if $w_i(q) < \delta \cdot \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L$, where δ is A-GREEDY's utilization parameter and L is the quantum length. In addition, we choose property A to be “accounted” and property B to be “deductible”. Since an accounted quantum is also deprived by definition, we know that any accounted job gets exactly the equal share P/n_t of processors at the beginning of each quantum, thus satisfying the requirement of the analysis framework.

Lemma 6.11 *Suppose that N-AGCEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then for each job $J_i \in \mathcal{J}$, its accounted processing power $g_A(J_i)$ and deductible response time plus scaled energy $g_B(J_i)$ satisfy*

$$\begin{aligned} g_A(J_i) &\leq \frac{1}{\delta} \cdot w(J_i), \\ g_B(J_i) &\leq \frac{2}{1-\delta} \cdot (2H_P)^{1/\alpha} \cdot g_1^*(J_i), \end{aligned}$$

where δ is the utilization parameter of A-GREEDY.

Proof. Since an accounted quantum is also efficient by definition, then for each job J_i in an accounted quantum, we have $w_i(q) \geq \delta \cdot \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L$. Let A denote its set of accounted quanta. We can get $w(J_i) \geq \sum_{q \in A} w_i(q) \geq \delta \cdot \sum_{q \in A} \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L = \delta \cdot g_A(J_i)$. Hence, we have $g_A(J_i) \leq \frac{1}{\delta} \cdot w(J_i)$.

For job J_i , let ES and I denote the set of efficient and satisfied quanta and the set of inefficient quanta, respectively. According to the proof of Theorem 3.5, we have that $|ES| \leq |I| + o(1)$. In addition, let B denote the set of deductible quanta, and according to definition, we have $|B| = |ES| + |I| \leq 2|I| + o(1)$, which suggests that the job's deductible response time plus scaled energy satisfies $g_B(J_i) \leq 2g_I(J_i) + o(1)$, where $g_I(J_i)$ denotes the job's inefficient response time plus scaled energy. Moreover, we say that job J_i is *saturated* at time t if its instantaneous parallelism is at least its processor allocation, thus all allocated processors are doing work. Otherwise, we say that the job J_i is *unsaturated*. In each inefficient quantum q , we have $w_i(q) < \delta \cdot \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L$. Hence, the saturated time is no more than $\frac{w_i(q)}{\left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right)} < \delta L$. The unsaturated time is then at least $(1-\delta)L$. Let $g_U(J_i) = \frac{\alpha}{\alpha-1} \int_0^\infty [J_i(t) \in \mathcal{J}_U(t)] dt$ denote the job's unsaturated response time plus scaled energy, where $\mathcal{J}_U(t)$ is the set of unsaturated jobs at time t . We then have $g_B(J_i) \leq \frac{2}{1-\delta} \cdot g_U(J_i) + o(1)$. During a time interval Δt when job J_i is unsaturated and does not experience any phase transition, suppose that the job is in its k -th phase, thus it has instantaneous parallelism h_i^k . The work

completed for J_i by N-AGCEQ during Δt is given by $\left(\sum_{j=1}^{h_i^k} s_{ij}(q)\right) \cdot \Delta t$. According to the proof of Lemma 6.2, to complete the same section of the job, the response time plus energy incurred by an optimal offline scheduler is at least $\frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{\left(\sum_{j=1}^{h_i^k} s_{ij}(q)\right) \cdot \Delta t}{(h_i^k)^{1-1/\alpha}} \geq \frac{\alpha}{\alpha-1} \cdot \frac{1}{(2H_P)^{1/\alpha}} \cdot \Delta t$. Thus, we have $g_1^*(J_i) \geq \frac{\alpha}{\alpha-1} \cdot \frac{1}{(2H_P)^{1/\alpha}} \int_0^\infty [J_i(t) \in \mathcal{J}_U(t)] dt = \frac{1}{(2H_P)^{1/\alpha}} \cdot g_U(J_i)$, which suggests that $g_B(J_i) \leq \frac{2}{1-\delta} \cdot (2H_P)^{1/\alpha} \cdot g_1^*(J_i)$.² $\square$

Theorem 6.12 *Suppose that N-AGCEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then with respect to the total response time plus energy, N-AGCEQ achieves*

- $\left(\frac{1+\delta}{\delta(1-\delta)} \cdot 2^{1+1/\alpha} H_P^{1/\alpha}\right)$ -competitive if all jobs in $\mathcal{J}$ are batched;
- $\left(\max\left\{\frac{4\alpha^3}{(\alpha-1)^2}, \left(\frac{4}{\delta}\right)^\alpha \alpha H_P\right\} + \frac{4\alpha}{1-\delta} \cdot (2H_P)^{1/\alpha}\right)$ -competitive for nonbatched jobs,

where δ is the utilization parameter of A-GREEDY.

Proof. The theorem is directly implied by combining the results of Theorems 6.6 and 6.10 with that of Lemma 6.11. $\square$

Now, we turn to the performance of the N-ACCEQ algorithm. Again, we set the convergence rate v of A-CONTROL to 0, hence it achieves one-step convergence. Moreover, we choose properties A and B to be “accounted” and “deductible”, respectively. The following lemma gives the properties of N-ACCEQ.

Lemma 6.13 *Suppose that N-ACCEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then for each job $J_i \in \mathcal{J}$, its accounted processing power $g_A(J_i)$ and deductible response time plus scaled energy $g_B(J_i)$ satisfy*

$$\begin{aligned} g_A(J_i) &\leq 2 \cdot w(J_i), \\ g_B(J_i) &\leq (C+1) \cdot (2H_P)^{1/\alpha} \cdot g_1^*(J_i), \end{aligned}$$

where C is the maximum transition factor of all jobs in the job set.

Proof. For each job J_i in a quantum q , we now define its *quantum work efficiency* to be $\alpha_i(q) = \frac{w_i(q)}{\left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right)L} = \frac{w_i(q)}{\int_{t_q}^{t_q+L} \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) dt}$, and define its *quantum span efficiency* to be $\beta_i(q) = \frac{l_i(q)}{\int_{t_q}^{t_q+L} \left(\sum_{j=1}^{h_i^{k_t}} s_{ij}(q)\right) / h_i^{k_t} dt}$, where t_q is the time when quantum q starts and $h_i^{k_t}$ is the instantaneous parallelism of the job at time t . Following the arguments in Section 3.4.1, we can get

$$\int_{t_q}^{t_q+L} \frac{\sum_{j=1}^{\bar{a}_i(t)} s_{ij}(q)}{\bar{a}_i(t)} dt \leq \alpha_i(q) \int_{t_q}^{t_q+L} \frac{\sum_{j=1}^{a_i(q)} s_{ij}(q)}{a_i(q)} dt + \beta_i(q) \int_{t_q}^{t_q+L} \frac{\sum_{j=1}^{h_i^{k_t}} s_{ij}(q)}{h_i^{k_t}} dt, \quad (6.8)$$

²The additive constant, which is represented as $o(1)$ in the bound of $g_B(J_i)$, will be absorbed by the performance of the optimal offline scheduler when the job is sufficiently large.

where $\bar{a}_i(t) = \min\{a_i(q), h_i^{k_t}\}$.

Since an accounted quantum is under-allocated by definition, we have $a_i(q) \leq A_i(q) = \frac{w_i(q)}{l_i(q)} = \frac{\alpha_i(q)}{\beta_i(q)} \cdot \frac{\int_{t_q}^{t_q+L} \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) dt}{\int_{t_q}^{t_q+L} \left(\sum_{j=1}^{h_i^{k_t}} s_{ij}(q)\right) / h_i^{k_t} dt}$. Substituting this inequality into Inequality (6.8) and simplifying, we can get $\alpha_i(q) \geq 1/2$, which suggests that $w_i(q) \geq \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L/2$. Let A denote the set of accounted quanta. We then have $w(J_i) \geq \sum_{q \in A} w_i(q) \geq \sum_{q \in A} \left(\sum_{j=1}^{a_i(q)} s_{ij}(q)\right) L/2 = g_A(J_i)/2$. Hence, we have $g_A(J_i) \leq 2 \cdot w(J_i)$.

From definition of deductible quantum and Lemma 3.2, we get that $a_i(q) \geq A_i(q)/C$ for job J_i in each deductible quantum q . Combining it with the definition of $A_i(q)$ and substituting it into Inequality (6.8), we can get $\beta_i(q) \geq \frac{1}{C+1}$, hence suggesting $l_i(q) \geq \frac{1}{C+1} \int_{t_q}^{t_q+L} \frac{\sum_{j=1}^{h_i^{k_t}} s_{ij}(q)}{h_i^{k_t}} dt$. Applying this result as well as the argument from the proof of Lemma 6.11, we have $g_B(J_i) \leq (C+1) \cdot (2H_P)^{1/\alpha} \cdot g_1^*(J_i)$. $\square$

Theorem 6.14 *Suppose that N-ACCEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then with respect to the total response time plus energy, N-ACCEQ achieves*

- $\left((C+3) \cdot 2^{1+1/\alpha} H_P^{1/\alpha}\right)$ -competitive if all jobs in $\mathcal{J}$ are batched;
- $\left(\max\left\{\frac{4\alpha^3}{(\alpha-1)^2}, 8^\alpha \alpha H_P\right\} + 2\alpha(C+1) \cdot (2H_P)^{1/\alpha}\right)$ -competitive for nonbatched jobs,

where C is the maximum transition factor of all jobs in the job set.

Proof. The theorem is implied by combining the results of Theorems 6.6 and 6.10 with that of Lemma 6.13. $\square$

6.4 A Lower Bound on Non-clairvoyant Non-uniform Speed Scaling

We now prove a lower bound on the competitive ratio of any non-clairvoyant algorithm that uses non-uniform speed scaling, which is shown in the following theorem.

Theorem 6.15 *Any non-clairvoyant scheduling algorithm that uses non-uniform speed scaling is $\Omega(\ln^{1/\alpha} P)$ -competitive with respect to the total response time plus energy, where P is the total number of processors.*

Proof. The proof is based on a single job with constant parallelism h and work w , where $1 \leq h \leq P$ and $w > 0$. For any non-clairvoyant algorithm A , we can assume without loss of generality that it allocates all P processors to the job and the speeds of the processors satisfy $s_1 \geq s_2 \geq \dots \geq s_P \geq 0$. Moreover, the processor speeds do not change throughout the execution of the job. Let $u = \sum_{j=1}^P s_j^\alpha$ denote the power consumption of A at any time. The total response time plus energy of the

job set scheduled by A is then given by $G_A = (1+u) \frac{w}{\sum_{j=1}^h s_j}$. The optimal offline scheduler, knowing the parallelism h of the job, will allocate exactly h processors of speed $\left(\frac{1}{(\alpha-1)h}\right)^{1/\alpha}$, thus incurring a total response time plus energy of $G^* = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{h^{1-1/\alpha}}$. The competitive ratio of algorithm A is then given by $\frac{G_A}{G^*} = \frac{(\alpha-1)^{1-1/\alpha}(1+u)}{\alpha} \cdot \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$.

