Managing contention on shared resources in a virtual organization

Percival, Xavier

2007

Percival, X. (2007). Managing contention on shared resources in a virtual organization.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2539>

<https://doi.org/10.32657/10356/2539>

Nanyang Technological University

Downloaded on 26 Apr 2023 19:06:07 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**MANAGING CONTENTION ON SHARED
COMPUTING RESOURCES IN A VIRTUAL
ORGANIZATION**

**XAVIER PERCIVAL
SCHOOL OF COMPUTER ENGINEERING
2007**

Managing Contention on Shared Computing Resources in a Virtual Organization

Xavier Percival

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Abstract

A computing grid can be organized as a virtual organization (VO) where distinct physical organizations collaborate to form a virtual pool of computing resources that is to be shared among users belonging to the VO.

Resource contention arises when the shared computing resources may not be sufficient to cope with the resource demands of users. We focus on resource contention coming from two potential sources – the user and the organization. Since users' requirements for computing resources are diverse in terms of both CPU and execution time, the grid is prone to a situation where large amounts of computing resources may be reserved by a single or a small group of users. When this happens, these users may deprive other users from gaining access to the shared resources. At the organization level, when the aggregate workload generated from an organization exceeds its resource contribution, this leads to resource contention at the inter-domain level.

Addressing resource contention requires a facility for admitting the workloads of each organization with respect to their contribution of shared resources. There are a variety of incentive schemes that can potentially be used to address the above two sources of resource contention. However, these incentive schemes are only sufficient to sustain an active interest among peers (organizations) to participate in the sharing community.

In this thesis, an extension of the token-exchange incentive scheme is proposed to manage contention on shared resources. The primary goal is to introduce additional mechanisms into the generic token-exchange incentive scheme so that it can be incorporated into a grid-based admission control framework for resource management.

To address user initiated contention, a pricing framework is developed as part of the token-exchange incentive scheme to administrate the admission of requests submitted by users. The role of the pricing framework is to prevent users from oversubscribing both number of CPUs and the duration required on each CPU with respect to the instantaneous degree of resource contention. The pricing framework translates a set of quality-of-service

parameters demanded by the user, and the instantaneous utilization of resources managed by a provider agent into an admission price. A consumer agent that acts on behalf of the user, must pay the amount stipulated by the admission price in order for the user's request to be successfully admitted by the broker's matchmaking policy. Our simulation results show that the system admission ratio of users's requests is significantly improved when the admission control mechanism is augmented with the pricing framework.

To address inter-domain contention, the pricing framework is further exploited to coordinate the exchange of tokens between participating organizations. The purpose of employing the token-exchange incentive scheme is to avoid the need to administer admission control is on a central broker. This improves the scalability of the system because accounting information need not be maintained at the broker to execute the admission control policies. However, by employing the token-exchange incentive scheme, we demonstrate from a mathematical analysis that if linearity in utility on expenditure, expenditure propensity on earnings and earnings on resource contribution are preserved, fairness will be zero. Fairness is a quantitative metric for assessing the degree of equitable resource usage amongst participating organizations. When fairness is zero, perfect equity amongst participating organizations is achieved.

We further show that the pricing framework helps in satisfying the conditions for linearity in earnings on resource contribution. Finally, because the performance trade-off between fairness and the system admission ratio is sensitive to the initial amount of tokens assigned to each organization, an adaptive policy is incorporated into the pricing framework to lessen the impact of token assignment on the trade-off. Experiments on the pricing framework and adaptive policing mechanisms were conducted through simulations. The results are presented and discussed in the thesis.

Acknowledgement

I would like to thank my supervisor Assoc. Prof. Cai Wentong for his patient guidance, kindness, encouragement and consideration throughout the course of my studies. I would also like to thank Assoc. Prof. Francis Lee for his concern and advice that he had given me during discussions pertaining to my work. Many thanks to the people in Parallel and Distributed Computing Centre (PDCC); especially to Irene and Lijun for their prompt attention to any of the technical problems I have faced when using the facilities in the centre. Last but not least, my colleagues, Tang Ming, Yuhong, and Xinjun for their help in so many ways.

Contents

1	Introduction	1
1.1	Background	1
1.1.1	The Computational Grid as a Virtual Organization	1
1.1.2	Sources of Resource Contention within a Virtual Organization . .	2
1.1.3	Attacking the Problem	3
1.2	Thesis Objectives, Scope and Contributions	5
1.2.1	Objectives and Scope	5
1.2.2	Main Contributions	6
1.3	Thesis Organization	8
2	System Architecture and Model Justifications	10
2.1	Introduction	10
2.2	Characterization of a Conceptual Framework for Metacomputing Systems	11
2.3	Mandatory Mechanisms for a Grid Infrastructure	15
2.3.1	Quality-of-Service Support	16
2.3.2	Advance Reservation	17
2.4	System Model	19

2.4.1	The Grid Computing Environment	19
2.4.2	Relational Mappings of System Entities	21
2.4.3	Quality-of-Service Parameters	22
2.5	Simulation Prototype	23
2.5.1	Overview	25
2.5.2	Job Management	27
2.6	Summary	29
3	Existing Approaches for Managing Contention on Shared Resources	31
3.1	Introduction	31
3.2	Requirements for Contention Management on Shared Resources	34
3.3	User-initiated Contention Management Schemes	36
3.3.1	Differentiated Services	36
3.3.2	Auction	38
3.4	Inter-domain Contention Management Schemes	43
3.4.1	Grid-wide Schedulers	44
3.4.2	Incentives	46
3.5	Discussion	55
3.5.1	A Pricing Framework for Admission Control of Advance Reservations	57
3.5.2	A Metric for Evaluating Fairness	58
3.5.3	System Architecture Considerations	59
3.5.4	Delegation of Trading Decisions	60
3.6	Summary	60

4	A Pricing Framework for Supporting Advance Reservations to Manage User-initiated Contention	63
4.1	Introduction	63
4.2	Related Work	65
4.3	Non-preemptive versus Preemptive Admission Control Strategy	67
4.3.1	Overview of System Architecture	67
4.3.2	A Motivating Example	68
4.4	Admission Control Framework	72
4.4.1	Pricing Framework Design	72
4.4.2	Time Delay	75
4.4.3	Admission Control and Resource Allocation	78
4.5	Performance Evaluation	81
4.5.1	Workload Trace Generation	81
4.5.2	Simulation Parameters and Performance Metrics	82
4.5.3	Benchmarks	84
4.5.4	Simulation Framework	85
4.6	Results and Discussion	86
4.6.1	Impact of Consumer Strategy on the System Admission Ratio	86
4.6.2	Effect of Workload Traces on System Admission Ratio	91
4.6.3	Effect of Consumer Strategy on Average Delay	97
4.7	Conclusions	104
5	A Preliminary Analysis on Inter-domain Resource Contention	106

5.1	Introduction	106
5.2	Inter-domain Contention on Shared Resources	108
5.2.1	Request Submission	109
5.2.2	Simulation Model	110
5.2.3	Queue Storm	111
5.3	Impact of Load-contribution Ratio on Inter-domain Contention	111
5.3.1	Analysis of VO Parameters on Queue Storms	114
5.3.2	The Effect of Introducing Delay to Requests and its Impact on Queue Storms	117
5.3.3	Discussion	120
5.4	Admission Control on the Broker's Queue	121
5.4.1	Request Management Framework	121
5.4.2	Experiments	123
5.5	Summary	128
6	Employing the Token-exchange Incentive Scheme to Manage Inter-domain Contention	131
6.1	Introduction	131
6.2	A Metric for Managing Inter-domain Contention	132
6.2.1	Measuring Utility and Resource Contribution	133
6.2.2	The Community Utility-contribution Ratio	134
6.2.3	Defining Metric for Fairness	136
6.3	Employing the Token-exchange Scheme to Achieve Fairness	138

6.3.1	Conceptual Resource Management Framework with Token-exchange Incentive Scheme	139
6.3.2	An Analysis on the Necessary Conditions to Achieve Fairness . .	140
6.3.3	Discussion	144
6.4	Managing Inter-domain Contention using the Pricing Framework	145
6.4.1	Achieving Linearity of Earnings on Contribution using the Pricing Framework	146
6.4.2	Achieving Linearity of Utility on Earnings based on the Pricing Framework	150
6.4.3	Discussion	151
6.5	Summary and Conclusions	152
7	Adaptive Policies for Managing Inter-domain Contention	154
7.1	Introduction	154
7.2	Related Work	155
7.3	Design of Trading Policies	156
7.3.1	Accounting Ratios	157
7.3.2	Policy Design	160
7.3.3	Formulation of Bids and Admission Price	165
7.4	Experimental Procedure	166
7.4.1	Performance Metrics	166
7.4.2	Variables	166
7.5	Results	167
7.6	Summary and Conclusions	174

8	Conclusions	176
8.1	Summary of Contributions	176
8.2	Concluding Remarks and Future Directions	178
8.2.1	The Pricing Framework and its Current Limitations	178
8.2.2	Relationship between VO-level Objective and System Architecture Design	181
	Appendices	183
A	Derivation of the Pricing Framework	183
B	Mathematical Workload Model for Observing the Effect of Job Arrival Correlation on Resource Contention	184
C	Derivation of Linearity in Earnings on Resource Contribution	186

List of Figures

2.1	Conceptual Framework of a Metacomputing System	13
2.2	Architecture of the Prototyped System	24
2.3	Illustration of Advance Reservation Window	28
3.1	Taxonomy of Existing Schemes for Managing Contention on Shared Resources	33
3.2	Algorithm for the Iterative Combinatorial Auction based Primal-Dual Theory [66]	42
3.3	Illustration for Reputation Index Scheme	48
3.4	Illustration for Token-exchange Scheme	53
4.1	Illustration of (1) Preemptive, (2) Non-preemptive and (3) Load-controlled Non-preemptive Admission Control	71
4.2	Calculation of the Admission Price of an Arbitrary Request	74
4.3	Full-load Contention on a Request Arrival Burst with (Top) and Without (Bottom) the Application of Delay	77
4.4	Admission Control of Request	79
4.5	Effect of Consumer Strategy on System Admission Ratio with Workload KTH SP2 for (1) $p=4.0$, (2) $p=40.0$, (3) $p=400.0$ and (4) $p=4000.0$. . .	87

4.6	Effect of Consumer Strategy on System Admission Ratio with Workload CTC SP2 for (1) $p = 4.0$, (2) $p = 40.0$, (3) $p = 400.0$ and (4) $p = 4000.0$. .	88
4.7	Effect of Consumer Strategy on System Admission Ratio with Workload SCSD Par95 for (1) $p = 4.0$, (2) $p = 40.0$, (3) $p = 400.0$ and (3) $p = 4000.0$	89
4.8	Effect of Skew-02 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$	92
4.9	Effect of Skew-01 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$	93
4.10	Effect of Skew-005 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$	94
4.11	Effect of Skew on Average Delay for KTH SP2 Workloads	100
4.12	Effect of Skew on Average Delay for CTC SP2 Workloads	101
4.13	Effect of Skew on Average Delay for Par95 Workloads	102
4.14	Graph of Performance Trade-off Between UTIL-0 and UTIL-MIX	103
5.1	Schematic of the Simulation Framework for Centralized Policing Scheme to Address Inter-domain contention	110
5.2	Community Workload Intensity Profile for (1) $H = 0.6$ and (2) $H = 0.8$. . .	113
5.3	Marginal Effect of N (MRC_N top) for $H = 0.6$ (left) and $H = 0.8$ (right) and the Marginal Effect of LC (MRC_{LC} bottom) for $H = 0.6$ (left) and $H = 0.8$ (right) on Queue Storms	116
5.4	Impact of Slack on System Admission Ratio in Presence of Queue Storms for $H = 0.6$ and $H = 0.8$	118
5.5	The Admission Control Framework for Centralized Policy	122

5.6	Impact of Queue Size Threshold on System Admission Failure Ratio for different LC ratios for $H = 0.6$ (top) and $H = 0.8$ (bottom)	124
5.7	Performance of Queue Size Thresholding for Different STF	125
5.8	The Admission Control Algorithm	126
5.9	System Admission Failure Ratio Performance for Different Request Mixes given Different Queue Size Thresholding Methods	128
6.1	An Illustration on Full-load and Partial contention on Shared Resources .	137
6.2	An Illustration Of the flow of aggregate utility gained by participating organizations based on the Token-exchange scheme	143
7.1	Schematic of Each Organization's Decision Mechanism.	157
7.2	Policy Schemes, COOPERATIVE (top) and COMPETITIVE (bottom) for Provider Agents	162
7.3	Policy Scheme for Consumer Agents	165
7.4	Impact of Initial Budget on System Admission Ratio for LC=0.4 (top), N=2 (left) and N=5 (right) and LC=2.0 (bottom), N=2 (left) and N=5 (right)	168
7.5	Impact of Initial Budget on Fairness for LC=0.4 (top), N=2 (left) and N=5 (right) and LC=2.0 (bottom), N=2 (left) and N=5 (right)	169
7.6	Demand and Supply Price Indices of COOPERATIVE for N=5 and Budget = 50, with LC=0.4 (top) and LC=2.0 (bottom)	172
7.7	Demand and Supply Price Indices of COOPERATIVE for N=5 and Budget = 1000, with LC=0.4 (top) and LC=2.0 (bottom)	173

List of Tables

2.1	Justification of Conceptual Framework from Existing Metacomputing Systems	14
3.1	Summary of Taxonomy of Contention Management Schemes	56
4.1	Quality-of-service Requirements and Submission Times of Requests . . .	69
4.2	Service Instance Assignment for Different Admission Control Schemes . .	69
4.3	Parameters for Synthetic Generation of Workload Traces	82
4.4	Results for Consistency in System Admission Ratio for Different Time Scale and LC Ratio	96
5.1	The Effect of Varying N and LC on Queue Storms	117
5.2	A Measure of Significance of Slack on Queue Storms	119
5.3	Workloads to Examine Effect of Queue Size Thresholding for Request Management	126
7.1	Values of β for Accounting Ratios	160
7.2	Mapping of Accounting Ratios to Distinct States	160

List of Major Notations

Type	Symbol	Chapter	Description
Utility	μ^{tot}	3,6	Total utility gained by all users for all epoches.
	μ_t^{tot}	3,6	Total utility gained by all users at epoch t .
	μ_k	3,6	Total utility gained by organization k for all epoches.
	$\mu_{k,t}$	3,6	Total utility gained by organization k at epoch t .
	μ_k^i	3	Utility gained by user in organization k when request r_i is admitted.
	$\mu_{k,t}^e$	6	Total expected utility of organization k at epoch t .
	$\mu(r_i, s_j)$	6	Utility gained by request r_i when it is assigned service s_j .
Resource Contribution	C_t^{tot}	3,6	Total resource contribution by organization k at epoch t .
	$C_{k,t}$	3,6	Resource contribution of organization k at epoch t .
	C, C_k	2,3,4,5,6,7	Resource contribution by an arbitrary organization or organization k for all epoches.
Resource Usage	$\Delta C, \Delta C_k$	4,6	Workload of an arbitrary organization or organization k .
	$C(s_j)$	6	Resource capacity used up by service s_j .
	$\frac{\Delta C}{C}$	4,6	Load ratio or utilization of an organization resources for all epoches.
	$\frac{\Delta C_k}{C_{k,t}}$	4,6	Load ratio or utilization of organization k 's resources at epoch t .
Utility Contribution Ratio	$\frac{\mu_k}{C_k^{tot}}$	3,6	Utility-contribution ratio of organization k for all epoches.
	$\frac{\mu_i}{C_t^{tot}}$	6	Utility-contribution ratio of all organizations at epoch t .
	$\frac{\mu_{k,t}}{C_{k,t}}$	3,6	Utility-contribution ratio of organization k , at epoch t .
	LC_k	4,5,7	Load-contribution ratio of organization k . (i.e., the ratio between the load generated by organization k and its resource contribution.)
	Υ_{opt}	5,6	Utility-contribution threshold for each organization.
Pricing Framework	$\Gamma_{\omega, \tau}$	4	Pricing framework.
	ω	4	Minimum CPU usage of all request.
	τ	4	Nominal time scale for pricing framework.
	σ_c, σ_p	7	Consumer and provider agent price indices.
Budget	$B_{k,t}$	6	Budget of organization k at epoch t .
	$B_{k,t}^+$	6	Earnings of organization k at epoch t .
	$B_{k,t}^-$	6	Expenditure of organization k at epoch t .
	B_t^{avg}	6	Average Budget of an organization at epoch t .
System Admission Ratio	μ	4,7	The ratio between total number of admitted and submitted requests.
Fairness	Φ	6,7	Metric for fairness.
Time Intervals	Δt	4	Time step for advance reservation schedule.
	ΔT	5,6	Time interval of an epoch.

Chapter 1

Introduction

1.1 Background

1.1.1 The Computational Grid as a Virtual Organization

In recent years, the existence of high speed networks has catalyzed the adoption of clusters of workstations to support the execution of parallel computing applications. Furthermore, cluster management systems enable machines (workstations) of different architectures and capabilities to coordinate the execution of parallel workloads. For the above reasons, cluster computing has become an economical alternative to supercomputing and has gained widespread attention in both research and commercial arenas. In the research community, there are a large number of projects involved in cluster computing. The Beowulf project enables ubiquitous linux workstations to be linked together across multiple Ethernet with TCP/IP support [12]. Research in Network of Computers (NOW) that combines distributed workstations into a single system, also investigates areas involving network interface hardware, fast communication protocols, distributed file systems, distributed scheduling and job control [13]. Solaris MC (Multi-Computer) is a distributed operating system for a multi-computer, that is, a cluster of computing nodes connected by a high-speed interconnect [79]. It provides a single system image, making the cluster appear like a single machine to the users, applications, and the network. In the industry, a plethora

of clusters management systems have also been developed, for example, Loadlever [60], PBS [68], Sun Grid Engine [81] and LSF [61].

In order to serve computation intensive applications that continue to scale due to their ever increasing complexity, the inherent benefit of pooling heterogeneous clusters has led to the development of software tools to facilitate in aggregating disperse computing clusters into common computing environment, which is commonly known as a computational grid.

In general, a Virtual Organization (VO) is a collaboration between distinct physical organizations that share their computing resources. Essentially, the shared resources are combined to form computational grid. This means that resources contributed by each participating organization are linked to a common infrastructure that can be accessed by any user who is a member of a participating organization of the VO.

The primary advantage of resource sharing is to allow distinct physical organizations to extend beyond their domain-level resource limits for on-demand exploitation of computing resources of other organizations. The benefit accrued from resource sharing is that computing resources need not be purchased just to meet peak demand requirements. In the event that peak demand for resources occurs intermittently, participating organizations can somehow serve each other and therefore, resulting in more efficient use of resources at each organization.