The adversary will now choose the parallelism h to maximize this ratio, that is, to find $\max_{1 \leq h \leq P} \frac{G_A}{G^*} = \frac{(\alpha-1)^{1-1/\alpha}(1+u)}{\alpha} \max_{1 \leq h \leq P} \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$, while the online algorithm A has the freedom to choose $(s_1, \dots, s_P)$ in an attempt to minimize $\max_{1 \leq h \leq P} \frac{G_A}{G^*}$ regardless of the choice of h . According to Lemma 6.16, the optimal non-clairvoyant algorithm will set the speeds $(s_1^*, \dots, s_P^*)$ of the processors such that they satisfy $\frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j^*} = \frac{(h-1)^{1-1/\alpha}}{\sum_{j=1}^{h-1} s_j^*}$ for all $h = 2, 3, \dots, P$, from which we can get $s_j^* = (j^{1-1/\alpha} - (j-1)^{1-1/\alpha}) s_1^*$ for all $j = 1, 2, \dots, P$. Since it can be easily shown that $j^{1-1/\alpha} - (j-1)^{1-1/\alpha} \geq \frac{1-1/\alpha}{j^{1/\alpha}}$, we will have $s_j^* \geq \frac{1-1/\alpha}{j^{1/\alpha}} s_1^*$. Substituting them into $u = \sum_{j=1}^P (s_j^*)^\alpha$, we can get $s_1^* \leq \frac{\alpha u^{1/\alpha}}{(\alpha-1)H_P^{1/\alpha}}$. The competitive ratio of any non-clairvoyant algorithm then satisfies $\frac{G_A}{G^*} \geq \frac{(\alpha-1)^{1-1/\alpha}(1+u)}{\alpha} \cdot \frac{1}{s_1^*} \geq \frac{(\alpha-1)^{1-1/\alpha}}{\alpha} \cdot \frac{\alpha-1}{\alpha} \cdot \frac{1+u}{u^{1/\alpha}} \cdot H_P^{1/\alpha} \geq \frac{\alpha-1}{\alpha} H_P^{1/\alpha}$. The last inequality is because $\frac{1+u}{u^{1/\alpha}}$ is minimized when $u = \frac{1}{\alpha-1}$. Since it is also known that $H_P = \Omega(\ln P)$, the theorem is proved. $\square$

Lemma 6.16 *For any $P \geq 1$, $\alpha > 1$ and $b > 0$, subject to the condition that $\sum_{j=1}^P s_j^\alpha = b$ and $s_1 \geq s_2 \geq \dots \geq s_P \geq 0$, $\max_{1 \leq h \leq P} \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j}$ is minimized when $(s_1, s_2, \dots, s_P)$ satisfy $\frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} = \frac{(h-1)^{1-1/\alpha}}{\sum_{j=1}^{h-1} s_j}$ for all $h = 2, \dots, P$.*

Proof. To prove this lemma, we transform the stated problem into a convex optimization problem. We then show that our proposed solution satisfies the KKT condition [38], which is known to be a sufficient condition for the optimality of convex minimization problems. This then leads to the proof of the lemma. First, by introducing a variable y , the original optimization problem can be transformed into the following minimization problem:

$$\begin{aligned} & \text{minimize} && y \\ & \text{subject to} && \sum_{j=1}^P s_j^\alpha = b \\ & && s_j \geq s_{j+1} \text{ for } j = 1, \dots, P-1 \\ & && y \geq \frac{h^{1-1/\alpha}}{\sum_{j=1}^h s_j} \text{ for } h = 1, \dots, P \end{aligned} \tag{6.9}$$

However, the above minimization problem is not convex because its equality constraint (Equation 6.9) is not linear. Substituting $z_j = s_j^\alpha$, we transform it into a convex

optimization problem as follows.

$$\begin{aligned} & \text{minimize} && y \\ & \text{subject to} && \sum_{j=1}^P z_j = b \end{aligned} \tag{6.10}$$

$$z_{j+1} - z_j \leq 0 \text{ for } j = 1, \dots, P-1 \tag{6.11}$$

$$\frac{h^{1-1/\alpha}}{\sum_{j=1}^h z_j^{1/\alpha}} - y \leq 0 \text{ for } h = 1, \dots, P \tag{6.12}$$

For this minimization problem, the objective function and the only equality constraint (Equation (6.10)) are linear, the inequality constraints (Inequalities (6.11) and Inequalities (6.12)) are convex. Note that Inequalities (6.12) are convex because of the fact that $1/f(x)$ is a convex function if $f(x)$ is a positive concave function, and $\sum_{j=1}^h z_j^{1/\alpha}$ is concave since $z_j^{1/\alpha}$ is concave for $\alpha > 1$. We have now transformed our min-max optimization problem into a convex minimization problem. We will prove that our proposed solution $(y^*, z_1^*, \dots, z_P^*)$, which has the form

$$y^* = \frac{h^{1-1/\alpha}}{\sum_{j=1}^h (z_j^*)^{1/\alpha}} \text{ for } h = 1, \dots, P, \tag{6.13}$$

is the optimal solution by showing that it satisfies the KKT condition. Let $x_j = j^{1-1/\alpha} - (j-1)^{1-1/\alpha}$ for $j = 1, \dots, P$ and apparently we have $x_j > x_{j+1}$. From Equation (6.13) and equality constraint (Equation (6.10)), we can get $z_j^* = b \cdot \frac{x_j}{\sum_{i=1}^P x_i^\alpha}$ and therefore $z_j^* > z_{j+1}^*$ for $j = 1, \dots, P-1$.

To prove $(y^*, z_1^*, \dots, z_P^*)$ satisfies the KKT condition, we need to show that it satisfies primal feasibility, dual feasibility, complementary slackness, and stationarity. It is not hard to see that the proposed solution satisfies the primal feasibility at Equation (6.10), Inequalities (6.11) and Inequalities (6.12). Let us now associate multipliers with constraints:

$$\begin{aligned} \lambda & : \sum_{j=1}^P z_j = b \\ w_j & : z_{j+1} - z_j \leq 0 \text{ for } j = 1, \dots, P-1 \\ \mu_h & : \frac{h^{1-1/\alpha}}{\sum_{j=1}^h z_j^{1/\alpha}} - y \leq 0 \text{ for } h = 1, \dots, P \end{aligned}$$

Since we have $z_j^* > z_{j+1}^*$ for $j = 1, \dots, P-1$, to satisfy complementary slackness, we get $w_j = 0$ for $j = 1, \dots, P-1$. Now we need to show that there exists λ and $\mu_h \geq 0$ such that dual feasibility and stationarity are satisfied. To derive stationarity

condition, let us look at the Lagrangian function:

$$L(y, z_j, \lambda, \mu_h) = y + \sum_{h=1}^P \mu_h \left(\frac{h^{1-1/\alpha}}{\sum_{j=1}^h z_j^{1/\alpha}} - y \right) + \lambda \left(\sum_{j=1}^P z_j - b \right).$$

Taking derivative of the Lagrangian function with respect to y and z_j , and substituting $(y^*, z_1^*, \dots, z_P^*)$ into it, we get the following set of stationarity conditions:

$$\sum_{h=1}^P \mu_h = 1, \quad (6.14)$$

$$\frac{(y^*)^2}{\alpha(z_j^*)^{1-1/\alpha}} \left(\sum_{h=j}^P \frac{\mu_h}{h^{1-1/\alpha}} \right) = \lambda \text{ for } j = 1, \dots, P. \quad (6.15)$$

Solving Equation (6.15), we have $\mu_h = c_h \cdot \lambda$, where $c_h = \frac{h^{1-1/\alpha} \left((z_h^*)^{1-1/\alpha} - (z_{h+1}^*)^{1-1/\alpha} \right) \alpha}{(y^*)^2}$, for each $h = 1, \dots, P$, and z_{P+1}^* is defined to be 0. According to the values of $(y^*, z_1^*, \dots, z_P^*)$, we know that $y^* > 0$, $z_h^* > 0$ and $z_h^* > z_{h+1}^*$. Therefore, we have $c_h > 0$ for $h = 1, \dots, P$. Substituting $\mu_h = c_h \cdot \lambda$ into Equation (6.14), we get $\lambda = \frac{1}{\sum_{h=1}^P c_h} > 0$, which implies that $\mu_h > 0$ for all $h = 1, \dots, P$. Thus, we have shown that the dual feasibility is satisfied. Moreover, there exists λ and μ_h that make our proposed solution $(y^*, z_1^*, \dots, z_P^*)$ satisfy the stationarity, hence the KKT condition. Therefore, it is the optimal solution for the convex minimization problem, and the corresponding speed assignment $s_j^* = (z_j^*)^{1/\alpha}$ is optimal for the original optimization problem. $\square$

6.5 Discussions

We have analyzed the performances of the N-AGCEQ and N-ACCEQ algorithms with respect to the total response time plus energy. From Theorems 6.12 and 6.14, we can see that both algorithms are $O(\ln^{1/\alpha} P)$ -competitive for batched jobs and $O(\ln P)$ -competitive for nonbatched jobs, since it is well-known that $H_P = O(\ln P)$. In particular, N-AGCEQ achieves the stated bounds when jobs under schedule are sufficiently large and the utilization parameter δ of A-GREEDY can be considered as constant. On the other hand, N-ACCEQ achieves these bounds when jobs have smooth parallelism variations, hence small transition factors. Moreover, Theorem 6.15 also suggests that the performances of N-AGCEQ and N-ACCEQ are asymptotically optimal in the batched setting. In fact, it was shown in [174, 176] that the N-EQUI algorithm, which combines the non-uniform speed scaling policy with the EQUI algorithm, achieves the same asymptotic performances as N-AGCEQ and N-ACCEQ with respect to the total response time plus energy for both batched and nonbatched jobs. In the next section, we will further evaluate the performances of N-AGCEQ and N-ACCEQ, and compare them with N-EQUI through simulations.

Figure 6.3: The performance comparison of N-ACCEQ and N-AGCEQ with N-EQUI in terms of total response time plus energy.

So far, all of these results assumed the maximum utilization policy in the execution model as described in Chapter 1. In the literature, Chan, Edmonds and Pruhs [49] assumed a different execution model, in which each job can be executed by multiple groups of processors. The processors in different groups can have different speeds, but the ones within the same group must share the same speed. The execution rate of a job at any time is given by the fastest rate of all groups. They proposed a scheduling algorithm MULTILAPS under this model based on the LAPS scheduler [48] for single processor speed scaling, and showed that it is $O(\log P)$ -competitive with respect to the total response time plus energy for nonbatched jobs. It is interesting that the same asymptotic results are achieved under two different execution models, and it should be useful to further illuminate the relation between the two models and identify more fundamental issues in multiprocessor speed scaling.

6.6 Empirical Evaluations

In this section, we conduct simulations to evaluate the performances of N-ACCEQ and N-AGCEQ with respect to the total response time plus energy, and to compare them with the N-EQUI algorithm for a set of jobs. Similar settings are used here as in evaluating the total response time performance of our scheduling algorithms in Section 4.5. The power parameter is set to $\alpha = 3$, following the well-known cube-root rule [41, 143]. Hence, according to the speed scaling policy of N-X as shown in Algorithm 5, the energy consumption is roughly half of the total response time for all three scheduling algorithms.

Figure 6.3 shows the simulation results. We can see that as with the total response time performances of the ACDEQ, AGDEQ and EQUI algorithms (see Figure 4.3(b)), the performances of N-ACCEQ, N-AGCEQ and N-EQUI with respect to the total

response time plus energy also exhibit converging behavior when the system load becomes increasingly heavy. In both cases, the reason is because each job tends to get more limited processor resources at heavier loads and hence there is less room for reallocating the processors among the jobs. Under light system loads, however, there is a notable difference between Figures 6.3 and 4.3(b). In particular, when total response time is the only concern, EQUI has comparable or even better performance than the two-level algorithms, because almost all jobs can be easily satisfied by EQUI under light loads, and excessive processor allocation to a job will not penalize its performance. However, this is no longer true when energy is also of concern, since the excessive processors allocated to a job by EQUI will now consume extra energy, thus incurring a penalty on the overall performance. This demonstrates that accurate processor allocation becomes even more important when energy is part of the objective, hence suggests that two-level adaptive scheduling has greater advantage over the simple EQUI-based algorithm in practical settings, where the parallelism of the jobs does not change frequently. From Figure 6.3, we can also see that N-ACCEQ always outperforms N-AGCEQ as we have observed many times in the previous experiments. That is again due to the more effective processor desire calculation of the A-CONTROL task scheduler.

Chapter 7

Semi-Clairvoyant Algorithms for Performance plus Energy on Variable Speed Processors

In this chapter, we continue to study adaptive scheduling that optimizes both performance and energy consumption on variable speed processors. However, unlike the previous chapters, in which we focused on non-clairvoyant algorithms that do not have any information regarding a job's current and future information, in this chapter, we study a class of semi-clairvoyant algorithms, which are aware of the job's instantaneous parallelism at any time but not its future parallelism and remaining work. Moreover, besides the total response time plus energy, we also consider the makespan plus energy for a set of jobs and present efficient scheduling algorithms for both metrics.

It is known that when performance such as total response time or makespan is the only concern, semi-clairvoyance does not significantly improve upon some well-known non-clairvoyant algorithms. In particular, the improvements come only in terms of smaller constants, since these non-clairvoyant algorithms already achieve $O(1)$ -competitive with respect to total response time or makespan. (See Chapter 2 for details.) In this chapter, we show that when both performance and energy consumption are of concern, semi-clairvoyance does have substantial benefits over non-clairvoyant scheduling.

Specifically, we present the U-CEQ algorithm, which combines a uniform speed scaling policy with the CEQ OS allocator that utilizes the instantaneous parallelism of the jobs at any time to allocate processors. We show that U-CEQ achieves $O(1)$ -competitive with respect to the total response time plus energy even for nonbatched jobs, thus significantly improves upon any non-clairvoyant algorithm. In addition, we present the P-FIRST algorithm, which works for any set of batched (PAR-SEQ)* jobs. We show that P-FIRST achieves $O(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy, and that it is the best asymptotic ratio possible in this setting. The content of this chapter was originally presented in [176, 177].

7.1 U-CEQ for Total Response Time plus Energy

We first present our semi-clairvoyant algorithm U-CEQ and analyze its performance with respect to the total response time plus energy in this section.

At time t , suppose that job J_i is in its k_t -th phase, thus it has instantaneous parallelism $h_i^{k_t}$. Upon calculating the equal share P/n_t of processors as well as knowing the instantaneous parallelism $h_i^{k_t}$ of the job, U-CEQ decides the processor allocation $a_i(t)$ and scales the processor speed $s_i(t)$ in a uniform manner for job J_i as shown in Algorithm 7.

Algorithm 7 U-CEQ

Require: number n_t of active jobs at time t , instantaneous parallelism $h_i^{k_t}$ of each active job J_i , and total number P of processors

Ensure: processor allocation $a_i(t)$ and speed $s_i(t)$ for each active job J_i at time t

- 1: $a_i(t) = \min\{h_i^{k_t}, P/n_t\}$
 - 2: $s_i(t) = \left(\frac{1}{(\alpha-1)a_i(t)}\right)^{1/\alpha}$
-

We say that job J_i is *satisfied* at any time t if $a_i(t) = h_i^{k_t}$, and that it is *deprived* if $a_i(t) < h_i^{k_t}$. As we can see, U-CEQ is similar to N-AGCEQ and N-ACCEQ in terms of the processor allocation policy, except that it never allocates more processors than a job's instantaneous parallelism. Thus, jobs whose parallelism is not more than the equal share of processors will be satisfied and jobs with large parallelism will be deprived and get the equal share. Moreover, the speeds of the processors in U-CEQ are scaled uniformly in such a way that each active job J_i at time t consumes power $u_i(t) = \frac{1}{\alpha-1}$ and therefore the overall power consumption is given by $u_t = \frac{n_t}{\alpha-1}$. Since there is no energy waste, we will show that this speed scaling policy also ensures sufficient execution rates of the jobs, which eventually leads to the competitive performance of the U-CEQ algorithm.

Theorem 7.1 *Suppose that U-CEQ schedules a set $\mathcal{J}$ of jobs on P processors. Then U-CEQ achieves $O(1)$ -competitive with respect to the total response time plus energy.*

Proof. As with the N-X algorithm analyzed in the preceding chapter, we will prove the $O(1)$ -competitiveness of U-CEQ for nonbatched jobs using amortized local competitiveness argument outlined in Section 6.2. Since batched scenario represents a special case of this general setting, the same bound also applies to batched job set.

We choose properties A and B to be “deprived” and “satisfied”, respectively. Apparently, for each job J_i , we have that its deprived processing power $g_A(J_i)$ and satisfied response time plus scaled energy $g_B(J_i)$ satisfy

$$\begin{aligned} g_A(J_i) &\leq w(J_i), \\ g_B(J_i) &\leq g_1^*(J_i). \end{aligned}$$

Thus, both γ_1 and γ_2 are equal to 1 in this case. In addition, we still adopt the potential function given in Equation (6.5), where η is now set to $\eta = \frac{\eta'}{P^{1-1/\alpha}}$ and $\eta' = \frac{2\alpha^2}{(\alpha-1)^{1-1/\alpha}}$. We can see that the boundary, the arrival and the completion conditions shown in Lemma 6.7 still hold regardless of the scheduling algorithm. We will now show that the execution of any job set $\mathcal{J}$ under U-CEQ (UC for short) satisfies the following running condition:

$$\frac{dG_{\text{UC}}(\mathcal{J}(t))}{dt} + \frac{d\Phi(t)}{dt} \leq \max\left\{\frac{2\alpha^2}{\alpha-1}, 2^\alpha\alpha\right\} \cdot \frac{dG^*(\mathcal{J}^*(t))}{dt} + 2\alpha \cdot \frac{dG_B(\mathcal{J}(t))}{dt}. \quad (7.1)$$

Following the proof of Lemma 6.7, we have $\frac{dG_{\text{UC}}(\mathcal{J}(t))}{dt} = \frac{\alpha}{\alpha-1}n_t$, $\frac{dG^*(\mathcal{J}^*(t))}{dt} = n_t^* + \sum_{j=1}^P s_j^*(t)^\alpha$, and $\frac{dG_B(\mathcal{J}(t))}{dt} = \frac{\alpha}{\alpha-1}(1-x_t)n_t$, where $x_t = n_t^A/n_t$. Moreover, we can show that the rate of change $\frac{d\Phi(t)}{dt}$ for the potential function $\Phi(t)$ at time t satisfies

$$\begin{aligned} \frac{d\Phi(t)}{dt} &\leq \frac{\eta'}{P^{1-1/\alpha}} \left(- \sum_{i=1}^{n_t^A} i^{1-1/\alpha} \cdot a_i(t) s_i(t) \right) \\ &\quad + \frac{\eta'}{P^{1-1/\alpha}} \left(n_t^{1-1/\alpha} \sum_{j=1}^P s_j^*(t) + n_t^* \sum_{i=1}^{n_t} (i^{1-1/\alpha} - (i-1)^{1-1/\alpha}) a_i(t) s_i(t) \right) \\ &\leq \eta' \left(- \frac{x_t^2}{2(\alpha-1)^{1/\alpha}} n_t + \frac{\lambda}{\alpha} \sum_{j=1}^P s_j^*(t)^\alpha + \frac{1-1/\alpha}{\lambda^{1/(\alpha-1)}} n_t + \frac{n_t^*}{(\alpha-1)^{1/\alpha}} \right), \end{aligned}$$

where $\lambda = 2^{\alpha-1}(\alpha-1)^{1-1/\alpha}$. Substituting these bounds into Inequality (7.1), we can see that the running condition holds for all values of x_t . Now, integrating the running condition over time, and applying the boundary, the arrival and the completion conditions, we can get $G_{\text{UC}}(\mathcal{J}) \leq \left(\max\left\{\frac{2\alpha^2}{\alpha-1}, 2^\alpha\alpha\right\} + 2\alpha \right) \cdot G^*(\mathcal{J})$, which suggests that U-CEQ is $O(1)$ -competitive with respect to the total response time plus energy, since the multiplier can be considered as constant with respect to P . $\square$

From Theorems 7.1 and 6.15, we can see that U-CEQ significantly improves upon any non-clairvoyant algorithm with respect to the total response time plus energy. The improvement comes from not wasting any energy yet still guaranteeing sufficient execution rates for the jobs. Since there are non-clairvoyant algorithms that perform similarly to the semi-clairvoyant algorithm DEQ with respect to the total response time alone [63, 71, 88], it reveals the importance of semi-clairvoyance when energy is also of concern.

7.2 P-FIRST for Makespan plus Energy

In this section, we consider the objective of makespan plus energy for a set of jobs. Unlike total response time plus energy, where the completion time of each job contributes to the overall objective function, makespan for a set of jobs is the completion time of the last job. Hence, in this case, the other jobs only contribute to the energy

consumption part of the objective, thus can be intuitively slowed down to consume less energy, and eventually lead to better overall performance.