1.1.2 Sources of Resource Contention within a Virtual Organization

Resource contention within a Virtual Organization is a perverse situation where the shared pool of resources cannot cope with the aggregate workloads submitted by users in the VO. This consequence is not in particular due to the fact that organizations deliberately do not contribute sufficiently to the community. But rather, VO resource administrators generally are unable to assess and therefore predict the size and duration

of sharp increases in aggregate demand for computing resources.

We place our attention on resource contention in the grid at two levels:

- User-initiated Contention

In the case of a computational grid, the number of CPUs and duration of their availability are critical resources that are likely to be competed for in the event of resource contention. Users who independently submit requests to the grid to have their jobs executed, have the freedom to independently define their own resource needs. Since the resource requirements of users are likely to be diverse, there is a chance that a single request may take up more resources than a larger number of other requests. As such, this single request can subsequently deprive other jobs of gaining access to the shared resources if it is admitted. Hence, from a management perspective there is a need to prevent a small group of users from dominating their use of computing resources in the event of resource contention.

- Inter-domain level Contention

In a situation where a large group of users from all organizations compete for shared resources at the same time, the combined workload of any organization must also be controlled with respect to the organization's contribution of resources so as to achieve a reasonable degree of equity among all participating organizations. This means that the quantity of admitted workload generated by each organization must be proportional to its contribution. Without policies to enforce this requirement, organizations are not motivated to limit their overall access to shared resources and can therefore be another potential source of contention at the inter-domain level.

1.1.3 Attacking the Problem

To deal with the above issues, we first consider the problem of overloading shared resources due to contention as a consequence of free riding. In our context, free riding is

a situation where users of an organization try to consume resources in a manner that the aggregate workload demand at the organization exceeds the capacity of the actual resource contribution. It arises from resource competition among concurrent users because grid applications are likely to consume vast amounts of computational resources. A greater concern is when this competition intensifies to an extent such that the aggregate workload will lead to resource contention among the different organizations.

At large, free riding is a critical issue existent in peer-to-peer file sharing systems (e.g., [56, 57]) in both theory and practice. This is because free riding amongst small group of peers can incite selfish behavior on other cooperative peers and therefore cause the entire system to ultimately collapse. To operate successfully, a peer-to-peer system requires sustained peer cooperation – that is, peers must continuously contribute resources as much as they make use of other resources shared to the community. To promote cooperative behavior amongst peers, several incentives schemes have been proposed. Much of the current research focus on designing incentive schemes to motivate peer cooperation. Managing contention on shared resources on the other hand requires, in addition to the incentive schemes, some form of admission control mechanism so that excess demand of resources can be trimmed off with respect to the resource contributions made by each participating organization. In the existing literature of handling the free riding problem on peer-to-peer systems, the subject of contention has not been brought to attention because, unlike a consortium of participating organizations in a VO, peer-to-peer communities typically are expected to scale up from thousands to hundred-of-thousands of participants. In such scenarios, the problem of contention is not likely to occur unlike in our case where we consider a consortium of resource sharing parties.

The token-exchange incentive scheme is a well known incentive for promoting cooperation among participants in a peer-to-peer community. It has been applied to manage distributed computing resources [23, 25] and file sharing applications. In this research, we further demonstrate its versatility by showing that it can concurrently serve as a mechanism to administer admission control to alleviate resource overloading in a VO.

The token-exchange scheme operates as follows: each organization that contributes a set of resources is assigned a number of tokens that is proportional to their contribution of resources. Users can submit their requests through consumer agents that use a fraction of the tokens available to their respective organizations, in exchange for using shared resources. Provider agents are employed at each organization to determine the minimum amount of tokens in order for requests to gain access to their resources.

Intuitively, the incentive scheme has the ability to manage both inter-domain and user-initiated contention. Firstly, the assignment of tokens, which is proportional to an organization's contribution, restricts the extent that each organization can utilize the shared computing resources. Secondly, the tokens can be further shared among users so that their usage can also be controlled based on the amount of tokens that are allocated to their requests.

1.2 Thesis Objectives, Scope and Contributions

1.2.1 Objectives and Scope

With the aim of unifying the management of the above mentioned two sources of resource contention that is unique to a grid-based VO, this thesis answers the following question:

How do we extend the token-exchange scheme to perform admission control on computing resources that are contributed and shared by a group of participating organizations?

The objective of admission control to address user-initiated contention is to employ a strategy that improves the overall system admission ratio. This metric measures the ratio between the total number of successfully admitted requests and the total number of requests submitted by all users. To manage inter-domain contention, fairness is another metric which is introduced to quantitatively measure the degree of equity achieved by the

admission control system when participating organizations compete for shared resources. Hence, to answer the above question, we extend the token-exchange incentive scheme by incorporating it into an admission control mechanism that satisfies the objectives of the above metrics.

The foundational constituent of the token-exchange scheme is a policy for translating resource usage requirements of a request into an admission price that provider agents use to decide if a request should be admitted. Consumer agents that are responsible for submitting their user's requests have to pay the amount stipulated by the admission price. To coordinate the exchange of tokens, a broker is employed to match requests from consumer agents to candidate services published by provider agents. A uniform pricing framework is used by the provider agents to compute the admission price for each request. The organization whose candidate service is selected by the broker's matchmaking policy will be credited the amount based on the admission price generated by its provider agent.

The complex nature of a grid computing infrastructure requires the use of quality-of-service frameworks to handle users' requests in order to cope with the diversity of resource requirements specified by users. Consequently, there is a tight coupling between quality-of-service specification of users' requests and the admission policies. The pricing framework is therefore built to take into account the instantaneous contention of resources at each organization and the resources demands of the particular request that is to be admitted.

1.2.2 Main Contributions

The above objectives and research scope have led to the following contributions:

1. *A pricing framework to support the administration of advance reservation to reduce the negative effects caused by user-initiated contention.* We made use of a simulation model to explore the effectiveness of the pricing framework in supporting admission control against varying demand for shared computing resources due to the diversity

of CPU and execution time requirements of requests. The synthetic workloads were obtained from scaled supercomputer workload traces. From experiments, we demonstrated that the pricing framework facilitates in producing more consistent admission ratio performance for different degrees of contention in comparison with other admission control algorithms.

2. *A performance study to examine the parameters that are contributory to inter-domain level contention.* We employed a centralized scheme to demonstrate the criticality of the overall workload generated by each organization relative to its resource contribution (i.e., load-contribution ratio) in order to control the extent of resource contention. Resource contention is measured from a cumulative distribution function at which the size of the broker's queue exceeds a nominal value throughout the entire duration of the simulation. Resource contribution becomes more significant when either or both the degree of participation or the overall burstiness of workload traffic (or both) is increased. From the performance study, we found that the load-contribution ratio has a major impact on the degree of inter-domain contention. We further developed a centralized policing strategy to admit requests by controlling the size of the queue at the broker. From experimental results, we demonstrated the difficulties of using a centralized policy to reduce inter-domain contention.
3. *Integration of the pricing framework into the token-exchange incentive scheme to manage inter-domain contention.* To do so, we first defined fairness – a metric to judge the degree of equitable resource access achieved in the process of handling inter-domain contention amongst participating organizations. Based on the metric, we mathematically established the mandatory properties necessary for the system to attain the ideal measure for fairness. We also showed that, in a realistic system, not all the properties can be governed by policy administration. We further demonstrated that by combining a brokering policy and the pricing framework to

administrate admission control on requests submitted by users, fairness can substantially be guaranteed.

4. *An adaptive scheme for the pricing framework to improve the performance trade-off between fairness and the system admission ratio.* As mentioned, the goal of managing resource contention requires the system to achieve good overall system admission ratio while ensuring that fairness is maintained in order to reduce inter-domain contention. We experimentally revealed that the system admission ratio and fairness are performance trade-offs with respect to the initial amount of tokens assigned to each organization. Because the trade-off is an undesirable side effect of employing the token-exchange incentive scheme, we treated this problem by adapting the admission price in response to the instantaneous level of contention based on a set of accounting ratios.

The contributions in this thesis constitute a resource management strategy for on-demand computing in a VO when the supply of shared computing resources are insufficient to meet the overall resource needs of users' belonging to the VO. As such, it is important that the admission control framework is able to admit the workload of each organization proportionally according to their resource contribution so that the cooperative interests of each participating organization can be sustained.

1.3 Thesis Organization

The rest of the thesis is organized as follows:

In *chapter 2*, we introduce the conceptual system architecture and then define its simulation model used in the experiments in the following chapters.

In *chapter 3*, we give a literature review of the existing strategies that can be applied to manage both user-initiated and inter-domain contention on grid computing systems.

In *chapter 4*, we describe the key motivations, design and experiments of the pricing framework to manage user-initiated contention.

In *chapter 5*, we discuss the experiments and results on our preliminary analysis to assess the impact of the selected VO parameters on inter-domain contention on the shared resources.

In *chapter 6*, we develop a metric for fairness. It quantifies the degree of equity for resource access amongst participating organizations in the event of inter-domain contention. We show how the token-exchange incentive scheme, when augmented with the pricing framework, can be used to achieve a partial solution for fairness.

In *chapter 7*, we describe and experimentally demonstrate a solution for applying adaptive manipulation on the admission price to address the trade-off between fairness and the system admission ratio.

In *chapter 8*, we conclude the thesis by first giving a summary of the contributions and then explore some open questions that have resulted from the process of finding solutions to the problems addressed.

Chapter 2

System Architecture and Model Justifications

2.1 Introduction

The chapter mainly deals with the foundational concepts leading to the formulation of a conceptual resource management framework for a grid infrastructure considered in the analysis of policies for managing contention.

Since grid computing is still at its infancy, we develop the conceptual framework for resource management by giving an overview of how large-scale computing systems, otherwise known as metacomputers, have evolved and its associated paradigm shifts from traditional resource management methodologies. We then compare the conceptual framework with the key components of the existing metacomputing systems to justify its validity as a representative model of a grid computing infrastructure.

Resources contributed by multiple administrative domains suggest the need for resource administration mechanisms to facilitate the management of inter-domain access to the shared resources. This increases the complexity of resource management because additional mechanisms have to be incorporated to coordinate the process of assigning resources to users from different organizations and hence, having a major influence on the administration of policies to manage contention on shared resources. The additional

features investigated to support resource sharing on the grid are the quality-of-service and advance reservation mechanisms. Provisioning of quality-of-service is necessary to cater to the diverse needs of users that make use of resources linked to the grid. The quality-of-service mechanism facilitates the fulfillment of the service requirements of each request that has been successfully admitted by the system. Advance reservation is a complementary mechanism to the quality-of-service provisioning and it is introduced to provide in advance and monitor the quality-of-service specifications guaranteed to users or applications. This mechanism is required because applications may have to concurrently gain access to resources at multiple organizations.

We give a detailed description of the system model to elaborate on the various abstractions that make up the conceptual resource management framework. Since the implementation of resource management policies is largely influenced by the design of the resource management framework, this section expresses the scope of the experimental design and analysis carried out in the later chapters.

In the last section of this chapter, we describe the architecture of a simulator that is used to analyze the admission control policies designed in this thesis. The major components of the simulation prototype are built based on the proposed conceptual model of the resource management framework. Apart from describing the operations of the various modules of the simulator, its limitations are also reflected.

2.2 Characterization of a Conceptual Framework for Metacomputing Systems

Metacomputing systems generally require middleware resource management tools [53]. In short, the metacomputing system operates between users and the underlying managed computing resources that are likely to be geographically distributed. We attempt to generalize metacomputing resource management mechanisms through three distinct tiers as shown in Figure 2.1. At the bottommost layer, computing resources are mounted with

software that retrieve static (e.g., operating system and hardware characteristics) and dynamic information (e.g., CPU status, available memory and storage etc.) of computing clusters. The retrieved information is possibly filtered and aggregated before it is forwarded to an information service. Domain level resource managers perform a multitude of roles such as assigning, scheduling and allocating resources to requests. To perform these activities, they draw dynamic meta-data of their resources from the information service. Each resource manager also has built-in policies that define which of their underlying resources can be shared to the community of users.

At the top layer, request messages from users must conform to possibly a protocol or an application programming interface (API) to gain access to resources via the metacomputing middleware (i.e., the layer in the middle). The metacomputing layer performs the role of interpreting a request's resource requirements, and then finding the appropriate resources using a matching scheme that is either based on scheduling algorithms or a set of fixed policies. The matchmaking process is co-managed by the metacomputing middleware, together with the domain level resource managers. This process will be described in further detail at the later part of this chapter.

The above described metacomputing resource management process is coordinated by three major generic components – the consumer agent, the provider agent and the broker – one at each layer. The consumer agent takes charge of managing resource requests from applications or users. It is responsible for submitting request on behalf of its users within the same organization or administrative domain¹. The provider agent, which is an internal module of the domain level resource manager, acts as a proxy to generate service-level contracts to guarantee the execution of requests admitted by their respective domain-level resource manager. The broker's main role is to find appropriate resources from the available pool of resources declared by each resource manager. It also plays the role of matchmaking requests to the declared resources. To justify our claim on the

¹From this point onwards, the terms organization and administrative domain are used interchangeably

Figure 2.1: Conceptual Framework of a Metacomputing System

general resource management framework for metacomputing environments, we review the existing projects on metacomputing systems to establish the validity of the identified core conceptual components. Table 2.2 shows a list of prominent projects in the existing literature that fit in our conceptual resource management framework for metacomputing systems.

Table 2.1: Justification of Conceptual Framework from Existing Metacomputing Systems

System	Broker	Consumer Agent	Provider Agent
AppLes [32]	Resource selector filters and chooses different resource combinations.	Planner agent generates resource dependant schedules for an application.	Agents use static and dynamic application and system information to configure a set of resources.
Condor [11]	Collector agent connects a set of different computing resources to a virtual pool and matches resources to requests.	Clients submit their requests through customer agents.	Resource agent runs on each machine and receives dynamic resource information for resource management.
Netsolve [6]	Servers are employed to map software tools via remote procedure calls to execute computations submitted by clients.	Clients are written to interact with servers via TCP/IP.	Metadata of resources are generated by Netsolve Agents for matchmaking.
Legion [88]	Mappers provide the basic mechanism to make mapping decisions between objects and request specifications.	API is provided to clients to generate object requests and interaction.	Resources are virtualized into Legion objects. Each object is a task execution.
Globus [41]	Gatekeeper admits jobs to resources by means of security policies.	Client API is used to communicate with gatekeeper for job execution.	Globus Resource Allocation Manager (GRAM) is used for execution and monitoring of jobs
OurGrid [4]	Matchmaking function is performed by application agent called MyGrid to access shared resources.	Consumer agents is used to source for available resources.	Providers inform peers on resource availability.

2.3 Mandatory Mechanisms for a Grid Infrastructure

For the need of building larger applications, the initiation of grid computing seeks to further aggregate computing and data resources over multiple administrative domains. From a resource management perspective, there is a need to consider additional features that the grid infrastructure must provide in order to cope with the scale of resources. This is because these features may have an impact on the design of the resource sharing policies that are institutionalized at each administrative domain.

In this section, we describe the additional features that a grid must possess (in addition to the current metacomputing middleware tools) in order for them to be realistically deployed. Specific to our work, we consider two features that are critical to the design of the admission control schemes for managing contention:

- Quality of Service
- Advanced Reservation

A quality-of-service framework is introduced to provide dedicated resource needs to each admitted request submitted by users. In a sharing environment, the resource management system is subjected to resource usage conflicts amongst submitted requests that are to be serviced. An advance reservation is further required to guarantee the resource requirements of a request before it starts executing so that resource usage conflicts can be avoided when other requests are admitted later. The development of grid standards for the above two features is in the scope of the Grid Resource Allocation Agreement Protocol – a working group charter of the Global Grid Forum [86].

2.3.1 Quality-of-Service Support

Quality-of-service (QoS) management is originated from the area of flow and congestion control of packet-switched computer networks [38, 39] in recognition that packets from different applications may require different levels of throughput. For example, with the provisioning of video and voice over Internet protocols, QoS metrics have been used to address packet delay and application jitters of multi-media transmissions [50, 74].

The requirement for QoS management in the grid-based environment is because, firstly, resource needs are highly differentiated. Users who make use of computing resources require different number of CPUs and other secondary resource requirements (e.g., memory, processor speed and storage). Secondly, the presence of resource heterogeneity requires a standard specification to describe the QoS to be established for a request submitted by users. This is necessary so that the broker is able to make use of the information to find a suitable match according to the QoS requirements specified by the user.

Metacomputing systems typically provide either Soft or Hard QoS support (or both) [19]. A system that provides explicit QoS attributes for resource requests but cannot enforce service levels via policing provides soft QoS support. Hard QoS on the other hand is provided when all requestors can police the service levels guaranteed by the resource management system. Soft QoS is most commonly provided by a wide variety of applications whereas hard QoS guarantees are normally provided in areas of real-time support where deadlines become a critical issue for the application of the requestor [83].

There are numerous metacomputing systems that have QoS driven mechanisms integrated into the resource management framework. For example, [71] expresses a resource manager with application-level QoS specifications. Each application is a process, instrumented with code to communicate with a ‘coordinator’ that manages an application’s request for QoS-based resource usage. Sensors are used to collect, maintain and process a variety of metric information within the instrumented process. They obtain information

from the application through probes at strategic points of the instrumented code. The data obtained from the code are fed to the ‘coordinator’ for adaptive management of the application’s demand for QoS. [76] defines a multi-level service discovery mechanism using QoS. It consists of 3 general levels – Application, middleware and network. Service level agreements at each level are made using contract-based protocols which can be incorporated into web services by extending the Web Service Description Language (WSDL). QoS has also been introduced into advanced reservation algorithms to allow tasks to gain differentiated levels of usage limits based on their priority [37].

2.3.2 Advance Reservation

Advance reservation is a fundamental software mechanism to facilitate inter-domain QoS support to applications needing large amount of shared computing resources with hard QoS requirements. An advance reservation is a possibly limited or restricted delegation of a particular resource capability over a defined time interval, obtained by the requester from the resource owner [86]. Provisioning hard QoS is possible if the system can receive an immediate response on the success or failure of gaining the access to resources based on the QoS requirement and guarantee the demanded resource specifications of the request.

An advance reservation mechanism in the context of grid computing, is introduced to support resource co-allocation at multiple administrative domains [26]. Such mechanisms are needed because grid applications may involve several parallel subtasks that have to be executed at different administrative domains [3, 10]. When shared resources are not immediately available, the application may co-reserve resources for usage at a later time [73].

In many cases, a typical reservation request specifies the number of computing nodes desired, the maximum amount of time the nodes will be used, the desired start time and the maximum allowable slack time. These parameters form the basic structure of the QoS specification in a request generated by a user. Once execution is started, the job that is

generated for the request cannot be directly preempted by the resource manager. Slack is introduced to improve the flexibility of scheduling advanced reservations to give a wider scope of guarantee to the demanded QoS. Therefore, it reduces the scheduling constraints on advance reservations so that the underlying resources will be better utilized.

Executing an advance reservation is the process of matching demanded QoS from requests to appropriate shared resources. There are two parts to establishing QoS for an advance reservation request— admission control and policing [19]. Admission control determines whether or not the requested level of service can be given prior to accepting a request (that is, whether or not Service-level Agreement (SLA) can be reached) and policing ensures that the SLA is not violated. Once the SLA is achieved between the resource provider and the requestor, the selected resource that qualifies the SLA must be assigned to the requestor unless there is a cancellation of request (i.e., resource allocation timeout or simply that the requestor has completed using the resource). The SLA can be achieved in two ways - negotiating and non-negotiating. For the negotiating scheme, protocols are introduced to enable the resource requestor and provider to engage in interactions to reach an agreement level of service. In the non-negotiating scheme, the resource provider allocates the desired level of service only upon availability. If the desired QoS cannot be met, then the resource provider will either reject the request or find the closest match based on the requested QoS.

Advance reservations may be scheduled either first-come-first-serve (FCFS) or based on a priority scheme. In the former, advance reservations operate without any consideration to prioritize requests according to their QoS demands. In the priority-based scheme, additional flexibility to schedule advanced reservations is incorporated so as to either improve the utilization of shared resources or admission rate of requests [48, 62, 78]. A simple priority scheme is to divide requests into two queues. The soft QoS queue contains the requests that require a certain QoS requirement to be met but are able to compromise their execution deadlines in the event of resource contention. On the other hand, the hard QoS queue contains the requests that are only admitted if the scheduler

is able to assign the resources and thereafter, guarantees the desired QoS. The result of partitioning submitted requests into two queues increases the overall admission rate of requests at the expense of relaxing the QoS guarantees for some other requests [62]. In our work, we consider providing only hard QoS because an advance reservation requires strict resource guarantees upon admitting a request.

2.4 System Model

In this section, we introduce the system model to elaborate on the abstractions that are used in the following chapters for analysis.

2.4.1 The Grid Computing Environment

Definition 2.4.1 (The Grid Environment). The grid computing environment is modelled as $\mathbf{G} = (\mathbf{N}, \mathbf{R}, \mathbf{S})$. $N = \{n_0, n_1, \dots, n_m, \dots\}$ is a set consisting of shared computing nodes managed by the grid resource management system. Each node is assumed to be able to execute a single task at any one time. $R = \{r_0, r_1, \dots, r_i, \dots\}$ is an ordered set of requests submitted by users that belong to the VO. Each request contains the quality-of-service (QoS) specification by a single user. $S = \{s_0, s_1, \dots, s_j, \dots\}$ is a set of the service instances. Each service instance is a mapping between a request to a set of shared computing nodes in N . Therefore, each service instance has to be scheduled and it contains the quality-of-service information granted to each request.

Definition 2.4.2 (Organization). $U = \{u_o, u_1, \dots, u_k, \dots\}$ defines a set of organizations. An organization $u_k \in U$, uniquely owns a subset of requests $R_k \subseteq R$, a subset of service instances $S_k \subseteq S$ and a subset computing nodes $N_k \subseteq N$.

Property 2.4.1 (Request and Resource Ownership). For any two distinct organizations u_k and $u_{k'}$, $R_k \cap R_{k'}$, $S_k \cap S_{k'}$ and $N_k \cap N_{k'}$ equals $\emptyset$. This means that every object in the grid environment's space distinctively belongs to a single organization. This also implies that each user of the grid also belongs to only one organization.

Definition 2.4.3 (Request Parameters). A request specifies a job, and a job may contain a number of tasks.

The following parameters defining an arbitrary request $r_i \in R$ are as given:

- $PE(r_i)$ (Number of Computing Nodes)

Each computing node runs a single task of a request r_i . Each request may require one or more computing node(s). Requested nodes are assumed to reside within the same organization.

- $T(r_i)$ (Requested Execution Time Interval)

Defines the request's expected execution time duration for all tasks that execute on distinct computing nodes. That is, the time interval between the job starting time and the expected time of termination.

- $submit(r_i)$ (Request Submission Time)

Defines the time when a request reaches the *Broker Job Queue*.²

- $admit(r_i)$ (Request Admission Time)

Defines the time when a request is submitted to a *Provider Agent* from the *Broker Job Queue* for the creation of the corresponding service instance.

Definition 2.4.4 (Service Parameters). For each request, a service instance will be created to execute the job specified by the request.

A service $s_j \in S$ has the following parameters:

- $start(s_j)$ (Service Execution Starting Time)

Defines the actual execution starting time of the job after its corresponding request is admitted.

²In the following section, we formally introduce the various components of the conceptual resource management system.

- $T(s_j)$ (Granted Execution Time Interval)
Defines the total execution time interval allocated to the service by the advance reservation mechanism.
- $PE(s_j)$ (Assigned Computing Nodes)
Defines the total number of computing nodes assigned to the service by the advance reservation mechanism.
- $end(s_j)$ (Service Execution End Time)
Defines the expiration time of the service. This also means that the job under this service will be terminated once it the corresponding expiration time reaches $end(s_j) = start(s_j) + T(s_j)$.

Definition 2.4.5 (Resource Contribution). C_k defines the total number of computing nodes contributed to the community by organization k . The computing nodes are assumed to be connected together by means of a high speed network to form a single cluster.

2.4.2 Relational Mappings of System Entities

For the above grid environment, we define the following mappings for the various system entities using the following relational matrices:

- M^{RS} is a set of tuples. $M^{RS} = \{(r_i, s_j) \mid \text{service instance } s_j \text{ is created and scheduled for the execution of } r_i\}$
- Similarly, M^{SN} is also a set of tuples. $M^{SN} = \{(n_m, s_j) \mid \text{computing node } n_m \text{ is assigned to service } s_j\}$. Note that, multiple computing nodes may be assigned to the same service instance.

For the above mappings, the following definitions and properties exist:

Property 2.4.2 (Request-Service Constraint). $\forall j \ s_j \in S, \exists r_i \in R$ such that $(r_i, s_j) \in M^{RS}$. Given that $r_{i'} \in R$ and $r_{i'} \neq r_i, (r_{i'}, s_j) \notin M^{RS}$. Also, given that $s_{j'} \in S$ and $s_{j'} \neq s_j, (r_i, s_{j'}) \notin M^{RS}$. This property suggests that each scheduled service instance is mapped to exactly one request and vice versa.

Property 2.4.3 (Service-Resource Constraint). $\forall j \ s_j \in S, \exists n_m \in N$ such that $(s_j, n_m) \in M^{SN}$. Also, given that $S_k \subseteq S$, if $s_j \in S_k$, then $\forall p, (s_j, n_p) \in M^{SN}, n_p \in N_k$. Each service must be mapped to at least one computing node. Also, for every service that is scheduled, its corresponding computing nodes must all belong to the same organization.

2.4.3 Quality-of-Service Parameters

Definition 2.4.6 (Maximum Allowable Slack Time). The allowable slack time, T_i^{aslk} for request r_i , is the maximum allowable time interval between the latest possible starting time of a service corresponding to the request and the admission time ($admit(r_i)$) of the request.

Definition 2.4.7 (Maximum Allowable Slack Ratio). The maximum allowable slack ratio $\alpha(r_i)$, is defined as the ratio between the maximum allowable slack time T_i^{aslk} and $T(r_i)$. Note that, $\alpha(r_i) \leq 0$ and we assume that it is less than or equal to 1.

Definition 2.4.8 (Slack Time). Slack time $T_{i,j}^{slk}$, is defined as the delay time between the admission of a request r_i and the actual starting time of a job of the corresponding service instance s_j . Therefore, $T_{i,j}^{slk} = start(s_j) - admit(r_i)$. Note that $T_{i,j}^{slk} \leq T_i^{aslk}$.

Definition 2.4.9 (Slack Ratio). The slack ratio $\alpha(s_j)$, is defined as the ratio between the allowable slack time $T_{i,j}^{slk}$ and $T(s_j)$, the execution time interval of the service assigned to request r_i .

Definition 2.4.10 (Requested Quality-of-Service). It is a vector defining the resource specification obtained from a request. The requested quality-of-service from r_i is $QoS_i^r = \langle PE(r_i), T(r_i), \alpha(r_i) \rangle_i$.

Definition 2.4.11 (Granted Quality-of-Service). The granted quality-of-service is defined after a service instance is scheduled. Therefore, given $(r_i, s_j) \in M^{RS}$, the quality-of-service of s_j granted to request r_i , is $QoS_j^s = \langle PE(s_j), T(s_j), \alpha(s_j) \rangle_j$.

Property 2.4.4. To satisfy the quality-of-service requirements for advance reservation, for any $(r_i, s_j) \in M^{RS}$, $QoS_i^r = \langle PE(r_i), T(r_i), \alpha(r_i) \rangle_i$ and $QoS_j^s = \langle PE(s_j), T(s_j), \alpha(s_j) \rangle_j$, the following conditions hold:

1. $PE(r_i) = PE(s_j)$
2. $T(r_i) = T(s_j)$
3. $\alpha(r_i) \geq \alpha(s_j)$

In (1), as long as the system is unable to allocate the specified number of computing nodes, the service cannot be instantiated and scheduled. In (2), the total execution time granted to a request is equal to the execution time interval specified in the request. This is a necessary condition for advance reservation. If the job does not complete its execution within the allotted time interval, such that its execution time intersects into the schedule of another service, the system may notify the application prior to terminating the application. This problem can be addressed by re-scheduling the application to another set of computing nodes. (3) shows that the granted QoS must also guarantee that the maximum allowable slack time is not exceeded.

2.5 Simulation Prototype

A discrete event simulator is built to implement the conceptual model of the resource management framework for a grid computing infrastructure to analyze the policies investigated to address both user-initiated and inter-domain contention. It is built to contain the mandatory middleware software components such as the broker, consumer and provider agents mentioned in section 2.2.

Figure 2.2: Architecture of the Prototyped System

The general advantage of employing a simulator as opposed to using a testbed is that the experiments to evaluate the policies for managing resource contention have to be conducted repeatedly, in a controlled environment. Furthermore, it is not practical to set up a grid having the same number of computing nodes to reflect the conceptual model of the grid computing environment. However, we have taken steps to cross-validate the design of the prototype with a real system. The specific differences will be explored in section 2.5.2.

We make use the simulator to analyze the impact of policies for treating both user and inter-domain level contention. In chapter 4, we develop a set of policies to perform admission control on requests with the aim of addressing user-initiated contention. The policies consist of a matchmaking policy implemented at the broker, and a pricing framework that is incorporated into the advance reservation mechanism of each organization's provider agent. In chapter 7, we make use of the simulator to evaluate the use of an adaptive policy that is built into the pricing framework to improve the design of the token-exchange incentive scheme.

2.5.1 Overview

An overview of the simulator's architecture is given in Figure 2.2. The simulator has two main subsystems - the *Broker* and the *Domain-level Resource Manager* for each distinct organization. Since they are both independent components, communication between the two components are based on predefined message protocols to coordinate the admission control of requests. Each *Consumer Agent* is contained in a *Domain-level Resource Manager* instead of existing as a separate component as depicted earlier in the conceptual framework for a generic metacomputing system in Figure 2.1. This is because the VO is a cooperative model where each organization is both a resource consumer and a provider.

Requests submitted by users within the same organization arrive at the *Request Queue*. Each organization has its own *Consumer Agent* that handles requests from users based

on a FIFO policy. The simulator generates requests from predefined workload traces. Each request is assumed to come from a single user. It contains information regarding the quality of service parameters (i.e., CPUs used and the execution time supplied by a single user) together with a timestamp to indicate the arrival time of a request.

The protocol for admitting a request is based on the following steps:

1. The *Consumer Agent* directs requests to the *Broker Job Queue* by submitting each request together with a bid in order to gain access to shared resources. It does this in two steps. First, it performs an internal admission check prior to submitting a request to the broker. Since a token-exchange scheme is employed³, the consumer agent has to first determine if it has adequate expendable tokens for the current request. It will employ an internal sub-module to compute the amount of tokens (price) to be used for the current request.
2. The request is then submitted to the *Broker Job Queue* until it is serviced.
3. The *Dispatcher* module will service each request in the *Broker Job Queue* using a FIFO policy. To serve each request, the *Dispatcher* will propagate the request to every *Provider* agent.
4. The *Provider Agent* determines the admissibility of the request. If sufficient resources can be allocated to the request, the *Provider Agent* will generate an admission price and create a candidate service instance for the request. Along with the admission price, each *Provider Agent* will submit a candidate service instance to the *Matchmaker*.
5. The *Matchmaker* in turn, selects one from the set of the candidate services and matches it to the request. The criterion for a successful transaction requires the

³In chapter 6 we will formally introduce the use of the token-exchange incentive scheme for resource management.

request price to be at least the same or greater than the admission price for the corresponding candidate service.

6. The respective *Provider* agent of the selected candidate service will be admitted and informed and the service instance will be transferred into the service queue. The *Scheduler* is responsible for scheduling the service instance in order to fulfill the guaranteed quality-of-service requirements of the corresponding request. Once the service instance has been scheduled, computing nodes are assigned to the service instance and a job execution will be started according to the schedule.
7. The provider agents of the unselected candidates service will discard the service instances.

2.5.2 Job Management

In the prototype system, for requests that are successfully admitted, their corresponding service instances will be processed by the *Scheduler*. The *Scheduler* is the sole component that allocates computing nodes to a request and monitors the execution of the job belonging to the request.

To meet the quality-of-service requirements of each scheduled service instance, the *Scheduler* has to repeatedly scan the service queue to track the assigned starting time of all scheduled services. When the current time reaches the prescribed starting time of the service, the *Scheduler* will start the execution of the job on the computing nodes that had pre-assigned to the service instance.

Because the *Provider Agent* is responsible for admitting a request and assigning the actual starting time to the corresponding service instance, it therefore coordinates with the *Dispatcher* to ensure that the prescribed quality-of-service of request can be fulfilled prior to its admission. This is achieved by the use of a reservation window. The reservation window is a chart consisting of the total resource capacity managed by the *Domain-level Resource Manager* on the y-axis and simulation time on the x-axis as shown

Figure 2.3: Illustration of Advance Reservation Window

in Figure 2.3. When a request is guaranteed admission, its corresponding service instance will be placed in the reservation window and scheduled for execution by the *scheduler*. The reservation window is used by the *Provider Agent* to estimate the current utilization of its resources. It is also used to determine if there are sufficient computing nodes for a newly arrived request.

The job management mechanisms of the simulator can be used directly in a physical resource management system, if we replace the simulated version of the *Scheduler* with a module that communicates with a cluster management system for job submission [8]. Although the simulator reflects the design of a real system, we noted one practical problem, which is the time skew between the advance reservation schedule and the actual job schedule. When the *Scheduler* submits a job to its computing nodes, it has to be further scheduled by a cluster management system. As such, there will be some delay between the time when the job enters the cluster queueing system to the time it starts

running on an computing node. Adyt [8] explored the issue of time skew in both capacity and time shared configurations. The experiments demonstrated a significant difference between the results from simulation and actual execution on a cluster of heterogeneous computing nodes for both job starting time and execution time. Another related problem arises when a job subsequently exceeds its estimated execution time as a result of the delay. Jobs that are scheduled to execute after the current job on the same computing node will also be delayed, and therefore, worsening the time skew between the provider's resource allocation schedule and the actual execution schedule.

The above problems can be corrected in a number of ways. First, the system may reserve additional computing nodes to act as buffers to execute the delayed job. Another way to address this problem is to estimate the probability that the quality-of-service requirements of the request can be met. The latter approach is more complex because it requires prediction techniques to estimate the time skew of a service that is to be executed later on. Predictions are typically made from a priori information obtained from executed jobs. In a grid, a variety of applications and platforms are supported. This implies the need for large amount of information in order for accurate predictions to be made.

The work in this thesis does not account for the time skew between the allocation schedule and the actual execution schedule. This is because the size of the reserved pool of resources is strictly an internal administrative issue within the organization. Furthermore, generating probabilistic values for quality-of-service parameters require more sophisticated representation models for predicting the occurrence of skew within the execution schedule.

2.6 Summary

In this chapter, we describe the evolution of metacomputing systems to derive a conceptual framework for a grid computing infrastructure. We cross justify the framework with existing implementations of metacomputing systems. Since the grid infrastructure is a

shared environment where resources are used by a diverse group of users, advance reservation and quality-of-service support must be provided. Advance reservation are employed to guarantee quality-of-service requirements to applications or users during contention for shared resources. Based on the requirements analysis, we express the details of the system model that is applied in the experiments conducted in this thesis. Finally, a prototype is described to examine how the conceptual framework operates as both a simulator and a system.

Chapter 3

Existing Approaches for Managing Contention on Shared Resources

3.1 Introduction

The VO assumes a cooperative model for resource sharing. This is because organizations have to depend on each other's contribution for the pool of resources that is to be shared amongst users in the VO.

The physical system that supports the VO is a grid infrastructure. With respect to the conceptual framework presented in chapter 2, users have to submit their requests to a broker which in turn, forwards the request to domain-level resource managers to be processed. Since requests have different quality-of-service requirements, user-initiated contention may occur if any user tries to use resources to such an extent that the execution of the requests generated from the user deprives other users from gaining access to the shared resources. One way to address this problem is by means of admission control by taking into account the collective resource needs of all users who are competing for the shared resources. To do this, valuation functions are introduced to allow users to flexibly relate their resource quality-of-service requirements to a single quantitative value. The value reflects either the degree of satisfaction gained or the price that the user is willing to pay for a given quality-of-service specification. The grid resource management

system makes use of the valuation functions of each competing user to allocate the shared resources to each user by maximizing the aggregate valuation of all the users.

The need to employ valuations functions implies the use of an economy-based framework for resource management in a grid computing environment. In a generic economy-based framework, a group of agents are employed to trade resources by means of a payment mechanism. With respect to the simulation prototype described in chapter 2, consumer agents determine, by means of a valuation function, the price to pay in order to have their requests admitted by the resource management system. The broker and a group of provider agents will determine the admission price in order for the request to be admitted.

However, the economy-based framework must not only be used to address user-initiated contention, it must also be adapted to meet the requirements necessary for managing inter-domain contention. Inter-domain contention manifests if organizations do not contribute sufficient resources relative to their workloads. In a perverse situation, that is, if all organization's workloads require more resources than their contributions, this problem must be treated, again, by means of admission control to limit amount of admitted requests of each organization to be proportional to their contributions.