We assume in this section that all jobs are released together in a single batch, and they are all (PAR-SEQ)* jobs, that is, each job consists of only fully-parallel and sequential phases. We propose a semi-clairvoyant algorithm P-FIRST (Parallel First) and show its performance with respect to the makespan plus energy.

7.2.1 Preliminaries and Lower Bounds

We first show that as far as minimizing the makespan plus energy for batched jobs is concerned, the optimal (online/offline) strategy executes the jobs with a constant total power of $\frac{1}{\alpha-1}$ at any time. This result corresponds to the *power equality property* proved by Pruhs, van Stee and Uthaisombut [157], which applies to any optimal offline algorithm for the makespan minimization problem with an energy budget.

Lemma 7.2 *For any schedule A on a set $\mathcal{J}$ of batched jobs, there exists a schedule B that executes the jobs with a constant total power of $\frac{1}{\alpha-1}$ at any time, and performs no worse than A with respect to the makespan plus energy, i.e., $H_B(\mathcal{J}) \leq H_A(\mathcal{J})$.*

Proof. For any schedule A on a set $\mathcal{J}$ of batched jobs, consider an interval Δt during which the speeds of all processors, denoted as $(s_1, s_2, \dots, s_P)$, remain unchanged. The makespan plus energy of A incurred by executing this section of the job set is given by $H_A = \Delta t(1 + u)$, where $u = \sum_{j=1}^P s_j^\alpha$. We now construct schedule B in such a way that it executes the same section of the job set by running the j -th processor at speed $k \cdot s_j$, where $j = 1, \dots, P$ and $k = \left(\frac{1}{(\alpha-1)u}\right)^{1/\alpha}$. Thus, schedule B will finish the same section in $\frac{\Delta t}{k}$ time, and its power consumption at any time during this interval is given by $\frac{1}{\alpha-1}$. The makespan plus energy of B incurred by executing the same section of the job set is $H_B = \frac{\Delta t}{k}(1 + \frac{1}{\alpha-1}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \Delta t u^{1/\alpha}$. Since $\frac{1+u}{u^{1/\alpha}}$ is minimized when $u = \frac{1}{\alpha-1}$, we have $\frac{H_A}{H_B} = \frac{(\alpha-1)^{1-1/\alpha}}{\alpha} \cdot \frac{1+u}{u^{1/\alpha}} \geq 1$, i.e., $H_A \geq H_B$. Extending the same argument to all such intervals in schedule A proves the lemma. $\square$

Now, with the help of Lemma 7.2, we derive the performances of the optimal offline scheduler for any batched set of (PAR-SEQ)* jobs. Given any job J_i in a set $\mathcal{J}$ of (PAR-SEQ)* jobs, define $J_{i,P}$ to be a job with a single fully-parallel phase of the same total work as J_i , and define $J_{i,S}$ to be a job with a single sequential phase of the same span as J_i . Note that since a fully-parallel phase has 0 span according to definition, that is, it does not contribute to the total span of a job, $J_{i,S}$ can be obtained by simply removing all fully-parallel phases of job J_i , and directly concatenating its sequential phases. Moreover, we also define job set $\mathcal{J}_P$ to be $\mathcal{J}_P = \{J_{i,P} : J_i \in \mathcal{J}\}$ and define $\mathcal{J}_S$ to be $\mathcal{J}_S = \{J_{i,S} : J_i \in \mathcal{J}\}$. The following lemma shows the optimal makespan plus energy for job set $\mathcal{J}$.

Lemma 7.3 *To schedule a batched set $\mathcal{J}$ of n (PAR-SEQ)* jobs on P processors, the optimal makespan plus energy $H^*(\mathcal{J})$ of the job set satisfies*

$$H^*(\mathcal{J}) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}}, \quad (7.2)$$

$$H^*(\mathcal{J}) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \left(\sum_{i=1}^n l(J_i)^\alpha \right)^{1/\alpha}. \quad (7.3)$$

Proof. Clearly, the optimal makespan plus energy for $\mathcal{J}_P$ and $\mathcal{J}_S$ will be no worse than that for the original job set $\mathcal{J}$, i.e., $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_P)$ and $H^*(\mathcal{J}) \geq H^*(\mathcal{J}_S)$, since the optimal schedule for $\mathcal{J}$ is always a valid schedule for $\mathcal{J}_P$ and $\mathcal{J}_S$. In the following, we will show that $H^*(\mathcal{J}_P)$ and $H^*(\mathcal{J}_S)$ satisfy Inequalities (7.2) and (7.3), respectively. Then the lemma is proved.

For job set $\mathcal{J}_P$, the optimal offline scheduler can execute the jobs in any order by treating them as a single job, since all jobs are fully-parallel in this case. Moreover, by the convexity of the power function, all P processors should be run with constant speed s throughout the entire execution. According to Lemma 7.2, we have $Ps^\alpha = \frac{1}{\alpha-1}$, hence $s = \left(\frac{1}{(\alpha-1)P} \right)^{1/\alpha}$. The makespan plus energy incurred is therefore $H^*(\mathcal{J}_P) = \frac{\sum_{i=1}^n w(J_i)}{Ps} (1 + Ps^\alpha) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}}$.

For job set $\mathcal{J}_S$, allowing the optimal offline scheduler to have at least $\max\{n, P\}$ processors can only improve its performance. It will then execute each job with a constant speed throughout execution, which is also due to the convexity of the power function. Moreover, all jobs are completed simultaneously in the optimal schedule, since otherwise those jobs completed earlier can be slowed down to save more energy without affecting the makespan. Let s_i denote the speed used by the optimal for job $J_{i,S}$. Thus we have $\frac{l(J_1)}{s_1} = \frac{l(J_2)}{s_2} = \dots = \frac{l(J_n)}{s_n}$. Moreover, the speeds satisfy $\sum_{i=1}^n s_i^\alpha = \frac{1}{\alpha-1}$ according to Lemma 7.2. Therefore, we get the optimal speed assignment $s_i = \frac{1}{(\alpha-1)^{1/\alpha}} \cdot \frac{l(J_i)}{(\sum_{i=1}^n l(J_i)^\alpha)^{1/\alpha}}$ for $i = 1, 2, \dots, n$. The optimal makespan plus energy is then given by $H^*(\mathcal{J}_S) = \frac{l(J_1)}{s_1} + \frac{l(J_1)}{s_1} (\sum_{i=1}^n s_i^\alpha) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} (\sum_{i=1}^n l(J_i)^\alpha)^{1/\alpha}$. $\square$

7.2.2 Performance of P-FIRST

We now present our semi-clairvoyant scheduling algorithm P-FIRST for any batched set $\mathcal{J}$ of (PAR-SEQ)* jobs and analyze its performance with respect to the makespan plus energy. Basically, P-FIRST first executes the fully-parallel phase of any job whenever possible, and then executes the sequential phases of all jobs at the same rate. Specifically, at any time t when there are n_t active jobs, P-FIRST works as shown in Algorithm 8.

Note that P-FIRST is designed to ensure that the overall energy consumption $E(\mathcal{J})$ and the makespan $M(\mathcal{J})$ of the job set satisfies $E(\mathcal{J}) = \frac{1}{\alpha-1} M(\mathcal{J})$, since from Algorithm 8 we can see that at any time t the total power consumption u_t is given by

Require: number n_t of active jobs at time t , instantaneous parallelism $h_i^{k_t}$ of each active job J_i , and total number P of processors

Ensure: processor allocation $a_i(t)$ and speed $s_i(t)$ for each active job J_i at time t

```

1: if  $\exists J_i$  such that  $h_i^{k_t} = \infty$  then
2:   // there exists a job in fully-parallel phase
3:    $a_i(t) = P$  and  $s_i(t) = \left(\frac{1}{(\alpha-1)P}\right)^{1/\alpha}$ 
4:   for any other job  $J_j$  do
5:      $a_j(t) = 0$  and  $s_j(t) = 0$ 
6:   else
7:     // all jobs are in sequential phases
8:      $P' = \min\{n_t, P\}$ 
9:     for each job  $J_i$  do
10:       $a_i(t) = P'/n_t$  and  $s_i(t) = \left(\frac{1}{(\alpha-1)P'}\right)^{1/\alpha}$ 

```

$u_t = \frac{1}{\alpha-1}$, and $E(\mathcal{J}) = \int_0^{M(\mathcal{J})} u_t dt$. The makespan plus energy of the job set under P-FIRST thus satisfies $H(\mathcal{J}) = E(\mathcal{J}) + M(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$.

Theorem 7.4 *Suppose that P-FIRST schedules any batched set $\mathcal{J}$ of (PAR-SEQ)* jobs on P processors. Then P-FIRST achieves $O(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy.*

Proof. Since the makespan plus energy of any job set $\mathcal{J}$ scheduled by P-FIRST satisfies $H(\mathcal{J}) = \frac{\alpha}{\alpha-1}M(\mathcal{J})$, we shall mainly focus on the makespan $M(\mathcal{J})$ by separately bounding the time when all P processors are utilized and the time when less than P processors are used.. We denote the two types of time as $M'(\mathcal{J})$ and $M''(\mathcal{J})$, respectively. Obviously, we have $M(\mathcal{J}) = M'(\mathcal{J}) + M''(\mathcal{J})$.

According to P-FIRST, the execution rate when all P processors are utilized is given by $\frac{P^{1-1/\alpha}}{(\alpha-1)^{1/\alpha}}$. In addition, the total work completed in this case is upper bounded by $\sum_{i=1}^n w(J_i)$. Hence, the time $M'(\mathcal{J})$ satisfies $M'(\mathcal{J}) \leq (\alpha-1)^{1/\alpha} \frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}}$.