The intention of this chapter is to survey a group of resource management schemes that are applicable to a grid infrastructure and are potentially suited to manage either one or both sources of resource contention. The aim of this survey is to find a suitable scheme that can fulfill the requirements necessary to manage both user-initiated and inter-domain contention. The diagram in Figure 3.1 depicts a taxonomy of contention management schemes that are employed in shared systems in diverse areas of application. The various schemes studied are broadly partitioned into two distinct groups that are capable of addressing either user-initiated or inter-domain level contention. Candidates that are suited for managing user-initiated contention embody the characteristics of an economy-based framework because these schemes make use of valuations functions to allocate

resources to a set of agents. The schemes classified under inter-domain contention on the other hand, are capable of achieving the requirements for managing inter-domain contention as will be described in the next section.

Figure 3.1: Taxonomy of Existing Schemes for Managing Contention on Shared Resources

Our aim is also to show that the token-exchange incentive scheme is uniquely suited for managing both user-initiated and inter-domain contention. The token exchange scheme operates on a principle where a fixed amount of tokens that is assigned to an organization is proportional to its corresponding resource contribution. In order for a user to gain access to another organization's resource, its consumer agent must pay an amount of tokens to the organization executing the request of the user. To address user-initiated contention, valuation functions can be used to translate the quality-of-service requirements of each user to the number of tokens to be used for a request. The provider agent of each organization can then use their own valuation functions to compute the admission price for the request. the request will only be admitted if the admission price is less or equal to the number of tokens that the requesting organization is willing to pay. Using

the token-exchange incentive scheme, inter-domain contention can be addressed because now, each organization will autonomically limit its use of other organizations resource with respect to the balance of tokens held.

The token-exchange incentive scheme, however, still lacks certain critical features necessary for it to support advance reservations and the quality-of-service provisioning requirements as prescribed by the conceptual model of a grid infrastructure as described in the previous chapter. The extended features for the token-exchange incentive scheme will be cited in the last section. In the following sections, we will describe the operating characteristics of different schemes shown in the taxonomy in Figure 3.1.

3.2 Requirements for Contention Management on Shared Resources

In order to critically analyze the suitability of the existing schemes for our problem domain, we first identify assumptions and requirements for managing both user-initiated and inter-domain level contention.

The basic requirements for managing user-initiated contention is the need to employ valuation functions for the grid resource management system to allocate resources to users, taking into account their quality-of-service requirements:

- Users have the autonomy to specify their own quality-of-service (QoS) specifications. Q is a vector containing a set of quality-of-service parameters consisting of a vector of real numbers. Therefore, $Q \in \mathbb{R}^n$, where $0 < n < \infty$, denotes the QoS specification of a request from a user. (Note that Q is a general notation to represent QoS_i^r , the quality of service specification of r_i).
- Each user has a valuation function $v : \mathbb{R}^n \rightarrow \mathbb{R}$, where $0 < n < \infty$. $v(Q)$ takes in the quality-of-service specification Q of the user. The valuation $p = v(Q)$ returns a value that can be used in several ways by the resource management system to

allocate resources to competing users in the event of contention.

- $\mu_k^i : \mathbb{R}^n \rightarrow [0, 1]$ is the utility derived by the user from organization k when its request r_i is successfully executed. μ_k^i returns 1 if the request r_i is successfully executed, provided that the quality-of-service requirements QoS_i^r , is fulfilled. Otherwise, μ_k^i returns 0.
- The system objective in managing user-initiated contention is to maximize μ^{tot} , the total utility gained by all users. $\mu^{tot} = \sum \mu_k$, such that $\mu_k = \sum \mu_k^i$.

Inter-domain contention management has in addition the following requirements.

- For each administrative domain (organization) k , the ratio between the total utility gained by an organization and its contribution must be kept below a certain threshold. By definition, $\frac{\mu_k}{C_k} \leq \Upsilon_{opt}$, where $\mu_k = \sum_i \mu_k^i$.

Since inter-domain contention is caused by every organization's interest to maximize the utility of their users during contention, the threshold Υ_{opt} is used to limit the total number of admitted requests belonging to an organization. We do not define how Υ_{opt} is to be computed at this stage.

The goal of inter-domain contention management is to reduce resource contention caused by “free-riding” through enforcement of fairness (that is, to make sure that all organizations use the shared resources according to their contributions).

The above assumptions and requirements demonstrate the critical differences for managing both user-initiated and inter-domain contention. There is no existing scheme that concretely provides a solution to concurrently address both user-initiated and inter-domain contention. As such, the various schemes that are discussed will focus on either one of the two sources of resource contention.

3.3 User-initiated Contention Management Schemes

We study two distinct approaches for managing user-initiated contention: the Differentiated Services and the Auction. Both schemes exploit the use of economic models to allocate resources to users. However, the differentiated-service is a simpler approach than the auction because it does not involve the use of elaborate negotiation mechanisms to allocate resources to users. An auction on the other hand, employs a negotiation mechanisms to reach an equilibrium on how resources should be shared amongst competing users. This is necessary for users to adaptively review their quality-of-service requirements and valuations with respect to the level of competition.

3.3.1 Differentiated Services

The differentiated services approach is a broad concept and therefore, the mechanisms that are designed on the basis of this approach are also purposed to fulfill very different system-level objectives. Differentiated service has traditionally been used to regulate internet traffic on web content servers that operate on clusters of workstations to serve a large group of geographically disperse users [38, 39, 64, 93].

A classical example for admitting differentiated services is the Paris Metro Pricing scheme that has been widely applied in the area of network packet routing. The Paris Metro Pricing scheme differentiates requests with different quality-of-service requirements by assigning a different priority to each request by means of a valuation function. Shared resources (bandwidth) are partitioned into distinct service pools, each of which admits requests of a certain priority range. The priority ranges of service pools do not overlap. This approach allows requests with the same priority to be passed to the same service pools so that they receive the same quality-of-service guarantees. By employing a priority scheme to determine the service guarantee assigned to each request, the differentiated services is essentially a simple economy-based resource management approach. A valuation function is employed at each network broker to determine the priority of requests

(packets) before routing packets to their destination nodes. The valuation is performed using a common policy that is embedded in the network protocols for packet routing.

In the Paris Metro Pricing scheme, the valuation functions are not defined by users. An agent is responsible for assigning the priority to submitted requests based on a single valuation function that is used for all requests. All requests are admitted to their respective service pools on a first-come-first-serve policy. However, this scheme does not impose a upper limit on the total number of requests that can be admitted to each service pool. As such, the quality-of-service guarantee to each admitted request will fluctuate with respect to the total number of request admitted to their corresponding service pool for a given time. In this way, the Paris Metro Pricing scheme can only provide soft quality-of-service support to each admitted request.

Another class of differentiated-service schemes are those that dynamically partition and allocate the shared resources to users according to their valuation functions. Utility or value based scheduling techniques are a broad class of dynamic resource allocation algorithms that fit this category of differentiated-service schemes. These schedulers generally rely on user-defined valuation functions to find an allocation that maximizes the overall utility (or alternatively, the aggregate valuation) of all users competing for the shared resources [17].

To do so, a resource broker employs a computational algorithm to partition the available shared resources and allocate each partition to users according to their quality-of-service demands and valuations. This technique differs from the Paris Metro pricing framework in terms of resource allocation because it relies on a dynamic partitioning strategy to meet the hard quality-of-service specifications of users by means of their individual valuation functions.

Such schedulers have traditionally been employed to cope with sudden increments in arrivals of requests with real-time execution requirements. For example, in the area of real-time database systems [43], a scheduler ranks submitted queries in ascending order,

based on their estimated time of execution. The queries are submitted together with their corresponding valuation functions to determine which queries should be prioritized and therefore, be scheduled for execution.

Utility or valuation functions are also used to resolve resource access conflicts amongst users. Both Bondavalli et al. [14] and Irwin et al. [27] employed utility functions to perform admission control on submitted requests to cope with resource overload conditions caused by sharp increases in job arrivals. Similar to the case of real-time databases, requests are ranked in ascending order based on the number of computing nodes required. The valuation functions are then used to select requests for admission in order to maximize the aggregate valuation of all competing users.

3.3.2 Auction

The public grace given to service-oriented grids motivates the metaphor that a computer is viewed upon as an economic utility [34, 59, 85]. As such, the management of distributed storage and computing resource has lately been associated with ‘marketplaces’ where resource allocation decisions are based on the interactions between consumer and provider agents by means of a standard protocol [36]. The auction is a resource management mechanism that allows multiple consumer agents to bid for resources through a broker that manages and allocates the shared resources to users.

In an auction-based system, the available resources are partitioned by finding a unique allocation and price for each consumer agent on the basis of their bids. A simple example of an auction mechanism is the sealed-bid auction. It is a single shot auction, meaning that, the auctioneer (broker) receives a single submission from consumer agents. Each request (or bid) contains the valuation function specific to each consumer agent. The auctioneer will then solve a computational problem, taking into account each valuation function, to obtain a unique price and allocation for each consumer agent. The auctioneer is therefore capable of managing user-initiated contention by deciding how (possibly

limited) resources are allotted to each consumer agent according to their valuations to maximize the total utility gained by all the consumers valuations.

We establish a general model, in order to illustrate the operating mechanism of a single shot sealed-bid auction. Let $B = \{1, \dots, i, \dots, m\}$ be a set of consumer agents or buyers and $A = \{1, \dots, j, \dots, n\}$ be a set of resources to be shared to a group of consumer agents. Let $\Omega = \mathcal{P}(A)$ be the power set of A , which contains all the possible combinations of resources (bundles) in A that can possibly be allocated to each consumer agent. Note that, $\emptyset \in \Omega$. Let Q represent the set of resource quality-of-service specifications (or bundles) that a consumer agent may specify. Note that $Q \subseteq A$ (and $Q \subset \Omega$). Then, $v_i(Q)$ is the valuation function that maps the quality-of-service as defined by consumer agent i to a single real number. Next, let $X = \{X_1, \dots, X_m\}$ define a set of feasible allocations of A to all consumer agents, and $\mathbb{X}$ is a global set of feasible allocations. There is a possibility that $X_i = \emptyset$, indicating that a consumer agent may not receive any resource for a given feasible allocation X . $\forall i, j$ where $i \neq j$, $X_i \cap X_j = \emptyset$, means that consumer agents do not share any resource. The auctioneer allocates resources to consumer agents by solving a linear program involving two objective functions $V_1(B)$ and $V_2(B)$ [66].

$V_1(B)$ tries to find a feasible allocation that maximizes the aggregate valuation of all agents based on the following equation:

$$V_1(B) = \max_{X \in \mathbb{X}} \sum_{i \in B} \sum_{X_j \in X} v_i(X_j) y_i(X_j) \quad (3.3.1)$$

subject to the following conditions

$$\sum_{X_j \in X} y_i(X_j) = 1$$

$$\sum_{X \in \mathbb{X}} z(X) = 1$$

$y_i(X_j)$ returns 1 if agent i is allocated resource specification X_j and it returns zero otherwise. The first condition mandates that any consumer agent is allotted exactly one bundle. $z(X)$ returns 1 if X is the chosen feasible allocation for the solution to $V_1(B)$, and 0 if otherwise. The second condition means that only one feasible allocation is chosen.

After the best feasible allocation is found, the next step is to calculate the clearing price for resources to be assigned to each consumer agents. This is necessary, because in many cases, there may be several feasible allocations that satisfies $V_1(B)$. As such, the dual $V_2(B)$ is used to find a price vector $p = [p_1, \dots, p_i, \dots, p_m]$ that minimizes the sum of π^s , the maximal payoff to the auctioneer and π_i , the maximal payoff to the buyer. The maximum payoff to the auctioneer is the maximum earnings gained from assigning a feasible allocation X to consumer agents. The maximal payoff to each consumer agent is the bundle that yields the largest difference between the agent's valuation and the assigned price. $V_2(B)$ is calculated based on the following equation:

$$V_2(B) = \min_{\pi, \pi^s, p} \left(\pi^s + \sum_{i \in B} \pi_i \right) \quad (3.3.2)$$

subject to

$$\pi^s \geq \sum_{i \in B} p_i(X_i) \quad \forall X \in \mathbb{X}$$

$$\pi_i \geq v_i(X_i) - p_i(X_i) \quad \forall i \in B, X_i \in X$$

The final solution, also known as competitive equilibrium, is obtained from the feasible allocations derived from both $V_1(B)$ and $V_2(B)$. To find the competitive equilibrium, we further define the demand set of consumer agent i with price p_i . The demand set contains a group of resource specifications from Ω that maximizes the payoff of a consumer agent.

$$D_i = \{ Q \in \Omega | v_i(Q) - p_i(Q) \geq v_i(T) - p_i(T) \quad , \forall T \in \Omega \} \quad (3.3.3)$$

In addition, a supply set ($L(B)$) for all consumer agents is also defined. The supply set returns a group of feasible allocations that maximize the earnings given price vector p .

$$L(B) = \left\{ X \in \mathbb{X} \mid \sum_{i \in B} p_i(X_i) \geq \sum_{i \in B} p_i(Y_i) \quad , \forall Y \in \mathbb{X} \right\} \quad (3.3.4)$$

The feasible allocation for competitive equilibrium, $X^* = \{X_1^*, \dots, X_m^*\}$, is chosen if it is obtained from $V_1(B)$, $V_2(B)$ and satisfies the following conditions.

- For every consumer agent $i \in B$, $X_i^* \in D_i$
- $X^* \in L(B)$

The mechanism that finds a feasible allocation based on this model is the Vickery-Clark-Groves (VCG) algorithm [67]. However, this model is limited because it only involves a set of homogeneous resources that are to be allocated to a group of consumer agents. In a grid computing environment, consumer agents require more than one type of resource. The quality-of-service specifications for grid resources often involve a combination of parameters (e.g., CPUs, execution time, and memory usage). Specific to our system model, each request requires not only a set of CPUs, but also different execution time intervals and starting time delay constraints.

A solution to the requirement for multiple resource types is to employ the iterative VCG mechanism. Instead of a single submission, valuations are repeatedly sought from consumer agents until a feasible solution for competitive equilibrium is attained. The primal-dual iterative linear program proposed by Parkes et al. implements this strategy [66]. It is an optimization algorithm designed to solve a constraint satisfaction problem. The optimization process is carried out iteratively between the consumer agents and the auctioneer. The algorithm further employs a condition that is called complementary slackness, to establish relaxations on the conditions necessary for competitive equilibrium.

This is to allow the iterative algorithm to terminate in the event that a feasible allocation for competitive equilibrium cannot be found within a pre-assigned time constrain.

A basic description of the algorithm is depicted in Figure 3.2. At step 1, it computes the initial price based on the dual solution $V_2(B)$. At step 2, the algorithm receives bids for consumer agents containing their valuations and demanded quality-of-service. The auctioneer then computes a locally feasible allocation X based on $V_1(B)$ at step 3. The algorithm then enters a conditional check to evaluate if the allocation and price outcome satisfies the constrains set by the complementary slackness criteria (step 4). If it fails, the price vector will be adjusted while receiving a second set of bids from consumer agents (step 5). This cycle repeats until the algorithm terminates when the slackness criteria is fulfilled.

Figure 3.2: Algorithm for the Iterative Combinatorial Auction based Primal-Dual Theory [66]

An auction is an elegant mechanism for resource management in the event of user-initiated contention because firstly, consumer agents (on behalf of their users) can independently express their resource requirements. Secondly, efficient mechanisms are employed to optimize the allocation of resources based on aggregate valuations of all

consumer agents. This suggests that the auction mechanism is capable of maximizing the aggregate utility gained by all consumer agents.

However, the lack of scalability is a major drawback of the auction mechanism. This is because it requires an auctioneer (central broker) to find a feasible allocation for all consumer agents based on the underlying shared resources it manages. Since consumer agents are likely to have multiple resource requirements, the mechanism requires consumer agents to repeatedly negotiate their quality-of-service requirements by repeatedly updating the valuation functions so that a near-optimal allocation can be found. This may not be feasible under certain circumstances especially if a request must be serviced almost immediately (e.g., if jobs submitted to the grid have to meet critical deadlines). Furthermore, the objective of an auction mechanism is not designed to be compatible with the requirements for managing inter-domain contention, even though it is suited for managing user-initiated contention.

There are many different classes of auction mechanisms that have been applied to address distributed resource management problems in diverse areas of application. They may not necessarily adhere to the model of the single-shot, sealed bid auction as described above. Spawn [31] uses an auction to automate the allocation of resources, but the price of a resource is formulated through a fixed pricing policy for admission control of jobs. D'Agents [15] is an agent-based architecture that simulates a market where agents have to plan their usage of computational resources according to their own budget. The D'Agents architecture has sophisticated cost-benefit models (similar to valuation functions) that enable each agent to decide on a combination of resources (e.g., memory, storage, and CPU speed) according to their budget and utilities.

3.4 Inter-domain Contention Management Schemes

One motivation for resource sharing is to allow organizations to benefit from each other so that they do not need to purchase computing resources to cope with peak demand

situations. In addition, the cooperative can also serve as a venue for organizations to help those with less computing resources. This is evidenced by current efforts to develop technologies for collaboratories or worldwide virtual laboratories where researchers at different geographical locations can concurrently interact by sharing data and computing resources, as if they were in a single laboratory [70, 90].

As explained in chapter 1, in a situation where the aggregate workload submitted by users in each organization exceed the resource contributions made by their organization inter-domain contention may happen. There are a number of approaches that are suitable candidates for managing inter-domain contention. We consider two classes of approaches: centralized grid-wide schedulers and incentives schemes that are widely studied in the area of peer-to-peer file sharing systems.

3.4.1 Grid-wide Schedulers

In this section, we discuss schedulers that sufficiently deal with inter-domain contention at the grid infrastructure level. These schedulers fall into the area of multi-site scheduling, where central brokers are employed to assign jobs to domain-level resources [16, 21, 29] from requests submitted by users. Site or domain-level schedulers are federated to a higher level grid broker. Each domain-level scheduler manages a group of computing resources that either belong to an organization or within a prescribed geographical area. The higher level grid broker has two general roles. First, it serves as an entry point to receive requests submitted by users. Second, it operates as a high level scheduler to forward requests directly to the domain-level schedulers based on some predefined policy. In addition, the grid broker may also take into consideration the quality-of-service requirements of a request when assigning it to any domain-level schedulers [77]. Each domain-level scheduler may need a single or group of cluster management systems such as Sun GridEngine [81], Condor [11] and LSF [61] to coordinate the execution of jobs based on the requests submitted by users.

A common technique for managing contention is to employ the round-robin scheme for selecting sites to schedule jobs. It distributes the workloads submitted by users to be equally managed by each domain-level scheduler. However, this technique alone does not meet the requirements necessary for managing inter-domain contention.