We now bound the time $M''(\mathcal{J})$ when less than P processors are used, which only occurs when P-FIRST executes sequential phases. Since all jobs are batch released, the number of active jobs monotonically decreases with time. Let T denote the first time when the number of active jobs drops below P , and let $m = n_T$. Therefore, we have $m < P$. For each of the m active job J_i at time T , let $\bar{l}_i$ denote the remaining span of the job. Now, rename the jobs in non-decreasing order of their remaining span, i.e., $\bar{l}_1 \leq \bar{l}_2 \leq \dots \leq \bar{l}_m$. Observe that, at any time t , P-FIRST executes the sequential phases of all jobs at the same speed of $\left(\frac{1}{(\alpha-1)n_t}\right)^{1/\alpha}$. Hence, the sequential phases of the m jobs will complete in exactly the above order. Define $\bar{l}_0 = 0$, then the time $M''(\mathcal{J})$ is given by $M''(\mathcal{J}) = \sum_{i=1}^m \frac{\bar{l}_i - \bar{l}_{i-1}}{\left(\frac{1}{(\alpha-1)(m-i+1)}\right)^{1/\alpha}} = (\alpha-1)^{1/\alpha} \sum_{i=1}^m \left((m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}\right) \bar{l}_i$. For convenience, define $c_i = (m-i+1)^{1/\alpha} - (m-i)^{1/\alpha}$ for each $1 \leq i \leq m$, and we can get $c_i \leq \frac{1}{(m-i+1)^{1-1/\alpha}}$. Let

$R = \sum_{i=1}^m \bar{l}_i^\alpha$, and subject to this condition and the ordering constraint of $\bar{l}_i$, we can get that $\sum_{i=1}^m c_i \cdot \bar{l}_i$ is maximized when $\bar{l}_i = R^{1/\alpha} \cdot \frac{c_i^{\frac{1}{\alpha-1}}}{\left(\sum_{i=1}^m c_i^{\frac{\alpha}{\alpha-1}}\right)^{1/\alpha}}$, which can be derived using the Lagrange Multiplier method [38]. Hence, we have $M''(\mathcal{J}) \leq (\alpha-1)^{1/\alpha} \cdot R^{1/\alpha} \cdot \left(\sum_{i=1}^m c_i^{\frac{\alpha}{\alpha-1}}\right)^{1-1/\alpha} \leq (\alpha-1)^{1/\alpha} \cdot R^{1/\alpha} \cdot \left(\sum_{i=1}^m \frac{1}{m-i+1}\right)^{1-1/\alpha} = (\alpha-1)^{1/\alpha} \cdot R^{1/\alpha} \cdot H_m^{1-1/\alpha}$, where H_m denotes the m -th harmonic number.

The makespan plus energy of the job set scheduled by P-FIRST thus satisfies $H(\mathcal{J}) \leq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \left(\frac{\sum_{i=1}^n w(J_i)}{P^{1-1/\alpha}} + R^{1/\alpha} \cdot H_m^{1-1/\alpha}\right)$. Since it is obviously true that $\sum_{i=1}^n l(J_i)^\alpha \geq \sum_{i=1}^m \bar{l}_i^\alpha = R$, comparing the performance of P-FIRST with that of the optimal offline scheduler in Lemma 7.3, we have $H(\mathcal{J}) \leq (1 + H_m^{1-1/\alpha}) \cdot H^*(\mathcal{J}) = O(\ln^{1-1/\alpha} P) \cdot H^*(\mathcal{J})$, as $H_m = O(\ln m)$ and $m < P$. The theorem is proved. $\square$

From the proof of Theorem 7.4, we can see that the competitive ratio of P-FIRST is dominated by the sequential phases of the (PAR-SEQ)* jobs, and without knowing the jobs' future work, the best strategy for any online algorithm does seem to execute them at the same rate. In the following theorem, we confirm that P-FIRST is asymptotically optimal with respect to the makespan plus energy by proving a matching lower bound on the competitive ratio of any semi-clairvoyant algorithm.

Theorem 7.5 *Any semi-clairvoyant algorithm is $\Omega(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy, where P is the total number of processors.*

Proof. Consider a batched set $\mathcal{J}$ of P sequential jobs, where the i -th job has span $l(J_i) = \frac{1}{(P-i+1)^{1/\alpha}}$. Since the number of jobs in this case is the same as the number of processors, any reasonable online semi-clairvoyant algorithm as well as the optimal offline algorithm will assign exactly one job to one processor throughout their executions.

From Lemma 7.3, we know that the optimal offline algorithm incurs a makespan plus energy $H^*(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \left(\sum_{i=1}^n l(J_i)^\alpha\right)^{1/\alpha} = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} H_P^{1/\alpha}$ on job set $\mathcal{J}$. For any semi-clairvoyant algorithm A, we will show that P-FIRST performs no worse than A on $\mathcal{J}$, i.e., $H_{\text{PF}}(\mathcal{J}) \leq H_A(\mathcal{J})$. From the proof of Theorem 7.4, we get that $H_{\text{PF}}(\mathcal{J}) = \frac{\alpha}{\alpha-1} M(\mathcal{J}) = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^P ((P-i+1)^{1/\alpha} - (P-i)^{1/\alpha}) l(J_i) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^P \frac{l(J_i)}{\alpha(P-i+1)^{1-1/\alpha}} = \frac{1}{(\alpha-1)^{1-1/\alpha}} H_P$. Comparing it with the performance of the optimal offline algorithm proves the lemma.

Now, to show $H_{\text{PF}}(\mathcal{J}) \leq H_A(\mathcal{J})$, we construct schedules from A to P-FIRST in several steps without increase in the total cost from each step. For the schedule produced by A, the adversary always assigns the i -th job to the processor that first completes $\frac{1}{(P-i+1)^{1/\alpha}}$ amount of work with ties broken arbitrarily. For convenience, we name the processors such that the i -th job is assigned to the i -th processor. First, we construct schedule A' from A by executing each job J_i with constant speed s'_i , which is derived by taking the average speed of processor i in schedule A. Based on the convexity of the power function, the completion time of each job remains the

same in A' but the total energy may be reduced. Thus, we have $H_{A'}(\mathcal{J}) \leq H_A(\mathcal{J})$. Moreover, according to the adversarial strategy, the processor speeds in A' satisfy $s'_1 \geq s'_2 \geq \dots \geq s'_P$. We then construct schedule A'' by executing each job J_i with speed s'_P throughout its execution. Since the spans of the jobs satisfy $l(J_1) < l(J_2) < \dots < l(J_P)$, apparently the makespan of the jobs in A'' is still determined by job J_P and is the same as the makespan in A' , but the total energy may be reduced by slowing down other jobs. Therefore, we have $H_{A''}(\mathcal{J}) \leq H_{A'}(\mathcal{J})$. Note that the speeds of all processors are the same in A'' now. Finally, according to Lemma 7.2, we construct schedule B from A'' such that it consumes constant total power of $\frac{1}{\alpha-1}$ at any time and $H_B(\mathcal{J}) \leq H_{A''}(\mathcal{J})$. By observing that B is identical to P-FIRST, the proof is complete. $\square$

7.3 Discussions

In this chapter, we studied semi-clairvoyant scheduling for both total response time plus energy and makespan plus energy. For the objective of total response time plus energy, we showed that the semi-clairvoyant algorithm U-CEQ can significantly improve upon any non-clairvoyant algorithm. Intuitively, the improvement comes from the fact that a semi-clairvoyant algorithm can always allocate a “right” number of processors to a job upon knowing its parallelism at any time. Thus, no energy waste will be incurred and at the same time sufficient execution rate can be ensured. Moreover, since a semi-clairvoyant algorithm can take advantage of the parallelism information of the job, it completely obliterates the need for non-uniform speed scaling. It is not hard to see, however, that non-uniform speed scaling in the semi-clairvoyant setting is not beneficial. Instead, it will degrade the performance of the algorithm, since generally less energy will be consumed at the same execution rate with uniform speed.

For the objective of makespan plus energy, although we have not studied non-clairvoyant scheduling, it seems quite challenging for any non-clairvoyant algorithm to obtain good results. The intuition is that a non-clairvoyant algorithm can potentially make mistakes not only in speed assignment, but also in processor allocation for the jobs. The former mistake leads to bad performance since jobs that complete early can in fact be slowed down to save energy, and this contributes to the lower bound of semi-clairvoyant algorithms shown in this chapter. The situation may deteriorate further in the non-clairvoyant setting as more energy will be wasted or slower execution rate will result if a wrong number of processors is also allocated to a job. Thus, we conjecture that minimizing makespan plus energy is inherently more difficult than minimizing total response time plus energy, hence is likely to incur a larger lower bound in the non-clairvoyant setting.

Finally, the results of both semi-clairvoyant algorithms presented in this chapter show the benefits of extra job information, in particular the instantaneous parallelism,

on performance and energy consumption. Although algorithms based on such information are shown to be more efficient, the instantaneous parallelism may nevertheless change wildly for some jobs, thus making these algorithms hard to implement or less feasible in practice. Therefore, the studies in this chapter are more of theoretical interest, which is meant to explore the possibilities and the limits of extra job information. In addition, the proposed semi-clairvoyant algorithms can only work for parallel jobs with special structures, i.e., consisting of sequential and fully-parallel phases in the case of P-FIRST, or having slow parallelism variations in the case of U-CEQ. Since these special job structures do not fit into our general malleable job construction model described in Section 1.6.2, the empirical simulations are omitted in this chapter.

Chapter 8

Conclusion

In this thesis, I have presented several provably-efficient adaptive scheduling algorithms for malleable parallel jobs on multiprocessor systems. Adaptation is mainly achieved by varying the number of processors allocated to the jobs using a two-level framework. In addition, the speeds of the allocated processors are also adapted with the help of dynamic speed scaling if possible. I have primarily focused on non-clairvoyant scheduling, where minimal assumptions about the jobs are made. Furthermore, semi-clairvoyant scheduling algorithms are also proposed if the instantaneous parallelism of the jobs is accessible. The metrics include various objective functions concerning both performance and energy consumption of the jobs. This chapter summarizes our main contributions and discusses possible future work.

8.1 Summary

I summarize the main contributions of this thesis in this section. First of all, we have presented a stable task scheduler A-CONTROL, which improves the existing task scheduler A-GREEDY in terms of several transient and steady-state properties.

- We have shown using control-theoretic analysis that A-CONTROL achieves much more stable processor desire calculation than A-GREEDY. In addition, A-CONTROL also achieves other desirable control-theoretic performances, including zero steady-state error, zero overshoot and a user-configurable convergence rate, which A-GREEDY again fails to attain.

We then combine A-CONTROL with the DEQ OS allocator to form a two-level scheduler ACDEQ. We have shown that both ACDEQ and AGDEQ, which combines A-GREEDY with DEQ, achieve competitive performances with respect to several objective functions on fixed speed processors.

- We have shown that ACDEQ and AGDEQ achieve $O(1)$ -competitive with respect to the makespan for a set of jobs. Both algorithms also achieve $O(1)$ -competitive

with respect to the total response time for batched jobs, and $O(1)$ -speed $O(1)$ -competitive with respect to the total response time for nonbatched jobs.

- We have shown that ACDEQ and AGDEQ when combined with EQUI achieve $O(1)$ -competitive with respect to the set response time for a collection of job sets if all job sets are batched. Both algorithms also achieve $O(1)$ -speed $O(1)$ -competitive with respect to the set response time for nonbatched job sets.

We have also presented the N-ACCEQ and N-AGCEQ algorithms, which combine a non-uniform speed scaling policy with specific variants of ACDEQ and AGDEQ for processors whose speeds can be dynamically scaled.

- We have shown that both N-ACCEQ and N-AGCEQ achieve $O(\ln^{1/\alpha} P)$ -competitive with respect to the total response time plus energy for a set of batched jobs, where P is the total number of processors and α is the power parameter. Both algorithms also achieve $O(\ln P)$ -competitive with respect to the total response time plus energy for nonbatched jobs.
- We have shown that with respect to the total response time plus energy, any non-clairvoyant algorithm using uniform speed scaling is $\Omega(P^{(\alpha-1)/\alpha^2})$ -competitive, and any non-clairvoyant algorithm using non-uniform speed scaling is $\Omega(\ln^{1/\alpha} P)$ -competitive. The latter lower bound also suggests that N-ACCEQ and N-AGCEQ are asymptotically optimal in batched setting.