An additional policy is required for admission control of requests submitted to the grid broker. The policy first measures the ratio between the total number of admitted jobs belonging to an organization and the total number of successfully executed jobs in the same organization's resources within a specified time range. To further illustrate, let L_k denote the total number of admitted jobs from organization k . Also, let C_k be the total number of jobs that run on resources managed by organization k . The ratio $\frac{L_k}{C_k}$ is compared with a predefined threshold Υ_{opt} . If $\frac{L_k}{C_k} \leq \Upsilon_{opt}$, then the request will be accepted. Given that, μ_k is proportional to L_k , we know that $\frac{\mu_k}{C_k} \leq \Upsilon_{opt}$. Enforcing admission control using this scheme ensures that the requirement for addressing inter-domain contention is fulfilled. Once the request is accepted, it will then be forwarded to the appropriate domain-level scheduler based on the round robin policy.

In the event that the demand for shared resources for all organizations exceed beyond the capacity their contribution can handle, the above admission control scheme is able to achieve: for any two organizations k and j , $\frac{\mu_k}{C_k} = \frac{\mu_j}{C_j}$. This is because in such a situation, the ratio $\frac{\mu_k}{C_k}$ is equal to Υ_{opt} for any organization k . We will explore the centralized scheme for admission control in chapter 5.

Grid-wide scheduling is a straightforward means of managing inter-domain contention because a single decision maker (i.e., broker) administers the admission of requests submitted by users. However, since the broker does not coordinate with domain-level schedulers to admit a request, the system therefore cannot ensure that the quality-of-service requirements of each admitted request will be met. Moreover, since the overall quality-of-service requirements of a group of users are likely to change over time, more sophisticated techniques must be employed to measure the ratio of L_k and C_k . In chapter 5, we fur-

ther investigate the additional mechanisms required for admission control using a central broker to take into account changes in the statistical properties of the overall workload of users.

Another disadvantage with this approach (which we will reiterate in chapter 5) is the need for an accounting mechanism to calculate both L_k and C_k for all organizations. Scalability is a major issue because the amount of accounting information increases with the number of participating organizations.

3.4.2 Incentives

Incentive schemes have been widely studied in the area of peer-to-peer (P2P) file sharing systems as a means to reduce free riding amongst participants (peers). Peer-to-peer systems are popular to file sharing Internet applications that rely on ad-hoc networks [5] to manage a distributed pool of data contributed by participants. Some examples of peer-to-peer file sharing applications are e-Donkey [28], Kaaza [52] and Napster [63]. Peer-to-peer sharing systems are also ubiquitous to computing resources [51, 72]. There are a wide variety of peer-to-peer systems but they are largely similar in function of services except for minor differences in their architecture or built-in tools to support automatic sharing.

The peer-to-peer sharing model has gained considerable attention in grid computing [35]. Despite the advantages of cooperative-based sharing, a peer-to-peer system in general, is vulnerable to free-riding due to the likelihood that peers attempt to use more resources than they contribute to the community. To reduce free riding, a host of incentive schemes have been proposed to promote fairness among participating peers. Inter-domain resource contention can be viewed upon as a consequence of free riding because organizations (due to the workloads of their users) will try to utilize more resources than they contribute. In this section, we explore three incentive schemes that are potentially applicable to grid computing environments.

Reputation Index Scheme

The reputation index or peer-approved scheme maintains a set of ratings for participants. Participants are only allowed to use those resources of other participants that have a lower rating than them. Introducing a reputation scheme thus encourages participating organizations to maximize their ratings by servicing more jobs either by increasing their resource contributions or by optimizing the admission capacity of their computing resources. The reputation index scheme has widespread applicability to many cooperative-based systems ranging from trust management in document sharing systems [58], inter-domain sharing of computing resources [56] and digital rights management [45]. In many cases, participants in cooperative-based communities are anonymous to each other. Reputation mechanisms are also employed to prevent malicious attacks on distributed computations of remote applications [75].

We illustrate how the reputation scheme can be employed to manage inter-domain contention according to the criteria set up earlier in this chapter. We use a two organization case with organization 1 having $C1$ resources and organization 2 having $C2$ resources. In our example, we let $C1=4$ and $C2=2$. The reputation index of each organization is initialized with respect to their contribution of shared resources. Therefore, the initial values for the reputation index for organization 1 and 2 are $M1=4$ and $M2=2$ respectively. We assume that each request consumes exactly one unit of resource.

Assuming that both organizations attempt to gain access to each others resources before using their own, the reputation index works as follows: In the event that an organization receives and admits a request belonging to another organization, its reputation index will be increased by 1 while the reputation index of the organization that submits the request will be reduced by the same amount. Admission is successful only if the other organization's reputation index is higher or equal to the organization that receives the request. In the event that a request is rejected, either because there is insufficient resources or due to relatively lower reputation index, the request will then be executed

Figure 3.3: Illustration for Reputation Index Scheme

locally if possible.

In Figure 3.3, we illustrate the operation of the reputation index incentive scheme under different degrees of contention caused by a burst of requests submitted by each organization. For organization 1, the request burst is a set of requests labelled $r1$. Similarly, for organization 2, every request is labelled $r2$. The top diagram depicts a situation with no likelihood of contention. This is because the size of the request burst for both organizations is smaller or equal to their contributions. In the case for partial contention, only the size of the request burst of organization 2 is greater than its contribution. In the bottommost diagram, the total request burst size is greater than the sum of contributions of both organizations 1 and 2. On each diagram, the left hand table shows $r1$ arriving before $r2$. On the right of each diagram, the sequence is reversed. Both tables show the transition in values of the reputation index of each organization when a request is admitted or rejected.

In the case for no contention (top diagram), on the left hand side, organization 1 first submits its burst of requests and then followed by organization 2. The first $r1$ from organization 1 is submitted to organization 2. Since its reputation is higher than organization 2, the request will be accepted by the latter and is therefore admitted to $C2$. After $r1$ is admitted, the reputation index of organization 1 will be reduced by 1 while the index of organization 2 will increase by the same amount. When the second $r1$ is submitted, the reputation indices will be further updated in the same fashion. When $r2$ is submitted to organization 1, the latter is obliged to service the request because its reputation index is now less than organization 2. Since there are available resources in $C1$, the second $r2$ can be serviced. Similarly, the reputation index for organization 2 will be deducted and organization 1 is increased.

The right hand side gives the results for requests being submitted in the reverse order. From the results, it can be seen that, for both $r2$ requests, the reputation indices for organizations 1 and 2 do not change. This is because organization 2 has no alternative but

to use its own resources since its initialized reputation index is below that of organization 1. In the following request burst for $r1$, organization 1 is limited to local resource access because the resources in organization 2 are used up.

At partial contention, for both organizations 1 and 2, the effectiveness of the reputation scheme is also demonstrated because the request from organization 2 will not be admitted beyond its contribution. At full-contention (bottommost diagram), equitable access is achieved because the admission-contribution ratio is now 1 for both organization 1 and organization 2.

We show that under different degrees of contention, the reputation scheme is able to fairly admit the requests of each organization with respect to their resource contributions. We demonstrate this result by calculating the ratio between the total number of successfully admitted requests generated by users in an organization and the total resource contribution by each organization after the entire burst of requests have all been serviced.

Reputation index schemes address inter-domain contention by permitting an organization to access other's resources using the relative difference of their reputation indices. From the illustration, for the three distinct cases of resource contention, the requirements for inter-domain contention established early in the chapter are met. Firstly, for both organizations, the admission-contribution ratio ($\frac{\mu_k}{C_k}$) is kept equal to or below 1. Secondly, under full-load contention, all organizations have the same admission-contribution ratio.

The current work to improve the management of the reputation index mechanism, is for participants to only maintain the relative indices of their immediate neighbors instead of all participants in the network. For example, Andrade et al. employs the concept of 'Network of Favors' as a means of distributing the management of reputation index information of each peer in the network [4]. Each peer will maintain only the list of peers that it is either in favor or in debt. It will use this list to prioritize the admission of requests based on the list of peers in favor or to block the requests from

peers in debt. The ‘Network of Favors’ approach, however, can lead to disjoint clusters of participants because the reputation index of any organization is only defined on consensus with other immediate neighbors. Furthermore, if an organization has not been physically active to service remote requests, it may in time, lose its opportunity to leverage on any specific organization’s resources since the differences in their reputation indices are likely to change.

Quality-service Incentive Scheme

The quality-service scheme is similar to the reputation index scheme except that instead of using a single rating to entirely represent the pool of services offered by the participant, a set of distinct ratings is given, depending on the number of services offered. This leads to a system that is similar to the Paris Metro pricing scheme to manage differentiated services where each service class has its own unique reputation index and it is updated using different schemes.

Ontann et al. proposed a peer-to-peer collaborative framework for participant domains to provide resources for solving problems involving complex mathematical operations [65]. Each participant offers different classes of problem solving services. The services are differentiated in terms of the computational complexity of the problems that are solved. Each request, or a case base, contains a problem and specifies a service class to be invoked to solve the problem. An agent will, on behalf of the user choose an appropriate peer to execute the case base. On the receiving side, another agent will admit the request depending on reputation of the sending party specific to the service class chosen. If the request is admitted, the case base will be executed. The agent that received the request will reply to the sender the solution of the case base. The sender will in turn update the receiver’s reputation index specific to the service class of the sender’s request.

The service-quality schemes is an extension of the basic reputation index scheme. By partitioning the service pool into different classes, the system can better serve requests

with different quality-of-service needs. Hence, this approach is reasonably more suitable than the reputation index scheme to manage user-initiated contention.

Token-exchange Incentive Scheme

The token-exchange scheme is an alternative to the reputation index scheme. The main difference to the reputation index scheme is that, it does not rely on the relative values of reputation indices but the absolute quantity of tokens to administer equitable access to shared resources. Equitable access is achieved by assigning the initial amount of tokens of each organization to be proportional to the quantity of their resource contribution. Users are assigned a fraction of the total amount of tokens of each organization if they want to utilize the shared resources. The actual amount of tokens required for successful admission of a request is determined by the resource provider that services the request submitted by the user.

An illustration of the operation of the token-exchange mechanism is shown in Figure 3.4. Again, we use the same two-organization scenario¹ to explain the operation of the token-exchange incentive scheme to administer equitable access to shared resources in the event of resource contention. Each organization will attempt to use the resources of the other organization before trying to access its own. We again assume that each request takes up exactly one computing resource. The amount of initial tokens assigned to each organization is equal to the amount of computing resources contributed. When an organization attempts to use resources of another organization, its balance of tokens will be deducted by some amount if its user's request is admitted and executed by the resource providing organization. The providing organization will be credited the same amount of tokens deducted from the requesting organization.

We consider only two cases: no contention and full-load contention. At any one time, a total of 6 tokens are circulated between organizations 1 and 2 depending on

¹See page 47 for the description of the two-organization scenario

Figure 3.4: Illustration for Token-exchange Scheme

the sequence of requests submitted by both organizations. As depicted in Figure 3.4, the total number of admitted requests for each organization is proportional to the total number of initial tokens assigned. Since the initial amount of tokens assigned to each organization is equal to the the total capacity of resources contributed, the utility (total number of admitted requests) gained by each organization is therefore proportional to its contribution of resources to the community. In this way, the criterion of managing inter-domain contention can be satisfied.

The main advantage of this approach is that an organization does not need to maintain a list of its neighbor's reputation indices to administer contention management policies. In fact, an organization simply relies on its ownership of tokens to gain access to shared resources. As such, no additional communication protocols are required to authenticate the reputation status of other organizations.

One example of the token-exchange incentive scheme that is employed in a distributed shared computing system is Rexec [24]. In this system, each organization manages its own set of tickets (tokens). The quantity of tickets for a request is used to compute the proportion of CPU strides per quantum that the corresponding application gets relative to other applications that are currently running on the provider's computing resources. Hence, the time slice allotted to the application depends on both the number of tickets used and the degree of competition with other applications. Mojonation [89] is another project where resource providers aggregate different groups of resources into a service based on consumer's quality-of-service requirements. A price that the consumer has to pay for using the service is then generated by a provider agent. Each provider agent employs its own pricing scheme for resources it wishes to provide to other organizations. Golle et al. [42] propose a similar framework for regulating download access to shared files in a peer-to-peer network.

The token-exchange incentive scheme appears to be a plausible approach for managing both user and inter-domain level contention. As demonstrated earlier in the illustration,

inter-domain contention can be addressed because a fixed assignment of tokens to each organization automatically imposes a limit on its usage capacity on shared resources. Meanwhile, each user leverages on a proportion of the tokens, of which the total amount is based on the quantity of computing resources contributed by the organization. The tokens are therefore exploited to exchange for the desired quality-of-service gained by user's request. In order to fulfill the requirements, for managing user-initiated contention, the pricing framework can be designed in such a way that it maximizes the total utility gained by all users.

3.5 Discussion

Table 3.1 gives an overview of the contention management schemes that has been covered in this chapter. It shows the capability of each scheme to address either user-initiated or inter-domain contention management according to the requirements set out earlier in this chapter. The table also defines the areas of application and quality-of-service support of each scheme.

In the previous chapter, we discussed the differences between soft quality-of-service and hard quality-of-service. For soft quality-of-service, the system must try to deliver the resource guarantee to a request but does not have an obligation to fulfill the requirements. On the other hand, a system that provides hard quality-of-service must reserve sufficient resources to fulfill the requirements specified when the request is admitted.

The table shows that, almost all the schemes that are potential candidates for addressing inter-domain level contention do not provide hard quality-of-service support. This is because inter-domain resource management issues that are resolved by these schemes are often focused on high-level objectives to reflect the community's interest as a whole rather than the individual. Candidate schemes for addressing user-initiated contention are, on the other hand, designed to take into consideration each user's resource requirements in order to optimize the allocation of resources.

Table 3.1: Summary of Taxonomy of Contention Management Schemes

Scheme	Type	User-initiated Contention	Inter-domain Contention	Quality of Service Support	Applications
Diff-Serv	Paris Metro	Yes	No	Soft QoS	Computer Networks, Web Caches
	Utility Functions	Yes	No	Soft & Hard QoS	Real-time Databases, Schedulers
Auctions		Yes	No	Hard QoS	E-commerce, Computer Networks, Schedulers
Grid-Wide Schedulers		No	Yes	Soft QoS	Metacomputers, Schedulers
Incentives	Reputation Index	No	Yes	Soft QoS	File Sharing, Volunteer Computing
	Quality-of-Service	Partially	Yes	Soft QoS	File Sharing, Volunteer Computing
	Token Exchange	Yes	Yes	Soft QoS	File Sharing, Distributed Resource Management

While the token-exchange incentive scheme has the inherent features for managing inter-domain contention, it is not naturally designed to handle the requirements necessary to manage user-initiated contention. This is because the grid infrastructure requires hard quality-of-service support to grid applications. To serve as an admission control framework, the token-exchange incentive scheme must be extended with additional features similar to those schemes that are capable of addressing user-initiated contention. From our discussion, the admission control mechanism based on the token-exchange incentive scheme resembles an auction except that the algorithms for resource allocation are different due to the need to support the requirements for both user-initiated and inter-domain contention. The major extensions of the basic token-exchange incentive scheme for supporting admission control on shared resources will be discussed in the following section.

In the following subsection, we explore the additional features required to extend the token-exchange incentive scheme in order to fulfill the requirements set in section 3.2 for applying it to manage resource contention into the grid computing infrastructure.

3.5.1 A Pricing Framework for Admission Control of Advance Reservations

From our study of the differentiated service and the auction, the treatment of user-initiated contention through the use of valuation functions, suggests a tight coupling between quality-of-service provisioned to an admitted request and the price of its corresponding service.

With respect to the token-exchange incentive scheme, the valuation function corresponds to the pricing framework that is used by both consumer and provider agents. Each consumer agent uses a pricing framework to translate the quality-of-service of each request to bid before it is submitted to the broker. Each provider agent, on the other hand, uses a pricing framework to generate the admission price for the same request.

Existing token-exchange incentive schemes are based on the proportional-share resource allocation scheme. Different from the illustration given in section 3.4.2, the proportional share scheduler operates on time shared computing resources. This means that multiple jobs can run on the same resource. This means that the instantaneous quality-of-service gained by a request is dependant on the current number of other executing jobs. Hence, in order for a request to maintain its original time share of resources, an agent, on behalf of the requestor must continuously increase the expenditure of tokens if other requests are admitted after it begins execution.

In a grid computing infrastructure with an advance reservation facility, it is a necessity to immediately guarantee the quality-of-service requirements specified by the request once it has been admitted. For this reason, the proportional token-based time sharing scheme cannot be used. Furthermore, the admission price of a given request's desired quality-of-service is also influenced by the instantaneous level of competition for resources.

The motivation for managing user-initiated contention requires the system to prevent users from oversubscribing for shared resources so as to maximize the total utility gained by all users. The above issues points out the need for a pricing framework that is to be integrated into the advance reservation mechanism to support admission control of requests based on the token-exchange paradigm. In chapter 4, we introduce the admission control framework that is augmented with a novel pricing framework to improve the system admission ratio. The system admission ratio is an equivalent metric to assess the total utility gained by users.

3.5.2 A Metric for Evaluating Fairness

The requirement set out for managing inter-domain contention in section 3.2 implies the need for a metric that must be fulfilled in order to ensure that all organizations obtain their share of resources proportional to their contribution.

The token-exchange operates in a grid infrastructure where agents independently make

resource management decisions on behalf of their organizations. Since these decisions are usually policies for trading computing resources, conflicts may arise when agents compete for resources during contention. The metric quantitatively evaluates the desired solution to address inter-domain level contention. Existing works that employ the token-exchange incentive scheme do not define any specific metric for equitable access to shared resources [22, 24, 25]. This is because it is taken for granted that equity can be achieved if the total number of tokens assigned to an organization is proportional to the quantity of resource contribution. In chapter 6, we define a metric for fairness and make use of the metric to show that token-exchange incentive scheme does not automatically fulfill the ideal conditions set by the metric unless additional system properties are enforced.

3.5.3 System Architecture Considerations

The system architecture of the grid infrastructure is often dictated by the applications it serves and therefore can exert some influence on the design of the token-exchange incentive scheme to manage resource contention. Our study of metacomputing systems in the previous chapter proposes a conceptual resource management framework that is divided into 3 tiers – the consumer, provider and broker. The broker is the point of contact where matchmaking functions are performed. The matchmaker’s role is to find a suitable candidate service created by a provider agent to a request forwarded by a consumer agent. Each successful pairing between a request and a service will result in expenditure on the requesting organization and earnings to the organization that creates the candidate service chosen by the broker.

In cases where each organization has its own matchmaker, equity may not be achieved if some organizations attempt to form coalitions by deliberately serving only certain organizations while leaving others out. Consequently, organizations that are isolated may not be able to benefit from sharing since they are unable to earn any tokens if their resources are not chosen. Therefore, in chapter 6, we show how the broker’s matchmaking mechanism acts as a governing authority to achieve the conditions for fairness.

3.5.4 Delegation of Trading Decisions

The major drawback of employing the token-exchange incentive scheme is that users need to make a pricing decision in order to gain access to shared resources. This problem is addressed by employing a consumer agent for each organization to make trading decisions on behalf of their users. The secondary benefit of using the consumer agent to act as a decision proxy for users is that, it eliminates the need to decide on how tokens should be re-distributed among users – especially when users’ resource demands are likely to vary over a period of time. Delegation of trading decisions to agents is also necessary for the system to satisfy the conditions imposed by the metrics to address both user-initiated and inter-domain contention. In chapter 7, we implement an adaptive policy to both consumer and provider agent’s trading decision mechanism in order to reduce the trade-off between fairness and the system admission ratio with respect to the initial amount of tokens assigned to each organization.

3.6 Summary

We have identified several approaches that are potential methods for managing contention in a grid computing infrastructure and compared their relative merits and weaknesses.

The differentiated service scheme and the auction are both suited to address user-initiated contention. In the differentiated-service schemes, valuation functions enable users to map their quality-of-service preferences to a single value. When contention occurs, the system makes use of the valuations of each user to find a feasible allocation using a certain pre-defined set of metrics. The auction is a more complex approach because it permits negotiation among consumer and the auctioneer to reach a competitive equilibrium on how resources are allocated.

Candidates for managing inter-domain contention are wide in range in terms of design. Grid-wide schedulers are management modules that handle resources linked to a virtual

organization. Since all scheduling decisions are typically made at a central node, user's valuations are normally not expressed because the scheduler's goal is to admit workloads based on a VO-level policy.

Incentive schemes have been augmented into peer-to-peer systems to encourage and sustain the peer participation. From specific examples, we illustrated that they are adequately suited for managing inter-domain contention.

Of the three incentive schemes cited, the token-exchange incentive scheme is most suited to manage both user and inter-domain level contention. The initial amount of tokens assigned is proportional to the actual contribution of shared resources and therefore, it automates equitable access to shared resources. Because tokens are exchanged for resource usage, this scheme resembles an economy-based model that can as well be employed to provide a facility for admission control of requests. It is this feature that essentially bridges the existing gap between user-initiated and inter-domain contention management schemes. The token-exchange incentive scheme as illustrated in section 3.4.2 ensures that equitable access to shared resources can be achieved in a situation of inter-domain contention. Since the amount of tokens that an organization owns is limited by the total capacity of resource contributed, each request can only be assigned a fraction of the tokens (budget) to gain access to shared resources. The quantity of tokens assigned to a request will determine if it can be admitted according to the admission price set by provider agents. User-initiated contention can also be addressed because the admission price² is determined by the level of resource contention and the quantity of resources demanded by the request.

To use it in a grid computing infrastructure for resource management, the token-exchange scheme is extended to serve as an admission control mechanism for advance reservation of the shared resources. The extensions proposed are:

1. A pricing framework to manage the hard quality-of-service requirements specified

²The admission price is computed from a formula that will be introduced in chapter 4.

by users' requests.

2. A metric to measure the desired conditions when addressing both user-initiated and inter-domain contention management.
3. Delegation of trading decisions to agents in order to free users from the need to determine the price for their requests to be admitted to shared resources.

Chapter 4

A Pricing Framework for Supporting Advance Reservations to Manage User-initiated Contention

4.1 Introduction

In this chapter, we focus on treating user-initiated contention. It occurs when one or more users oversubscribe for shared resources to such an extent that a large number of other users may be deprived of gaining access to shared resources in the virtual organization (VO).

We acknowledge this problem because real workload traces of supercomputers exhibit sharp but intermittent jumps in request arrivals when observed across their entire time frame [80]. At large, the problem of distributed resources contention on computing resource has been a concern but has not been concretely addressed. In an experimental study conducted by Kleban et al. [55], they showed the presence of queue storms on resource brokers due to resource overload caused by correlations in request arrivals. While their work concluded that only with adequate resource contributions can the grid deal with the adverse effects caused by the request arrival correlations, even though they did not propose any technique to deal with this problem. Planet Lab reported intermittent contention on multi-domain computing resources and proposed an auction-based

mechanism for resource allocation in the event of resource shortage. However, they also commented that an auction mechanism might not scale well for a grid-based implementation [7].

To address user-initiated contention of shared resources, we considered the statistical distribution of the workload traces in terms of both CPU usage and demanded execution time. Taking reference from a variety of real workload traces of supercomputers, a common feature observed is that, in terms of CPU and execution time, there are a small number of requests having relatively larger resource requirements. Our concern is that these requests may prevent a large number of shorter requests requiring less resources from gaining admission. The straightforward approach is to preempt larger requests as to allow those with less computing resource requirements to fill-in. While a number of existing schedulers have been designed to handle the admission of deadline-based jobs (requests) in parallel computing systems using this strategy [49, 69, 82], they cannot be directly applied to a grid infrastructure. This is because these schedulers generally assume that re-scheduling of jobs (requests) is possible as long as their execution deadlines are not exceeded. The grid infrastructure, however, needs to provide an advance reservation facility in order to provide hard quality-of-service guarantees to grid applications [26, 37]. An advance reservation facility may not always allow preemption because once the quality-of-service of an admitted request is granted, it may not be modified unless permitted by the application or the user.

An alternative strategy is to design a policing strategy to regulate the admission of requests that have relatively larger resource requirements. We do this by means of a pricing framework to administrate the admission control of requests. The pricing framework is utilized by a provider agent to compute an admission price in order to determine if a request should be admitted. The consumer agent, on behalf of the user must bid above the admission price in order to gain access to the shared resources. The price is based on (1) the degree of resource contention, and (2) the opportunity cost faced by the resource provider to admit the request. In (1), the level of contention is estimated by calculating

the instantaneous utilization of resources managed by a provider agent. In (2), the opportunity cost of admitting a request is measured by calculating the total earnings for servicing a number of requests that may require less resources than the request that is to be admitted. By taking the opportunity cost into consideration, the requestor has to absorb the possible loss of earnings by the resource provider for not admitting other requests that require less resources.

In section 4.2, we review the existing work on advance reservation admission control policies that are applied to a grid computing infrastructure. In section 4.3, we introduce the necessary background work to illustrate the admission control strategy that is based on the pricing framework. In section 4.4, details of the admission control policing will be described. In section 4.5, we study and compare the performance of the admission control framework with other benchmarks. This is followed by the conclusions in section 4.7.

4.2 Related Work

To date, a solution to manage contention in a grid computing platform that supports advance reservation has not been articulated. In fact, most of the additional features incorporated to manage advance reservations are purposed to improve the utilization of the underlying managed resources. The advance Reservation Server (ARS) [40] that employs the Dynamic Soft Real Time (DSRT) [1] algorithm to allocate CPU resources for applications is essentially a FCFS scheme. It uses a best effort scheme to assign computing resources by sampling a CPU-time reservation window so as to avoid any resource usage conflicts amongst application. The Resource Broker (RB) [54] is an extension of the ARS because it further introduces a basic mechanism to support re-negotiation. However, improvement to the overall system utilization is dependent on the rate of re-negotiation initiated by applications. In the event that a large proportion of users/applications are unwilling to re-negotiate their quality-of-service to accommodate newly arriving requests, the performance would fall back to that of ARS. Foster et al. has demonstrated this

effect by a series of experiments and their results show that there is a sharp improvement in resource utilization if the fraction of re-startable requests over the total number of submitted requests is increased [78]. In the context of their work, restartable means that the application, upon its execution is preempted, is re-started from the point of preemption when its execution is resumed.

There are other user-centric schemes where resource allocation is improved by assigning usage quotas to the intended users [48]. With usage quotas, users would self-impose a limit on their usage of resources. With less CPUs assigned to each request, the advance reservation mechanism is able to better allocate and hence, utilize resources more efficiently because smaller requests create less ‘gaps’ in reservation window. This admission control strategy is similar to the approach we take, except that the policies used to admit requests are different.

In order to find alternative quality-of-service levels that meet a request’s resource requirements in the event that the desired resource specification cannot be met, utility or value based functions are introduced to allow the system to re-negotiate with the requestor an acceptable lower quality-of-service. Utility functions have traditionally been introduced to improve the flexibility and performance of real-time systems [17]. As already discussed in chapter 3, there are numerous applications that make use of utility functions to perform resource optimization under contention. An example of applying utility functions to manage resources in a grid computing infrastructure is the Reservation Scheduler with Priorities and Benefit Functions (RSPB) designed by Maheswaran et al. [62]. In the event of resource shortage, the advance reservation facility will try to find an alternative allocation to a set of requests in ascending order of priority. The priority of each request is determined by a benefit (utility) function which takes into account the demanded quality-of-service and the actual amount of resources currently allocated to the request. The main objective of admission control, in this case, is to partition the available resources and allocate them to requests such that the aggregate benefit of all requests is maximized.

The above admission control schemes on advance reservations are generally designed to provide the flexibility for applications to negotiate lower quality-of-service levels so that the system can improve the overall utilization of the underlying resources. To deal with contention, especially in a situation where users have the rights to disallow any re-negotiation of their guaranteed quality-of-service, a deliberate attempt to filter or block requests on the basis of their quality-of-service requirements is required. For instance, if a request with large resource requirement is admitted during contention, it can hinder the admission of a large number of requests having lower resource requirements. If the shared system is to be user-centric, it is therefore reasonable for the system to admit more requests since this will improve the total utility gained by all the users. For these reasons, an admission control must be incorporated to reduce the number of requests demanding large amount of computing resources that are likely to cause resource contention.

4.3 Non-preemptive versus Preemptive Admission Control Strategy

In this section, we present a general strategy to support the administration of admission control on advance reservations. We first introduce the system model and its architecture.

4.3.1 Overview of System Architecture

In this section, we explain the general strategy for each provider agent to perform admission control on its underlying resources. Resource contention occurs when a burst of requests are submitted to the system. Under resource contention, one or more requests may not be able to gain admission because the system's advance reservation mechanism is unable to locate and assign the required resources to meet the quality-of-service requirements of all the request in the burst.

Given the presence of an advance reservation framework, once a request is successfully admitted, the starting time of its corresponding service cannot be directly modified by

the scheduler unless permitted by the application. Therefore, even if the preemption does not result in the service exceeding its maximum allowable slack, the service still cannot be delayed.

Systems without an advance reservation facility, on the other hand, may allow admitted services to be delayed, thereby, allowing urgent requests to be admitted as long as the preempted services do not exceed their maximum allowable slack. Below is a formal definition for both preemptive and non-preemptive services.

- Preemptive Admissibility

In our context, if a system allows preemption, this means that the starting time of any scheduled service may be delayed as long as its slack ratio of the service does not exceed the maximum allowable slack ratio. Let $create(s_j)$ be the time when a service s_j , is scheduled to start execution by a provider agent for its corresponding request (r_i) . Also, let $\alpha(s, t)$ ¹ be the slack ratio assigned to service s at t . Formally, if a service s_j can be preempted, then $\exists t', t' > create(s_j)$, such that $\alpha(s_j, create(s_j)) < \alpha(s_j, t') \leq \alpha(r_i)$. The service s_j will be rescheduled to start at time t' .

- Non-preemptive Admissibility

If the system does not support preemption, as in the case of advance reservations, once the service is scheduled, it cannot be changed.

4.3.2 A Motivating Example

We compare the admission performance between a generalized non-preemptive admission control scheme to a preemptive one using an illustration. Since the preemptive scheme has lesser scheduling constraints, we use it to serve as a benchmark to gauge the admission performance of the non-preemptive scheme – that is, a system supporting advance reservation. We use the example to demonstrate the difficulties of scheduling the same

¹ $\alpha(s_j, t)$ is an extension of $\alpha(s_j)$, to include the time when the slack ratio of a scheduled service is modified.

Table 4.1: Quality-of-service Requirements and Submission Times of Requests

r_i	0	1	2	3	4	5	6	7	8	9	10
$\alpha(r_i)$	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
$submit(r_i)$	0	0	0	0	0	2	2	8	8	12	12
$T(r_i)$	16	16	16	10	10	8	8	8	8	8	8
$PE(r_i)$	20	20	20	2	2	2	2	2	2	2	2

Table 4.2: Service Instance Assignment for Different Admission Control Schemes

r_i		0	1	2	3	4	5	6	7	8	9	10
s_j	Preemptive	0	1	–	2	3	4	–	5	–	6	–
	Non-preemptive	0	1	–	–	–	2	–	3	–	4	–
	Load-controlled non-preemptive	0	–	–	1	2	3	4	5	6	7	8

set of requests in a non-preemptive setting. In addition, we also use this as a motivating example to illustrate how the proposed admission control strategy can be applied to improve resource allocation results for the non-preemptive scheme so that its performance can match more closely to the preemptive scheme.

We consider an ordered set of a burst of 3 large requests followed by a trail of 8 small requests with different submission times. $R = \{r_0, r_1, \dots, r_{10}\}$. The quality-of-service requirements and submission time of requests are presented in table 4.1. The overall objective of admission control is to improve the admission ratio – that is, the ratio between the total number of admitted requests and the total number of submitted requests.

Figure 4.1 shows the scheduling results for a generalized preemptive scheme (case 1), a best-effort non-preemptive scheme (case 2) and a load-controlled non-preemptive scheme (case 3). Each request is assigned a service instance that must be scheduled. Table 4.2 shows the corresponding sequence of service instances for a given admission control scheme. Let $mapped()$ be a function that returns a set of computing nodes for a given scheduled service instance. For case 1, at $t = 0$, the system will process requests r_0 to r_4 in order. Service instance s_0 and s_1 are created and scheduled upon admitting r_0 and r_1 respectively. Therefore, $mapped(s_0) = \{n_0, n_1, \dots, n_{15}\}$ and

$mapped(s_1) = \{n_{16}, n_{17}, \dots, n_{31}\}$. r_2 requires more than the available capacity and cannot be placed behind either s_0 or s_1 . When the system processes r_3 , it finds that there are not enough resources as most are already taken by r_0 and r_1 . Since either r_0 and r_1 can be delayed, it chooses r_0 . The system creates s_2 for r_3 and the assignment is $mapped(s_2) = \{n_0, n_1, \dots, n_7\}$. The slack ratio on s_0 for r_0 is now as $\alpha(s_0) = 0.1$. For r_4 , s_3 is created and s_1 has to be delayed for the newly generated service. The slack ratio on s_1 is $\alpha(s_1) = 0.1$. At $t = 2$, the system first processes r_5 and then r_6 . For r_5 , s_4 is created and the system allocates the remaining PEs such that $mapped(s_4) = \{n_{10}, n_{34}, \dots, n_{19}\}$. When r_6 is processed, the system identifies s_0 and s_1 as candidate services to be delayed. We assume that the system chooses s_0 (in ascending scheduled order of index) to be delayed. s_5 is created and assigned $mapped(s_5) = \{n_0, n_1, \dots, n_7\}$. The slack on s_0 is $\alpha(s_0) = 0.2$. At $t = 8$, only r_7 can be admitted, because both s_0 and s_1 are still executing. Similarly, at $t = 12$, only r_9 can be admitted. Therefore, in the preemptive scheme a total of 8 requests are admitted.

In a best-effort non-preemptive scheme, r_0 and r_1 have services s_0 and s_1 created. However, it cannot service r_3 , r_4 and r_6 . This reduces the total number of admissions to 5. Suppose policies are introduced to block out large requests. Therefore, r_0 to r_2 are blocked. In this way r_3 and r_{10} can all be admitted. The total number of admitted requests can be increased to 8 and thus meeting the gain achieved by preemptive admission control scheme. However, if we selectively admit large requests, instead of rejecting them all, e.g., accepting r_0 but reject r_1 and r_2 , all other smaller requests can still be admitted. This raises the total number of admitted requests to 9.

From the short illustration, it is thus shown that in general, with additional freedom to re-schedule large requests, the preemptive scheme has a clear advantage over the non-preemptive scheme. However, for the non-preemptive scheme, if large requests are only selectively admitted when resources are heavily utilized, the overall gain in admissions can still be comparable to the performance of the preemptive scheme. We also exemplify that not in all circumstances preventing the admission of large requests is optimal to

Figure 4.1: Illustration of (1) Preemptive, (2) Non-preemptive and (3) Load-controlled Non-preemptive Admission Control

the collective benefit experienced by users because not all long requests are necessarily a hinderance to better admission performance. The negative effect caused by large requests is dependant on the instantaneous utilization of the resources within the request's execution window. For example in case 3, the instantaneous utilization prior to admitting r_0 is smaller than when r_1 is considered for admission. Essentially, if admittance of a request is based on its size (CPU and execution time) and instantaneous utilization, we show that unnecessary blocking of large requests can be avoided. In our approach for admission control, we thus make use of three parameters to determine the price for admitting a request – its CPU usage, execution time requirements and the instantaneous resource utilization of the underlying resources where the request is to be admitted and executed.

4.4 Admission Control Framework

4.4.1 Pricing Framework Design

In line with a number of quality-of-service frameworks for grid applications [2, 46, 84], we first model the capacity of the underlying resource managed by each provider agent as a schedule in terms of the total resource capacity against time. An instantiated service covers a fraction of the capacity-time window. The basis of pricing is to establish a value for each scheduled candidate service with respect to its demanded quality-of-service and the current degree of contention or instantaneous utilization. For each candidate service s_j , to be created for a request r_i , the quality-of-service obtained is in the form $\langle PE(s_j), T(s_j), \alpha(s_j) \rangle_j$ as described earlier in chapter 2. To quantify the degree of contention, the resource utilization within the request's permissible execution window is measured. The permissible execution window is the time interval between the time when the service for a request is admitted at the provider agent (i.e., $t_s = admit(r_i)$) and the execution deadline of the service (i.e., $t_e = (\alpha(r_i) \times T(r_i)) + T(r_i) + admit(r_i)$). To calculate the resource utilization, we first introduce the notation C to represent the full capacity of

resources that each provider agent manages. We obtain ΔC , the resource utilization by calculating the average resource capacity consumed by all scheduled services overlapping or contained in the permissible execution window of the request. Eqn 4.4.1 shows how the resource utilization is computed. $C(t)$ represents the total number of resources used by all scheduled services at time slot t .

$$\Delta C = \frac{1}{t_e - t_s} \int_{t_s}^{t_e} C(t) dt \quad s.t. \quad t_e > t_s \quad (4.4.1)$$

The top diagram in Figure 4.2, illustrates the basic scheme for calculating the admission price of a newly generated service. For simplicity, we let $PE_j = PE(s_j)$ and $T_j = T(s_j)$. To obtain the base admission price, we add PE_j to ΔC and divide the sum with C , to get $\frac{\Delta C + PE_j}{C}$. In this way, the admission price is proportional to both the resource utilization and the total number of CPUs required by the request.