Besides theoretical analysis, our empirical results have confirmed the improvements of A-CONTROL over A-GREEDY. Moreover, the simulation results have also shown that ACDEQ-based schedulers outperform AGDEQ-based ones, and both types of schedulers outperform simple EQUI-based algorithms under various settings.

Finally, we have studied semi-clairvoyant scheduling and shown that it can significantly improve any non-clairvoyant algorithm on variable speed processors.

- We have presented the U-CEQ algorithm, which combines a uniform speed scaling policy with a conservative version of EQUI. We have shown that U-CEQ achieves $O(1)$ -competitive with respect to the total response time plus energy for any set of jobs.
- We have presented the P-FIRST algorithm, and have shown that it achieves $O(\ln^{1-1/\alpha} P)$ -competitive with respect to the makespan plus energy for any set of batched (PAR-SEQ)* jobs.
- We have shown that with respect to the makespan plus energy, any semi-clairvoyant algorithm is $\Omega(\ln^{1-1/\alpha} P)$ -competitive, which suggests that P-FIRST is asymptotically optimal in this setting.

To the best of our knowledge, this thesis is the first to present multiprocessor scheduling algorithm that offers both control-theoretic properties and algorithmic performance guarantees. The various upper bounds and lower bounds appeared in this thesis also suggest that our algorithms achieve either asymptotically optimal or nearly optimal performances with respect to several objective functions. Finally, it is worth noting that for objectives that involve sum of jobs' response time, such as total response time, set response time and total response time plus energy, the results were obtained using two important techniques, namely local competitiveness argument and amortized local competitiveness argument. These two techniques have been shown to be very effective when dealing with objectives of this kind.

8.2 Future Work

Now, I discuss some possible future work and open questions along the line of research presented in this thesis.

Adaptive Scheduling

Both task schedulers A-CONTROL and A-GREEDY considered in this thesis calculate processor desires for a job based on the job's execution in a single quantum. From experiences in model predictive control [45], it could be more effective to calculate processor desires using more than one historical quanta. This may require a task scheduler to model the parallel jobs as dynamic higher-order systems, and will be especially applicable to jobs with regular parallelism structure. In addition, better system-wide adaptivity may be achieved by dynamically adjusting quantum length or automatically tuning scheduler-specific parameters during runtime. Moreover, identifying alternative job characteristics, such as frequency or variance on a job's parallelism transition, may also help tighten the analysis of existing schedulers and better understand adaptive scheduling in general.

Hierarchical Scheduling

This thesis has mainly focused on two-level scheduling, where all jobs are organized under a single centralized OS allocator. As parallel jobs can have more sophisticated distributions, depending on their priorities or the structure of the organization from which jobs are submitted, it may be necessary to organize the jobs in multiple levels. Thus, it will be natural to explore hierarchical scheduling in this case. With properly defined objective functions, a scheduling algorithm for this problem needs to decide how to effectively allocate the processors in each level. Our preliminary study [47] on this topic has suggested that scheduling algorithms based on ACDEQ and AGDEQ are able to achieve scalable makespan for a set of jobs, that is, their makespan ratios are

not affected by the number of levels in the scheduling hierarchy. It will be interesting to explore other objective functions in this setting.

Heterogeneous Resource Scheduling

This thesis only considered homogeneous resource scheduling, i.e., there is only one type of processors in the system. However, parallel applications can incur operations on multiple processing resources, interleaving computations, I/Os, and communications. Many parallel systems nowadays also embed special-purpose processors like vector units, floating-point co-processors, and various I/O processors. Thus, how to efficiently schedule heterogeneous resources will be a useful future direction to consider. In our preliminary study [90] on this topic, we have proposed an algorithm that uses DEQ policy to allocate processors of each type to the jobs according to the jobs' instantaneous parallelism of each corresponding type. We have shown that the competitive ratios of this algorithm with respect to both total response time and makespan grow linearly with the number of resource types. However, whether it is possible to obtain better results remains an interesting open question.

Multiple-Release Scheduling

As far as the objectives that involve the response time of the jobs are concerned, such as total response time, set response time, or total response time plus energy, most researchers so far have only focused on scenarios where jobs either have different release times or are all released in a single batch. It is generally true that scheduling batched jobs can yield better performances than scheduling nonbatched ones. However, other scenarios between the two extremes do not seem to be fully understood by the scheduling community. Hence, it should be useful to study the scenarios lying in between the two extremes, i.e., consider jobs that are released in multiple batches. The results in this case can help further understand impact of various job arrival patterns on the behavior of a scheduling algorithm.

Energy-Efficient Scheduling

This thesis studied energy-efficient scheduling by applying the idealized power function and considering total response time plus energy or makespan plus energy for a set of jobs. To extend this line of research to more realistic power functions or other objective functions will be worth studying next. Moreover, for the objective of makespan plus energy, we have only studied the performance of semi-clairvoyant algorithms on jobs with specific parallelism structures. How to deal with jobs with arbitrary parallelism profiles and what would be the performance in the non-clairvoyant setting remain interesting open questions.

8.3 Concluding Remarks

Efficient schedulers are essential to the performance of computing systems, no matter whether they are personal computers with only a handful of cores, or clusters with tens of servers, or supercomputers with thousands of processors. This is why scheduling has been one of the most extensively studied areas in computer science since its earliest days, and it will remain an important area of active research in the foreseeable future. Today, as the entire computing community is experiencing revolutionary transition from sequential to parallel programming, there will soon be an unprecedented number of parallel applications that need to be efficiently scheduled on various multiprocessor platforms. I believe that adaptive scheduling will play an important role in addressing the scheduling challenges in this new era. This thesis has laid some algorithmic foundations and provided objective justifications for adaptive scheduling. At the same time, I am also pleased to see that there have been some successful implementations of adaptive scheduling on actual systems in the past decade (see, e.g., [186, 61, 165, 158, 187]). The positive results reported in these work confirm the potential benefits adaptive scheduling brings to the new multicore era. I am looking forward to seeing a wider adoption of various adaptive schedulers on high-performance computing platforms in the future.

List of Publications

Conference Papers

1. Hongyang Sun, Yangjie Cao and Wen-Jing Hsu. Non-clairvoyant Scheduling for Sets of Parallel Jobs with Fairness and Efficiency. Submitted, 2011.
2. Hongyang Sun, Yuxiong He and Wen-Jing Hsu. Speed Scaling for Energy and Performance with Instantaneous Parallelism. In *Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms in Computer Systems (TAPAS)*. Rome, Italy, 2011.
3. Yangjie Cao, Hongyang Sun, Depei Qian and Weiguo Wu. Scalable Hierarchical Scheduling for Multiprocessor Systems Using Adaptive Feedback-Driven Policies. In *Proceedings of the 8th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. Taipei, Taiwan, 2010.
4. Yangjie Cao, Hongyang Sun, Wen-Jing Hsu and Depei Qian. Malleable-Lab: A Tool for Evaluating Adaptive Online Schedulers on Malleable Jobs. In *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*. Pisa, Italy, 2010.
5. Hongyang Sun, Yangjie Cao and Wen-Jing Hsu. Competitive Two-level Adaptive Scheduling Using Resource Augmentation. In *Proceedings of the 14th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. Rome, Italy, 2009.
6. Hongyang Sun, Yangjie Cao and Wen-Jing Hsu. Non-clairvoyant Speed Scaling for Batched Parallel Jobs on Multiprocessors. In *Proceedings of the 6th ACM International Conference on Computing Frontiers*. Ischia, Italy, 2009.
7. Hongyang Sun and Wen-Jing Hsu. Adaptive B-Greedy (ABG): A Simple yet Efficient Scheduling Algorithm. In *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Miami, USA, 2008.
8. Yuxiong He, Hongyang Sun and Wen-Jing Hsu. Adaptive Scheduling of Parallel Jobs on Functionally Heterogeneous Resources. In *Proceedings of the 36th International Conference on Parallel Processing (ICPP)*. Xi'an, China, 2007.

Journal Papers

1. Hongyang Sun, Yangjie Cao and Wen-Jing Hsu. Efficient Adaptive Scheduling of Multiprocessors with Stable Parallelism Feedback. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):594-607, 2011.
2. Yuxiong He, Hongyang Sun, Wen-Jing Hsu. Improved Results for Scheduling Batched Parallel Jobs by Using a Generalized Analysis Framework. *Journal of Parallel and Distributed Computing*, 70(2):173-182, 2010.

Bibliography

- [1] <http://www.threadingbuildingblocks.org/>.
- [2] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, 2003.
- [3] K. Agrawal, Y. He, W.-J. Hsu, and C. E. Leiserson. Adaptive scheduling with parallelism feedback. In *PPoPP*, pages 100 – 109, New York City, NY, USA, 2006.
- [4] K. Agrawal, Y. He, and C. E. Leiserson. An empirical evaluation of work stealing with parallelism feedback. In *ICDCS*, pages 19 – 29, Lisbon, Portugal, 2006.
- [5] K. Agrawal, Y. He, and C. E. Leiserson. Adaptive work stealing with parallelism feedback. In *PPoPP*, pages 112–120, San Jose, CA, USA, 2007.
- [6] S. Albers. Algorithms for energy saving. *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 173–186, 2009.
- [7] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. In *STACS*, pages 621–633, Marseille, France, 2006.
- [8] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298, San Diego, CA, USA, 2007.
- [9] L. L. Andrew, M. Lin, and A. Wierman. Optimality, fairness, and robustness in speed scaling designs. In *ACM Sigmetrics*, pages 2007–2015, New York City, NY, USA, 2010.
- [10] L. L. Andrew, A. Wierman, and A. Tang. Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.
- [11] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. Thread scheduling for multiprogrammed multiprocessors. In *SPAA*, pages 119–129, Puerto Vallarta, Mexico, 1998.