As explained earlier in this chapter, the pricing framework also needs to take into account the opportunity cost of admitting a request. Since requests have different CPU requirements, there is a need to account for the loss in earnings for admitting a request requiring more CPUs in place of a number of requests requiring less CPUs. Assuming that ω is the smallest resource capacity required by any request submitted to the system, eqn 4.4.2 shows how the base admission price of a request requiring PE_j CPUs is computed when opportunity cost is taken into consideration.

$$\Gamma_\omega = \left[\frac{\Delta C + \omega}{C} + \frac{\Delta C + 2\omega}{C} + \dots + \frac{\Delta C + PE_j}{C} \right] \quad (4.4.2)$$

As depicted in the bottom diagram of Figure 4.2, the base admission price is computed as if a burst of requests, each requiring ω CPUs are submitted to the system. We assume that the requests, when admitted, are stacked – starting from the top of ΔC until they reach $PE_j + \Delta C$. This assumption leads to the formulation of eqn 4.4.2, which yields eqn 4.4.3 when the arithmetic sum is applied. The calculations are given in appendix A.

Figure 4.2: Calculation of the Admission Price of an Arbitrary Request

$$\Gamma_{\omega} = \frac{PE_j}{2\omega C} [2\Delta C + \omega + PE_j] \quad (4.4.3)$$

We then consider the cost for executing a service on its execution time by multiplying the ratio between the T_j and a fixed scaling factor τ with the base admission price to obtain $\Gamma_{\tau,\omega}$, the final pricing framework in eqn 4.4.4. Likewise, ω is a fixed value that is used scale the influence of PE_j on the admission price.

$$\Gamma_{\tau,\omega} = \Gamma_{\omega} \times \frac{T_j}{\tau}$$

$$\Gamma_{\tau,\omega} = \frac{PE_j T_j}{2\omega \tau C} [2\Delta C + \omega + PE_j] \quad (4.4.4)$$

4.4.2 Time Delay

We assume that users would try to obtain their reservation of shared resources as soon as they submit their requests. Resources may be locked-in for a considerable period of time if a scheduled service has both very long execution times and high CPU requirements. To cope with this problem, some delay on the starting time of a service is deliberately introduced so that other requests demanding less execution time submitted subsequently can be admitted and their corresponding services be scheduled ahead in the reservation schedule. The delay is therefore introduced to a service when it is scheduled. Furthermore, a service cannot be rescheduled by the system unless permitted by the requestor. Since it does not make sense to delay the execution of services with very short execution times, the delay is only introduced to services with 50hrs of execution time or more. The delay introduced to a service is computed based on the maximum allowable slack ratio ($\alpha(r_i)$) and the instantaneous degree of contention ($\frac{\Delta C}{C}$). To compute the delay that is to be assigned to a scheduled service, the slack ratio ($\alpha(s_j)$) is first calculated using Eqn 4.4.5. As shown, $\alpha(s_j)$ is increases proportionally to the maximum allowable slack ratio with

respect to the instantaneous degree of contention. Since $0 \leq \frac{\Delta C}{C} \leq 1$, the slack ratio will always be between 0 and $\alpha(r_i)$.

$$\alpha(s_j) = \alpha(r_i) \times \frac{\Delta C}{C} \quad (4.4.5)$$

The time delay is then calculated by multiplying the slack ratio with the execution time of the service ($T(s_j)$). To compute the starting time of the service ($start(s_j)$), the time delay is added to the request's admission time ($admit(r_i)$). Therefore, $start(s_j) = admit(r_i) + \alpha(s_j)T(s_j)$.

To illustrate the key benefit of applying delays to service instances with very long execution times, we first consider a burst of requests that arrive at the provider agent. Figure 4.3 depicts the schedule of computing resources within the permissible execution window for the request having the longest execution time (i.e., r_{i+n}). The top diagram shows the application of time delay to each request with respect to the policy described earlier. All requests are assumed to require the same quantity of CPUs (that is, m). The first request r_i is not assigned any delay. The second request is assigned a slack ratio of $\frac{1}{m}\alpha(r_{i+1})$. When subsequent requests are admitted, the slack ratio is assigned in the order depicted in the diagram.

The full-load contention is a situation when the full capacity of resources is used up by the burst of requests that have been admitted by the provider agent. From Figure 4.3, it can be seen that by introducing delays (top diagram), the time interval (or region) of full-load contention is much smaller than the case if no time delay is introduced (bottom diagram).

The purpose of this time delay assignment policy is to reduce the time interval when resources become fully utilized if a burst of requests demanding long execution times are submitted. The advantage of introducing delays in this manner is that by effectively reducing the time interval of full-load contention, there is a higher chance for other requests requiring less CPU and execution time to be admitted. Doing so, the admission

Figure 4.3: Full-load Contention on a Request Arrival Burst with (Top) and Without (Bottom) the Application of Delay

control mechanism is able to admit more requests with less resource requirements.

From eqn 4.4.5, it can be seen that the reduction of the time interval of full-load contention is proportional to the amount of $\alpha(r_i)$. Since there is a practical limit at which $\alpha(r_i)$ can be increased, there is also a limit that system can further improve the admission of requests. Moreover, requests demanding long execution times may be unnecessarily assigned with delays. In the experimental evaluation, we compare the performance of the admission control policies with and without the application of delays to requests. We then evaluate the trade-off between the improvement to the admission of requests and the average delay incurred on requests having long execution times.

4.4.3 Admission Control and Resource Allocation

The pseudocode in Figure 4.4.3 defines the admission control scheme administered by each provider agent. Upon receipt of request r_i , the algorithm will find the permissible execution window bounded by t_s and t_e .

In line(1), *getCurrentTime* obtains t_s from an information provider that is synchronized between the provider agent and the broker.

In line(2), *findDeadline* computes the t_e of the request by using t_s and the quality-of-service metadata obtained from r_i .

The next task is to find the set of services that are covered by the permissible execution window of r_i . That is, those services with execution time intervals that either intersect or are contained by the time interval between t_s and t_e .

Lines (3) to (9) perform the search function on currently executing and queued services. S^r is a set of services that are covered by t_s and t_e . It is used to compute the average utilization, *ld_sum*, that is later used by the pricing framework to compute the admission price.

ld_sum is obtained by calculating the average load-ratio within the permissible execu-

```

On request  $r_i$  Arrival
#Calculate request's permissible execution window
1.  $t_s \leftarrow \text{getCurrentTime}()$ 
2.  $t_e \leftarrow \text{findDeadline}(t_s, r_i)$ 
#Find services in request's permissible execution window
3. for all  $s_j \in S$ 
4.    $c_s \leftarrow \text{start}(s_j)$ 
5.    $c_e \leftarrow \text{end}(s_j)$ 
6.   if  $!(c_s < t_s \wedge c_e < t_s) \wedge !(c_s > t_e \wedge c_e > t_e)$  then
7.      $S^r \leftarrow S^r \cup s_j$ 
8.   end if
9. end for
#Calculate resource utilization
10.  $t \leftarrow t_s$ 
11.  $ld\_sum \leftarrow 0$ 
12. while  $t \leq t_e$ 
13.    $CPU\_sum \leftarrow 0$ 
14.   for all  $s_j \in S^r$ 
15.      $c_s \leftarrow \text{start}(s_j)$ 
16.      $c_e \leftarrow \text{end}(s_j)$ 
17.     if  $!(c_s < t \wedge c_e < t) \wedge !(c_s > t + \Delta t \wedge c_e > t + \Delta t)$ 
18.        $CPU \leftarrow PE(s_j)$ 
19.        $CPU\_sum \leftarrow CPU\_sum + PE(s_j)$ 
20.     end if
21.   end for
22.    $ld\_sum \leftarrow ld\_sum + \frac{CPU\_sum}{C}$ 
23.    $t = t + \Delta t$ 
24. end while
#Apply pricing framework to compute admission price
25.  $ld\_sum \leftarrow \frac{ld\_sum}{t_e - t_s}$ 
26.  $\alpha \leftarrow \text{getWorstCaseSlack}(r_i)$ 
27.  $t^q \leftarrow \text{getStartTime}(t_s, \alpha, T(r_i))$ 
28.  $p \leftarrow \text{computePrice}(PE(r_i), T(r_i), ld\_sum)$ 
#Check request's bid against admission price
29. if  $p \leq \text{getBid}(r_i)$  then
30.    $s_{new} \leftarrow \text{createContract}(r_i, t^q, p)$ 
31. else
32.    $s_{new} \leftarrow \text{NULL}$ 
33. end if

```

Figure 4.4: Admission Control of Request

tion window. The load-ratio measures the resource utilization at a narrower time interval within t_s and t_e . The while loop in line(12) tries to find the sum of all load-ratios at each regular time interval denoted by Δt , between t_s and t_e . In the body of the while loop, the load ratio is calculated by iterating through S^r (line(13) to (21)), to find the services that are covered at each time step t . The load ratio at each time step is first calculated by finding the sum of CPUs used up by all services executing between t and $t + \Delta t$. The sum is then divided by the organization's actual resource contribution C .

From lines (25) to (28), the admission control scheme first computes the stipulated time delay to be introduced to a service and its corresponding admission price. The maximum allowable slack ratio, $\alpha(r_i)$, is obtained from *getWorstCaseAlpha*. Based on *ld_sum* and the request's quality-of-service parameters, the actual starting time of the service is obtained from *getStartTime* in line(27). The admission price for the newly generated service will then be computed based on the pricing framework that is implemented in *computePrice*.

In line (29), the admission price will then be compared against the consumer's bid (request) price. If the bid is greater than the admission price, the provider agent will allocate the required resources and create the corresponding candidate service for the request. Otherwise, it will return a *NULL*. If resource allocation is successful, *createContract* will return a candidate service handle to the broker. Again, a *NULL* is returned if resource allocation fails. If the candidate service is selected by the broker, the provider agent will keep the candidate service in the service queue and scheduled for execution. Otherwise, it will be deleted.

4.5 Performance Evaluation

4.5.1 Workload Trace Generation

Synthetic workload traces are generated from real workload traces of supercomputers to evaluate the performance of the admissions control scheme against a set of benchmarks. Because the workload traces are obtained from supercomputers with different number of computing nodes, the first step is to standardize the workloads to fit into a system with a prescribed number of computing resources. To do that, we employed a technique proposed by Ernemann et al. [30] to scale supercomputer workload traces to fit a 1024-CPU system. The intention of their work was to improve the portability of widely used parallel workload traces to evaluate schedulers that manage larger sized systems than the originating supercomputers.

Workload scaling is performed in three steps: First, each request in the trace is either replicated or the demanded CPU capacity scaled. To do that, two parameters need to be defined: a precise scaling factor and a threshold for choosing between replicating or increasing the request's capacity requirements using the scaling factor. The scaling factor is determined by the ratio between the size of the scaled system and the number of CPUs in the supercomputer that the real workload traces are obtained from. After scaling the workloads in terms of CPU consumption, requests taking up more than 1024 CPUs are removed from the synthetic workload traces. Finally, time scaling is introduced to manipulate the execution time of requests to simulate different degrees of resource oversubscription by users. This is done by dividing the demanded execution time of each request in the synthetic traces by a time scaling factor which is less than 1. Table 4.3 is a set of synthetic workloads generated from their respective real workload traces obtained from the parallel workload archive [33]. The leftmost column of the table shows the synthetic workload traces that are named after the original workload. The second column shows the time scale factor applied to the synthetic workload traces. The following columns contain data on the mean CPU usage and the corresponding standard deviation,

Table 4.3: Parameters for Synthetic Generation of Workload Traces

Workload Trace	Time Scale Label (value)	CPU- μ	CPU- σ	T- μ (Hrs)	T- σ (Hrs)
KTH SP2	Skew-005 (0.05)	62.13	104.47	81.85	128.71
	Skew-01 (0.10)	62.28	103.45	45.78	71.58
	Skew-002 (0.20)	61.31	98.60	23.72	33.62
CTC SP2	Skew-005 (0.05)	57.08	76.19	72.88	100.25
	Skew-01 (0.10)	55.69	75.44	42.23	51.61
	Skew-002 (0.20)	52.75	67.20	24.46	26.66
Par 95	Skew-005 (0.05)	62.48	70.37	80.97	87.61
	Skew-01 (0.10)	63.85	76.80	46.05	44.68
	Skew-002 (0.20)	64.67	71.03	24.62	21.94

the mean execution time of requests and standard deviation after time scaling is applied.

4.5.2 Simulation Parameters and Performance Metrics

- Load-contribution Ratio

In the experiments, we used a single workload trace to represent the user demand patterns for each organization. The load-contribution or LC ratio is basically a ratio between the average amount of CPUs demanded by all requests in a workload of an organization and the resource contribution made by the organization. The load contribution therefore, quantifies the likelihood of resource contention that is created by all users in an organization. As mentioned, all workload traces are scaled to fit into a system of 1024 CPUs. Hence, if an organization contributes 1024 CPUs, its LC ratio is nominated to be 1.0. Other LC ratio values are obtained by changing only the size of resource contributions. In our work, the resource contribution of each organization ranges from 512 to 2560. Correspondingly, the LC ratio for each organization ranges from 2.0 to 0.4. Therefore, a lower LC ratio reflects a lower degree of contention and vice-versa. The LC ratio for all organizations are assumed to be the same for each simulation run.

- Consumer Strategy

We define different classes of consumer bidding strategy based on the specific bid values submitted by users. To simplify the analysis on user's overall impact on the system performance, we apply the same bid value to every request for a given consumer strategy class. Given that p is the bid price, we choose $p = 4.0$, $p = 40.0$, $p = 400.0$ and $p = 4000.0$ to represent four distinct classes. The above values are obtained with respect to the potential earnings that an organization gains when its underlying resources are fully utilized (that is, full-load earning in short). The full-load earnings are obtained by calculating the sum of the admission price of a regular set of requests, such that, when admitted, they will fully utilize the resources owned by an organization.

Since the admission price is calculated from the pricing framework, the following nominal values were used for the various parameters of the pricing framework: $\omega = 8$, $\tau = 5$, $PE_j = 64$ and $T_j = 40$. Because the resource contribution made by an organization ranges from 512 to 2560, we calculated two extreme values for the full-load earnings. For a resource contribution of 2560, the maximum full-load earnings gained by an organization is 1060. The minimum full-load earnings gained as a result of contributing 512 computing nodes is 125. We defined the bids with respect to the potential full-load earnings of an organization because users ultimately have to leverage of their organization's earnings to make use of shared resources.

We associate $p = 4.0$ with reluctant consumers. In this case, their bid value is the lowest – indicating that users are unwilling to use up their tokens in exchange for using the shared resources. $p = 40.0$ and $p = 400.0$ are associated with competitive consumers. They are characterized by leveraging a fraction of their organization's earnings to make use of shared resources. At $p = 4000.0$, consumers adopt an aggressive strategy because the bid value assigned exceeds the maximum full-load earnings by about 377%. This consumer strategy therefore simulates the absence of the admission control facility because this bid value is greater than any admission price regardless of the degree of resource contention. We therefore, employ

this consumer strategy to study the effectiveness of the pricing framework on the admission control performance.

We obtain the bid price $p = 400.0$ by dividing the maximum full-load earning (1060) by the the average submission rate which is about 3 requests per hour. This gives a bid value of 353.33, which is then rounded to 400.0. To obtain $p = 40.0$, we divide the minimum full-load earnings with the same average submission rate to obtain 41.7, which is then rounded to 40.0.

- System Admission Ratio

It is the ratio between the total number of admitted requests divided by the total number of submitted requests. In the requirements set out in chapter 3, the primary objective of managing user-initiated contention is to maximize the total utility gained by all users. The total utility gained by users is represented by the total number of admitted requests. Similarly, the expected total utility is represented by the total number of submitted requests. Hence, the system admission ratio implicitly compares the total utility gained by users against their total expected utility.

- Average Delay

Measures the average time interval between the $admit(r_i)$ and $start(s_j)$, for a given service s_j and its corresponding request r_i . (Note that, the delay is also the slack time, $T_{i,j}^{slk}$, already defined in chapter 2).

4.5.3 Benchmarks

To compare the performance of resource allocation using the proposed non-preemptive admission control scheme, we use the Modified Slack Based (MSB), Modified Real Time (MRT) and the QoS-based (QoPS) scheduler designed and evaluated by Islam et al. [49]. These are preemptive admission control schemes that are traditionally applied to manage real-time multi-processors. As already mentioned, since the preemptive algorithms have

less scheduling constraints, we use them as benchmarks to evaluate the performance of the proposed admission control scheme.

The MSB is an extension of the Slack-based algorithm originally proposed by Talby et al. [82]. The idea behind this algorithm is that each request is assigned a slack. This real time algorithm allows all pending requests to be delayed to fit in any newly arriving requests as long as the pending services of the admitted requests remain in the order of the time they are first scheduled and their deadlines are not violated. The MRT algorithm is likewise, an extension of the real-time algorithms proposed by Ramamritham et al. [69]. The scheduler tries to meet the deadlines of parallel requests by applying heuristics functions to re-schedule pending requests. QoPS is an extension of the MSB algorithm. It also captures the characteristics of the MRT algorithm but allows only partial re-scheduling of pending requests.

4.5.4 Simulation Framework

The simulator proposed in chapter 2 is used to evaluate the performance of the non-preemptive admission control scheme. Each provider agent also implements all the above preemptive algorithms. To achieve consistency in performance evaluation of each algorithm, we isolated the resource allocation mechanism (i.e., the *Scheduler* module) from the admission control mechanism. The resource allocator finds and assigns CPUs to the admitted requests on each organization's underlying physical resources managed by the provider agent. Each simulation run involves the specification of a workload trace, an admission control scheme, the LC ratio and a consumer strategy. For each run, the simulation duration is set to 20,000 hours. Depending on the workload trace, each contains between 150,456 to 160,223 requests. For requests that demand an execution time exceeding 50hrs, the maximum allowable slack ratio assigned is 0.2 and otherwise, 0.0 is assigned. The time step of the scheduler, Δt , is fixed at 5 minutes. For the pricing framework, the value assigned to ω is the least amount of computing resources required by all the requests contained in a set of workload traces in Table 4.3. τ is assigned the

mean execution time length of the workload traces.

4.6 Results and Discussion

4.6.1 Impact of Consumer Strategy on the System Admission Ratio

The consumer strategy determines the extent at which large requests are successfully admitted to shared resources. As such, we can assess how large requests, when admitted at different degrees, can on overall, affect the system admission ratio. We analyze the performance of admission control under different degrees of contention and workload characteristics. Hence, we compare the results against the benchmarks by varying the load-contribution ratios for the different modified workload traces. In terms of time scale, we limit our analysis to workload traces that are labelled Skew-02.