- [12] K. J. Åström and T. Hägglund. *PID Controller : Thoery, Design and Tuning, 2nd Edition*. Instrument Society of America, 1995.
- [13] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 1989.
- [14] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539, Rome, Italy, 2004.
- [15] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, San Diego, CA, USA, 2003.
- [16] N. Bansal, D. P. Bunde, H.-L. Chan, and K. Pruhs. Average rate speed scaling. In *LATIN*, pages 240–251, Búzios, Brazil, 2008.
- [17] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *ICALP*, pages 409–420, Reykjavik, Iceland, 2008.
- [18] N. Bansal, H.-L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *SODA*, pages 693–701, New York, NY, USA, 2009.
- [19] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS*, pages 520–529, Rome, Italy, 2004.
- [20] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *STACS*, pages 460–471, Stuttgart, Germany, 2005.
- [21] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *SODA*, pages 805–813, New Orleans, LA, USA, 2007.
- [22] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.
- [23] L. Becchetti and S. Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *STOC*, pages 94–103, 2001.
- [24] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded computations on processors of different speeds. In *SPAA*, pages 13–21, Bar Harbor, ME, USA, 2000.
- [25] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240, Seattle, WA, USA, 2005.
- [26] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the Grid using AppleS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

- [27] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6(2):181–193, 1999.
- [28] M. Berry, D. Chen, P. Koss, D. Kuck, S. Lo, Y. Pang, L. Pointer, R. Roloff, A. Sameh, E. Clementi, S. Chin, D. Schneider, G. Fox, P. Messina, D. Walker, C. Hsiung, J. Schwarzmeier, K. Lue, S. Orszag, F. Seidl, O. Johnson, and R. Goodrum. The PERFECT Club Benchmarks: Effective performance evaluation of supercomputers. *International Journal of Supercomputer Applications*, 3(3):5–40, 1988.
- [29] B. N. Bershad, E. D. Lazowska, and H. M. Levy. Presto: a system for object-oriented parallel programming. *Software – Practice & Experience*, 18(8):713–732, 1988.
- [30] R. D. Blumofe. *Executing Multithreaded Programs Efficiently*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
- [31] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. In *PPoPP*, pages 207–216, Santa Barbara, CA, USA, 1995.
- [32] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM Journal on Computing*, 27(1):202–229, 1998.
- [33] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.
- [34] R. D. Blumofe and D. Papadopoulos. Hood: A user-level thread library for multiprogramming multiprocessors. Technical report, University of Texas at Austin, 1998.
- [35] R. D. Blumofe and D. Papadopoulos. The performance of work stealing in multiprogrammed environments. In *SIGMETRICS*, pages 266–267, Madison, WI, USA, 1998.
- [36] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.
- [37] G. E. P. Box, G. M. Jenkins, and G. Reinsel. *Time Series Analysis: Forecasting and Control, 3rd Edition*. Prentice Hall, 1994.
- [38] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [39] T. Brecht, X. Deng, and N. Gu. Competitive dynamic multiprocessor allocation for parallel applications. In *IPDPS*, pages 448 – 455, San Antonio, TX, USA, 1995.

- [40] T. Brecht and K. Guha. Using parallel program characteristics in dynamic processor allocation policies. *Performance Evaluation*, 4(27-28):519–539, 1996.
- [41] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.
- [42] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., 2001.
- [43] D. P. Bunde. Power-aware scheduling for makespan and flow. In *SPAA*, pages 190–196, Cambridge, MA, USA, 2006.
- [44] M. Calzarossa, A. P. Merlo, D. Tessa, G. Haring, and G. Kotsis. A hierarchical approach to workload characterization for parallel systems. In *HPCN*, pages 102–109, Milan, Italy, 1995.
- [45] E. F. Camacho and C. A. Bordons. *Model Predictive Control, 2nd Edition*. Springer-Verlag, 2004.
- [46] Y. Cao, H. Sun, W.-J. Hsu, and D. Qian. Malleable-Lab: A tool for evaluating adaptive online schedulers on malleable jobs. In *PDP*, pages 1–8, Pisa, Italy, 2010.
- [47] Y. Cao, H. Sun, D. Qian, and W. Wu. Scalable hierarchical scheduling for multiprocessor systems using adaptive feedback-driven policies. In *ISPA*, pages 1–8, Taipei, Taiwan, 2010.
- [48] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 409–420, Freiburg, Germany, 2009.
- [49] H.-L. Chan, J. Edmonds, and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, Calgary, Canada, 2009.
- [50] S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers, 5th Edition*. McGraw-Hill, 2005.
- [51] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with ϵ resource augmentation. In *STOC*, pages 363–372, Chicago, IL, USA, 2004.
- [52] B. Chen and A. P. A. Vestjens. Scheduling on identical machines: How good is LPT in an on-line setting. *Operations Research Letters*, 21(4):165–169, 1997.

- [53] J.-J. Chen, K. Iwama, T.-W. Kuo, and H.-I. Lu. Flow time minimization under energy constraints. In *ASP-DAC*, pages 866–871, Yokohama, Japan, 2007.
- [54] Z.-L. Chen. Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, 129:135–153, 2004.
- [55] S.-H. Chiang, R. K. Mansharamani, and M. K. Vernon. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In *ACM sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 33–44, Nashville, TN, USA, 1994.
- [56] W. Cirne and F. Berman. A model for moldable supercomputer jobs. In *IPDPS*, page 59, San Francisco, CA, USA, 2001.
- [57] R. Conway, W. Maxwell, and L. Miller. *Theory of Scheduling*. Addison-Wesley, 1967.
- [58] E. C. Coopers and R. P. Draves. C threads. Technical Report CMU-CS-88-154, Department of Computer Science, Carnegie-Mellon University, June, 1988.
- [59] J. Corbalán and J. Labarta. Dynamic speedup calculation through self-analysis. Technical Report UPC-DAC-1999-43, Departament d’Arquitectura de Computadors, Universitat Politècnica de Catalunya, 1999.
- [60] J. Corbalán and J. Labarta. Improving processor allocation through run-time measured efficiency. In *IPDPS*, pages 74–79, San Francisco, CA, USA, 2001.
- [61] J. Corbalán, X. Martorell, and J. Labarta. Performance-driven processor allocation. In *OSDI*, Berkeley, CA, USA, 2000.
- [62] X. Deng and P. Dymond. On multiprocessor system scheduling. In *SPAA*, pages 82–88, Padua, Italy, 1996.
- [63] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996.
- [64] DESMO-J: A framework for discrete-event modelling and simulation. <http://asi-www.informatik.uni-hamburg.de/desmoj/>.
- [65] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Load balancing using MIMO linear control. *IBM Research Report RC23213(W0309-186)*, 2003.
- [66] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Incorporating cost of control into the design of a load balancing controller. In *RTAS*, pages 376–385, Toronto, Canada, 2004.

- [67] A. B. Downey. A parallel workload model and its implications for processor allocation. In *HPDC*, page 112, Portland, OR, USA, 1997.
- [68] D. L. Eager, J. Zahorjan, and E. D. Lazowska. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers*, 38(3):408–423, 1989.
- [69] J. Edmonds. Every deterministic nonclairvoyant scheduler has a suboptimal load threshold. Available at <http://www.cse.yorku.ca/jeff/research/>.
- [70] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.
- [71] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, pages 120–129, El Paso, TX, USA, 1997.
- [72] J. Edmonds, S. Datta, and P. Dymond. TCP is competitive against a limited adversary. In *SPAA*, pages 174–183, San Diego, CA, USA, 2003.
- [73] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, pages 685–692, New York, NY, USA, 2009.
- [74] D. G. Feitelson. Parallel workloads archive. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
- [75] D. G. Feitelson. Packing Schemes for Gang Scheduling. In *JSSPP*, pages 89–110, Honolulu, HI, USA, 1996.
- [76] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems (extended version). *IBM Research Report RC19790(87657) 2nd Revision*, 1997.
- [77] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *JSSPP*, pages 188–206, Cambridge, MA, USA, 2001.
- [78] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *JSSPP*, pages 1–24, Orlando, FL, USA, 1998.
- [79] G. F. Franklin, M. L. Workman, and D. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., 1997.
- [80] A. Goel, J. Walpole, and M. Shor. Real-rate scheduling. In *RTAS*, pages 434–441, Toronto, Canada, 2004.
- [81] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.
- [82] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

- [83] G. Greiner, T. Nonner, and A. Souza. The bell is ringing in speed-scaled multi-processor scheduling. In *SPAA*, pages 11–18, Calgary, AB, Canada, 2009.
- [84] D. Grunwald, I. Charles B. Morrey, P. Levis, M. Neufeld, and K. I. Farkas. Policies for dynamic clock scheduling. In *OSDI*, pages 6–6, San Diego, CA, USA, 2000.
- [85] A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications. In *SIGMETRICS*, pages 120–132, San Diego, California, United States, 1991.
- [86] R. H. Halstead Jr. MultiLisp: a language for concurrent symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–538, 1985.
- [87] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, second edition, 1988.
- [88] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.
- [89] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient online non-clairvoyant adaptive scheduling. In *IPDPS*, pages 1–10, Long Beach, CA, USA, 2007.
- [90] Y. He, H. Sun, and W.-J. Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. In *ICPP*, Xi’an, China, 2007.
- [91] Y. He, H. Sun, and W.-J. Hsu. Improved results for scheduling batched parallel jobs by using a generalized analysis framework. *Journal of Parallel and Distributed Computing*, 70(2):173–182, 2010.
- [92] J. L. Hellerstein, Y. Diao, and S. Parekh. Applying control theory to computing systems. *IBM Research Report RC23456(W0412-008)*, 2004.
- [93] J. L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley-Interscience, 2004.
- [94] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*, pages 38–43, Portland, OR, USA, 2007.
- [95] High Performance Fortran Forum. High performance Fortran language specification. Technical report, Rice University, 1993.
- [96] W. D. Hillis and G. L. Steele, Jr. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, 1986.

- [97] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong. A control theoretic analysis of red. In *INFOCOM*, pages 1510–1519, Anchorage, AK, USA, 2001.
- [98] K. Hong and J.-T. Leung. On-line scheduling of real-time tasks. *IEEE Transactions on Computers*, 41(10):1326–1331, 1992.
- [99] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
- [100] J. M. Jaffe. An analysis of preemptive multiprocessor job scheduling. *Mathematics of Operations Research*, 5(3):415–421, 1980.
- [101] V. Janjic, K. Hammond, and Y. Yang. Using application information to drive adaptive grid middleware scheduling decisions. In *MAI*, pages 7–12, Oslo, Norway, 2008.
- [102] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan. Modeling of workload in MPPs. In *JSSPP*, pages 95–116, Geneva, Switzerland, 1997.
- [103] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. Technical report, NASA Ames Research Center, 1999.
- [104] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *FOCS*, pages 214–221, Milwaukee, WI, USA, 1995.
- [105] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
- [106] A. Kamra, V. Misra, and E. M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In *IEEE International Workshop on Quality of Service*, pages 47–56, Montreal, Canada, 2004.
- [107] C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In *Conference on Hot Topics in Operating Systems*, pages 9–13, Santa Fe, NM, USA, 2005.
- [108] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In *algorithms and theory of computation handbook*, Chapter 35, CRC Press. 1998.
- [109] M. Karlsson, X. Zhu, and C. Karamanolis. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *ICCA*, pages 709–714, Budapest, Hungary, 2005.
- [110] S. Keshav. A control-theoretic approach to flow control. In *Conference on Communications architecture & protocols*, pages 3–15, Zurich, Switzerland, 1991.

- [111] S. Khuller, J. Li, and B. Saha. Energy efficient scheduling via partial shutdown. In *SODA*, pages 1–13, Austin, TX, USA, 2010.
- [112] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, pages 123–134, Salt Lake City, UT, USA, 2008.
- [113] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.
- [114] T.-W. Lam, L.-K. Lee, H.-F. Ting, I. K. To, and P. W. Wong. Sleep with guilt and work faster to minimize flow plus energy. In *ICALP*, pages 665–676, Rhodes, Greece, 2009.
- [115] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *SPAA*, pages 256–264, Munich, Germany, 2008.
- [116] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Nonmigratory multiprocessor scheduling for response time and energy. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1527–1539, 2008.
- [117] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, Karlsruhe, Germany, 2008.
- [118] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Improved multiprocessor scheduling for flow time and energy. *Journal of Scheduling*, DOI 10.1007/s10951-009-0145-5, 2009.
- [119] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *handbooks in operations research and management science*, Volume 4, Chapter 9, North Holland, 1993.
- [120] D. Leijen, W. Schulte, and S. Burckhardt. The design of a task parallel library. In *OOPSLA*, pages 227–242, Orlando, FL, USA, 2009.
- [121] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *STOC*, pages 110–119, El Paso, TX, USA, 1997.
- [122] J. Y.-T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
- [123] S. T. Leutenegger and M. K. Vernon. The performance of multiprogrammed multiprocessor scheduling policies. In *SIGMETRICS*, pages 226–236, Boulder, Colorado, United States, 1990.

- [124] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. RinnooyKan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
- [125] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9):1632–1650, 1999.
- [126] R. Liu, K. Klues, S. Bird, S. Hofmeyry, K. Asanović, and J. Kubiatowicz. Tessellation: Space-time partitioning in a manycore client OS. In *HotPar*, pages 1–6, Berkeley, CA, USA, 2009.
- [127] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 163–176, Nice, France, 2005.
- [128] C. Lu. *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, Charlottesville, VA, USA, 2001.
- [129] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *RTAS*, pages 51–62, Taipei, Taiwan, 2001.
- [130] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1-2):85–126, 2002.
- [131] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):550–561, 2005.
- [132] Y. Lu, T. F. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework for QoS guarantees and its application to differentiated caching services. In *International Workshop on Quality of Service*, pages 23–32, Miami Beach, FL, USA, 2002.
- [133] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *ICDCS*, pages 615–622, Phoenix, AZ, USA, 2001.
- [134] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.
- [135] W. Ludwig and P. Tiwari. Scheduling malleable and nonmalleable parallel tasks. In *SODA*, pages 167–176, Arlington, VA, USA, 1994.

- [136] S. Majumdar, D. L. Eager, and R. B. Bunt. Scheduling in multiprogrammed parallel systems. In *SIGMETRICS*, pages 104–113, Santa Fe, NM, USA, 1988.
- [137] S. Majumdar, D. L. Eager, and R. B. Bunt. Characterization of programs for scheduling in multiprogrammed parallel systems. *Performance Evaluation*, 13(2):109–130, 1991.
- [138] T. Matsumoto. Competitive analysis of the round robin algorithm. In *ISAAC*, pages 71–77, Nagoya, Japan, 1992.
- [139] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, 1993.
- [140] J. McCullough and E. Torng. SRPT optimally utilizes faster machines to minimize flow time. In *SODA*, pages 343–351, New Orleans, LA, USA, 2004.
- [141] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.
- [142] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *SODA*, pages 422–431, Austin, TX, USA, 1993.
- [143] T. Mudge. Power: A first-class architecture design constraint. *Computer*, 34(4):52–58, 2001.
- [144] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Maximizing speedup through self-tuning of processor allocation. In *IPPS*, pages 463–468, Honolulu, HI, USA, 1996.
- [145] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Parallel application characterization for multiprocessor scheduling policy design. In *JSSPP*, pages 175–199, Honolulu, HI, USA, 1996.
- [146] T. D. Nguyen, R. Vaswani, and J. Zahorjan. Using runtime measured workload characteristics in parallel processor scheduling. In *JSSPP*, pages 155–174, Honolulu, HI, USA, 1996.
- [147] J. K. Ousterhout. Scheduling techniques for concurrent systems. In *ICDCS*, pages 22–30, Miami/Ft. Lauderdale, FL, USA, 1982.
- [148] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster-based web services. *IEEE Journal on Selected Areas in Communications*, 23(12):2333–2343, 2005.

- [149] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys*, pages 289–302, Lisbon, Portugal, 2007.
- [150] S. Parekh, N. Gandhi, J. L. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23(1-2):127–141, 2002.
- [151] S. Parekh, K. Rose, Y. Diao, V. Chang, J. L. Hellerstein, S. Lightstone, and M. Huras. Throttling utilities in the IBM DB2 universal database server. *IBM Research Report RC23163(W0403-139)*, 2004.
- [152] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *STOC*, pages 140–149, El Paso, TX, USA, 1997.
- [153] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag New York, Inc., Third edition, 2008.
- [154] K. Pruhs. Competitive online scheduling for server systems. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.
- [155] K. Pruhs, E. Torong, and J. Sgall. Online scheduling. In *handbook of scheduling: Algorithms, models, and performance analysis*, Chapter 15, CRC Press. 2004.
- [156] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. In *SWAT*, pages 14–25, Humlebæk, Denmark, 2004.
- [157] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *WAOA*, pages 307–319, Mallorca, Balear Islands, Spain, 2005.
- [158] C. J. R. Rajesh Sudarsan. ReSHAPE: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. In *ICPP*, page 44, Xi’an, China, 2007.
- [159] J. Robert and N. Schabanel. Ordonnancement non-clairvoyant: petites simplifications et améliorations de l’analyse de la famille d’algorithmes $LAPS_\beta$. Available at <http://hal.archives-ouvertes.fr/hal-00384663/fr/>.
- [160] J. Robert and N. Schabanel. Non-clairvoyant batch set scheduling: Fairness is fair enough. In *ESA*, pages 741–753, Eilat, Israel, 2007.
- [161] J. Robert and N. Schabanel. Pull-based data broadcast with dependencies: be fair to users, not to items. In *SODA*, pages 238–247, New Orleans, LA, USA, 2007.

- [162] J. Robert and N. Schabanel. Non-clairvoyant scheduling with precedence constraints. In *SODA*, pages 491–500, San Francisco, CA, USA, 2008.
- [163] Y. Robert and F. Vivien. *Introduction to Scheduling*. CRC Press, 2009.
- [164] R. Różycki and J. Weglarz. On job models in power management problems. *Bulletin of the Polish Academy of Science Technical Sciences*, 57(2):147–151, 2009.
- [165] S. Sen. Dynamic processor allocation for adaptively parallel jobs. Master’s thesis, Massachusetts Institute of technology, 2004.
- [166] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 171–180, Oakland, CA, USA, 1989.
- [167] D. B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *SODA*, pages 448–454, Austin, Texas, USA, 1993.
- [168] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines online. In *FOCS*, pages 131–140, San Juan, Puerto Rico, 1991.
- [169] J. P. Singh, W. Weber, and A. Gupta. SPLASH: Stanford parallel applications for shared-memory. *Computer Architecture News*, 20(1):5–44, 1992.
- [170] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
- [171] M. S. Squillante, D. D. Yao, and L. Zhang. Analysis of job arrival patterns and parallel scheduling performance. *Performance Evaluation*, 36-37(1-4):137–163, 1999.
- [172] H. Sun, Y. Cao, and W.-J. Hsu. Non-clairvoyant scheduling for sets of parallel jobs with fairness and efficiency. *Submitted*.
- [173] H. Sun, Y. Cao, and W.-J. Hsu. Competitive two-level adaptive scheduling using resource augmentation. In *JSSPP*, pages 1–24, Rome, Italy, 2009.
- [174] H. Sun, Y. Cao, and W.-J. Hsu. Non-clairvoyant speed scaling for batched parallel jobs on multiprocessors. In *CF*, pages 99–108, Ischia, Italy, 2009.
- [175] H. Sun, Y. Cao, and W.-J. Hsu. Efficient adaptive scheduling of multiprocessors with stable parallelism feedback. *IEEE Transactions on Parallel and Distributed Systems*, 22(4):594–607, 2011.
- [176] H. Sun, Y. He, and W.-J. Hsu. Energy-efficient multiprocessor scheduling for flow time and makespan. *CoRR abs/1010.4110*, 2010.

- [177] H. Sun, Y. He, and W.-J. Hsu. Speed scaling for energy and performance with instantaneous parallelism. In *TAPAS*, pages 1–12, Rome, Italy, 2011.
- [178] H. Sun and W.-J. Hsu. Adaptive B-Greedy (ABG): A simple yet efficient scheduling algorithm. In *IPDPS*, pages 1–8, Miami, FL, USA, 2008.
- [179] M. A. Trick. Scheduling multiple variable-speed machines. In *IPCO*, pages 485–494, Waterloo, Canada, 1990.
- [180] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *SOSP*, pages 159–166, New York, NY, USA, 1989.
- [181] J. Turek, W. Ludwig, J. L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. S. Yu. Scheduling parallelizable tasks to minimize average response time. In *SPAA*, pages 200–209, Cape May, NJ, USA, 1994.
- [182] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994.
- [183] J. Turek, J. L. Wolf, and P. S. Yu. Approximate algorithms scheduling parallelizable tasks. In *SPAA*, pages 323–332, San Diego, CA, USA, 1992.
- [184] Z. Wang, X. Zhu, and S. Singhal. Utilization vs. SLO-based control for dynamic sizing of resource partitions. In *DSOM*, pages 133–144, Barcelona, Spain, 2005.
- [185] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *OSDI*, pages 13–23, Monterey, CA, USA, 1994.
- [186] J. B. Weissman. Prophet: automated scheduling of SPMD programs in workstation networks. *Concurrency: Practice and Experience*, 11(6):301–321, 1999.
- [187] J. B. Weissman, L. R. Abburi, and D. England. Integrated scheduling: the best of both worlds. *Journal of Parallel and Distributed Computing*, 63(6):649–668, 2003.
- [188] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*, pages 24–36, S. Margherita Ligure, Italy, 1995.
- [189] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, Milwaukee, WI, USA, 1995.
- [190] J. Zahorjan, E. D. Lazowska, and D. L. Eager. The effect of scheduling discipline on spin overhead in shared memory parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):180–198, 1991.

- [191] J. Zahorjan and C. McCann. Processor scheduling in shared memory multiprocessors. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–225, Boulder, CO, USA, 1990.
- [192] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *ICDCS*, pages 301–310, Vienna, Austria, 2002.