Figures 4.5, 4.6 and 4.7 show the results for the impact of consumer strategy on the admission ratio of the proposed admission control schemes with respect to the benchmarks based on the modified workload traces. UTIL-0 gives the results for the admission control scheme without the application of time delay. As for UTIL-MIX, time delay is applied to requests exceeding 50Hrs of execution time with a maximum allowable slack ratio of $\alpha(r_i) = 0.2$ for all requests.

We examine the impact of consumer strategy on the admission ratio under different degrees of contention by varying the load-contribution ratio. First, we look at the results for reluctant users ($p = 4.0$). Because a large number of bids cannot exceed the admission priority generated by the pricing framework, the system admission ratio is very low regardless of the LC ratio because requests are unnecessarily rejected even when there are sufficient computing resources. The system admission ratio for both UTIL-0 and UTIL-MIX are very close when consumers have low bids. This is because when bids are low, a majority of large requests² are rejected. As such, the impact that time delay have

²We use the term ‘large request’ to denote requests that require large amounts of CPU and long

Figure 4.5: Effect of Consumer Strategy on System Admission Ratio with Workload KTH SP2 for (1) $p = 4.0$, (2) $p = 40.0$, (3) $p = 400.0$ and (4) $p = 4000.0$

(1)

(2)

(3)

(4)

Figure 4.6: Effect of Consumer Strategy on System Admission Ratio with Workload CTC SP2 for (1) $p = 4.0$, (2) $p = 40.0$, (3) $p = 400.0$ and (4) $p = 4000.0$

(1)

(2)

(3)

(4)

Figure 4.7: Effect of Consumer Strategy on System Admission Ratio with Workload SCSD Par95 for (1) $p = 4.0$, (2) $p = 40.0$, (3) $p = 400.0$ and (3) $p = 4000.0$

on the system admission ratio is relatively minimal. At higher LC ratios (1.6 – 2.0), with the exception of KTH SP2 trace (Figure 4.5(1)), both schemes (UTIL-0 and UTIL-MIX) perform either better or within the benchmarks. In some cases, for example SCSD Par 95, the system admission ratio is better than the benchmarks at $LC = 1.6$, when large requests are sacrificed for a significantly larger portion of smaller requests. The admission control becomes effective because there is a critical need to restrict large requests from gaining access when there is a high degree of contention. Compare to the benchmarks, *UTIL – 0* and *UTIL – MIX* are less affected by the load-contribution ratios.

Competitive consumers bid at a range proportional to their earnings (p=40.0 to p=400.0). The system admission ratio is generally better than reluctant consumers because the opportunity to admit requests increases, and it therefore leads to better system admission ratios. At a higher degree of contention, that is, when LC ratios is between 1.6 and 2.0, the system admission ratios for all workloads are still within the benchmarks despite the fact that large requests are now admitted. At LC ratios between 0.8 to 1.2, the system admission ratios are either within (KTH SP2) or above (both SCSD Par 95 and CTC SP2) the benchmarks. When $LC = 0.4$, the system admission ratios converge to towards the admission performance of the benchmarks. It can also be seen that the system also achieves gradual improvement in system admission ratio with respect to LC. In contrast with the benchmarks (e.g., CTC SP2 and SCSD Par 95), the system admission ratio results are more consistent with change in LC ratios. The discrepancy in system admission ratio performance between UTIL-0 and UTIL-MIX also becomes more prominent with higher LC ratios.

With aggressive consumers, requests are admitted on a first-come-first-serve basis as long as there are available resources because the bid value assigned is likely to exceed the admission price. At higher LC ratios (1.6 – 2.0), the results are better or within the benchmarks, but are however weaker than the competitive case. At mid-range load-contribution ratios (between 1.2 – 1.6), the system admission ratio deteriorates below the execution times.

benchmarks in some cases. Within this range of load-contribution ratios (that is, with the increased amount of resources), the negative effects of admitting very large requests manifests when a large portion of smaller requests fail in gaining admission to the shared resources. Without any chance for rescheduling, the system admission ratios are therefore lower than the benchmarks. Another noticeable feature is that when consumers are aggressive, the system admission ratio performance, on overall, matches closely with the benchmarks.

4.6.2 Effect of Workload Traces on System Admission Ratio

In this subsection, we evaluate the performance of the admission control framework using workloads that differ in their statistical properties. The requests in the workload traces are different in terms of CPU requirements and their inter-arrival distribution. Time scaling is further introduced to modify the execution time distribution of each workload trace. The results in Figures 4.8, 4.9 and 4.10 are graphed by taking the average value of the system admission ratio resulting from the variety of consumer strategy classes discussed earlier. We compare the average system admission ratio result against the benchmarks for different LC ratios and time scales. As given in Table 4.3, Skew-02, Skew-01, and Skew-005 represent different scaling factors that are applied to the execution time of the requests contained in the synthetic workload traces. For Skew-02, Skew-01 and Skew-005, the execution time of each request is divided by 0.2, 0.1 and 0.05 (or multiplied by 5, 10 and 20) respectively.

In order to observe the resilience of each admission control policing, we evaluate the consistency of the average system admission ratio results based on different workload traces. For each admission control scheme, we measure consistency for a given time scale and a LC ratio as follows:

$$\frac{|AR_{CTC} - AR_{PAR95}| + |AR_{KTH} - AR_{PAR95}| + |AR_{CTC} - AR_{KTH}|}{3} \quad (4.6.1)$$

The equation measures the overall discrepancy in the average system admission ratio for

(1)

(2)

(3)

Figure 4.8: Effect of Skew-02 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$

(1)

(2)

(3)

Figure 4.9: Effect of Skew-01 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$

(1)

(2)

(3)

Figure 4.10: Effect of Skew-005 on System Admission Ratio for (1) $LC = 0.4$, (2) $LC = 1.2$ and (3) $LC = 2.0$

different workload traces. AR_{KTH} , AR_{CTC} and AR_{PAR95} represent the average system admission ratio results of different consumer strategies for a respective workload trace (as labelled), given a time scale and a LC ratio. Lower values therefore, reflect better consistency in the average system admission ratio.

The average consistency for all the preemptive schemes (or benchmarks), together with the non-preemptive schemes (UTIL-0 and UTIL-MIX) are presented in Table 4.4. In general, the admission control framework based on the non-preemptive scheme produces more consistent results, especially when comparing between different workload traces. However, the consistency also deteriorates as the LC ratio and time scale is increased.

When LC=0.4, it is noted that the average system admission ratio for both UTIL-0 and UTIL-MIX are consistent for different workload traces regardless of the amount of time scaling introduced. However, they give average poorer system admission ratios than the benchmarks. This is largely attributed to the results of reluctant consumers as presented in the earlier subsection. Particularly, under skew-01 and skew-005, as shown in Figure 4.9(1) and 4.10(1), substantial inconsistency in performance results can be observed (in contrast with UTIL-0 and UTIL-MIX) for all the benchmarks when different workloads traces are applied.

When LC=1.2, as shown in Table 4.4, the contrast in performance results, in terms of consistency, between the benchmarks and the non-preemptive admission control schemes becomes more prominent as time scaling is increased. At skew-005 (Figure 4.10(2)), the system admission ratio for both UTIL-0 and UTIL-MIX begins to exceed some of the benchmarks quite significantly for all workload traces. When compared with UTIL-0, UTIL-MIX gives the same or better results in terms of both consistency and system admission ratio in all circumstances.

At LC=2.0, with exception of skew-005, the performance of both the benchmarks and the proposed admission control schemes have about the same system admission ratio. The consistency in resource system admission ratio remains relatively resilient for both UTIL-

Time Scale	LC Ratio	Admission Control Policy	Consistency
Skew-02	0.4	Preemptive Avg.	0.032
		UTIL-0	0.020
		UTIL-MIX	0.023
	1.2	Preemptive Avg.	0.023
		UTIL-0	0.013
		UTIL-MIX	0.033
	2.0	Preemptive Avg.	0.064
		UTIL-0	0.086
		UTIL-MIX	0.023
Skew-01	0.4	Preemptive Avg.	0.048
		UTIL-0	0.013
		UTIL-MIX	0.016
	1.2	Preemptive Avg.	0.180
		UTIL-0	0.063
		UTIL-MIX	0.016
	2.0	Preemptive Avg.	0.103
		UTIL-0	0.010
		UTIL-MIX	0.100
Skew-005	0.4	Preemptive Avg.	0.200
		UTIL-0	0.013
		UTIL-MIX	0.013
	1.2	Preemptive Avg.	0.210
		UTIL-0	0.064
		UTIL-MIX	0.016
	2.0	Preemptive Avg.	0.096
		UTIL-0	0.033
		UTIL-MIX	0.041

Table 4.4: Results for Consistency in System Admission Ratio for Different Time Scale and LC Ratio

0 and UTIL-MIX. At skew-02 (Figure 4.8(3)) and Skew-01 (Figure 4.9(3)), the system admission ratio performance for UTIL-0 and UTIL-MIX closely match the benchmarks when compared based on different workload traces. As for skew-005 (Figure 4.10(3)), both UTIL-0 and UTIL-MIX perform better than the benchmarks in almost all the cases – the difference in the results is especially distinct for both CTC SP2 and SCSD Par 95.

The system admission ratio performance of UTIL-MIX is the same or better than UTIL-0 in all cases. However, improvement to the system admission ratio by introducing time delay to requests (UTIL-MIX) is only effective at higher LC ratios and only when the requests have longer execution times. As evidenced by Figure 4.9(2) and Figure 4.10(2), we show that when $LC=1.2$, both UTIL-0 and UTIL-MIX exhibit better consistency relative to the benchmarks. However, when $LC=2.0$, as observed from Figure 4.9(3) and Figure 4.10(3), the discrepancy in system admission ratio performance between the two schemes becomes larger in some cases (e.g., KTH SP2) but less so in others (e.g., CTC SP2 and SCSD Par 95).

4.6.3 Effect of Consumer Strategy on Average Delay

Consumer strategy has an influence on the amount of large requests that are admitted. The application of time delay is purposed to allow admitted requests with shorter execution times to fill-in so that the overall admission ratio can be improved. Doing this, however, can cause unnecessary waiting for services to begin executing especially if there are already sufficient resources to be catered to the requests with shorter execution times.

Hence, in this section we first study the impact of consumer strategy on the overall average delay of all admitted requests. Since any delay assigned to a request is generally undesired, we then make use of these results to study the trade-off between the marginal improvement on the system admission ratio and the resulting average delay on admitted requests.

We compare the performance results between the benchmarks and the proposed ad-

mission control scheme with different LC ratios and consumer strategy. Figures 4.11, 4.12 and 4.13 give the results on the average delay of the benchmarks and UTIL-MIX. It can be observed that consumer strategy has a strong influence on the average delay especially for UTIL-MIX. At $p = 4.0$ (reluctant consumers), the average delay is less than one hour for all workload traces regardless of execution time scale. At $p = 40.0$ (competitive consumers), the average delay for skew-02 ranges from 0.83 Hrs to 1.7 Hrs and for skew-01, it ranges between 6.5 – 9.0 Hrs for all workload traces. For skew-005, the range is 6 – 12 Hrs. Since the delay assigned to requests with longer execution times is proportional to their demanded execution time, the average delay also increases especially with higher likelihood of contention (i.e., high LC-ratios). When $p = 4000.0$, the average delay is almost two-fold when compared with $p = 40.0$ in most cases. When aggressive strategy is permitted, the opportunity for larger requests to gain admission increases. Hence, if requests with long execution times are consequently admitted, their execution contributes considerably to the increment of the delay of other requests.

The average delay of UTIL-MIX matches more closely to benchmarks for the competitive strategy (i.e., $p = 40.0$)³ mainly because requests with very large execution times are not admitted. Furthermore, for workload traces with a shorter execution time distribution (skew-02), the average delay of UTIL-MIX is relatively close to the benchmarks. This is because the execution starting time delay introduced by UTIL-MIX is significantly lower. For the benchmarks, requests with longer execution times are delayed only if their latest starting time are not exceeded when requests with shorter execution times are allowed to execute first. As such, their overall average delay for different LC-ratios are better than UTIL-MIX. An evaluation on the output trace from the simulations when the proposed schemes are used showed that the largest request for all three workloads has 74 hours of execution time. On the contrary, traces for the benchmarks (MSB, MRT and QOPS) show that the largest request has 210 hours of execution time. Requests with 74 hours execution time and above constitute 6% to 9% of the successfully admitted

³We do not show the results for $p = 400.0$ because they are close to the results for $p = 40.0$

requests.

For the competitive strategy, the results for average delay are more consistent with different LC-ratios when compared with the benchmarks. This shows how the proposed admission control scheme gives predictable system performance if there are changes in the community's overall contribution of shared resources especially with likelihood that resource contention will occur. The downside is the undue penalty of introducing starting time delay on longer requests even if there are sufficient resources. When consumers are aggressive ($p = 4000.0$), the average delay remains consistent except for workloads with short requests (skew-02). This condition manifests because when large requests are assigned delays, the sum of their delays contribute a large value to the calculation of the average delay. The discrepancy in system admission ratio performance is therefore influenced by the number of large requests, which vary with respect to the LC ratio and workload statistics.

To assess the benefit of incorporating delays to requests with long execution times so as to improve the system admission ratio, we examine the gain in system admission ratio over the overall increase in average delay, comparing UTIL-MIX with UTIL-0 scheme. The marginal improvement in the system admission ratio over the average delay for different time scales and LC ratios is calculated using the following equation:

$$TF = \frac{(AR_{UTIL-MIX} - AR_{UTIL-0}) \times 100}{WT_{UTIL-MIX}} \quad (4.6.2)$$

Both $AR_{UTIL-MIX}$ and AR_{UTIL-0} are the average system admission ratio for UTIL-MIX and UTIL-0 schemes respectively, based on all consumer strategies and workload traces for a given time scale and LC ratio. $WT_{UTIL-MIX}$ is the corresponding average delay of UTIL-MIX.

With reference to Figure 4.14, for most of the time, TF improves as the LC ratio is increased. It also true that improvement to the system admission ratio based on UTIL-MIX (i.e., larger value for TF) is also more prominent when the workload traces have

(1)

(2)

(3)

Figure 4.11: Effect of Skew on Average Delay for KTH SP2 Workloads

(1)

(2)

(3)

Figure 4.12: Effect of Skew on Average Delay for CTC SP2 Workloads

(1)

(2)

(3)

Figure 4.13: Effect of Skew on Average Delay for Par95 Workloads

Figure 4.14: Graph of Performance Trade-off Between UTIL-0 and UTIL-MIX

relatively shorter execution times (i.e., with larger time scale value).

From Figure 4.14, we observe that when LC ratio is 0.4, TF is zero for different time scales. This is because the total contribution of resources exceeds the peak workload conditions resulting in negligible difference in the performance of the system admission ratio between UTIL-0 and UTIL-MIX. For skew-005, the presence of requests with longer execution times increases the likelihood of delays assigned to admitted requests. As such, the improvement in the system admission ratio with respect to the average delay becomes inconspicuous. The benefit of introducing delays exists when the LC ratio becomes higher – especially when a large proportion of the requests have relatively shorter execution times (e.g., skew-02). However, the application of time delay is obviated when a large proportion of requests have long execution times (e.g., considering the case of LC=1.2 with different time scale values). This is because delays may unnecessarily be assigned to large requests without having the opportunity to fill-in the requests with shorter execution times, resulting in a large increase in the average delay without any marginal improvement

to the system admission ratio.

4.7 Conclusions

Intermittent contention on shared resources is anticipated because in a sharing environment, we expect organizations to limit their contribution of resources with the primary intention of reducing their investment on computing resources. As such, an admission control mechanism is needed to minimize the negative effects due to resource oversubscription by users. The major constraint in admission control is that, for efficiency reasons, requests are handled FCFS and their quality-of-service level agreements must be non-preemptive. This problem is addressed by employing a pricing framework as a means of regulating request admissions. We examined the performance of the proposed admission control scheme with the pricing framework under different workload and resource contribution scenarios against a set of benchmarks.

First of all, the experiments highlight that the proposed admission control schemes (both UTIL-MIX and UTIL-0) are able to better achieve consistent performance in terms of system admission ratio. However, because consumer strategy has a major impact on the performance consistency, it is essential for each consumer agent to adopt a competitive strategy – user’s bids are generally limited to a fraction of the organization’s current earnings (or, in the same sense, the current supply of tokens). We will demonstrate in chapter 6 that the token-exchange incentive scheme is able to achieve this requirement autonomously. This is because expenditure of tokens is linked directly to an organization’s ability to earn tokens as a result of resource sharing. We further show that, the maximum quantity of tokens that is earned by an organization is limited and proportional to the resource contribution. In order to maintain a consistent supply of tokens, the consumer agents of each organization must employ the competitive strategy when assigning tokens to each request.

The main weakness with the benchmarks lies with their inability to progressively

block extremely large requests. As a consequence, despite their advantage of being able to achieve better utilization of shared resources due to the opportunity to re-schedule pending requests, they may still perform poorly in perverse conditions of resource over-subscription by users. In contrast, the advantage of employing the proposed admission control schemes lies with their ability to control the rate of admitting very large requests. The end effect is better consistency in system admission ratio performance regardless of different LC ratios and workloads.

Assigning time delay to requests with long execution times (i.e., UTIL-MIX scheme) can play a supportive role in improving the average system admission ratio when consumers are competitive or aggressive. However, the benefit accrued from introducing delays to improve the system admission ratio has to be traded-off with the possibility of introducing unnecessary waiting time to users' requests even when there is no contention.

To summarize, the pricing framework not only helps in achieving better system admission ratio under contention situations, but facilitates in producing consistent resource management performance as well. By incorporating delays to requests with large execution times, in addition to the pricing framework, the system is capable of achieving even better system admission ratio performance. This is especially true when the LC ratio is high and a majority of requests have relatively shorter execution times.

Chapter 5

A Preliminary Analysis on Inter-domain Resource Contention

5.1 Introduction

In the case of user-initiated contention, users who cause contention can come either all from the same or from different organizations. We now focus our attention on resource contention amongst different organizations. This means the aggregate workloads of users becomes a source of resource contention rather than individual users. In this case, organizations may have to compete for shared resources if the aggregate workloads of their users require more resources than their contributions. When all organizations concurrently require more resource than their contributions, a perverse situation, which we term as inter-domain contention, will result. It must be addressed by an VO-wide admission control facility. In terms of the requirements set out in chapter 3, equitable resource usage is achievable by keeping the ratios of aggregate admitted workloads and the resource contribution of an organization below a pre-defined threshold.

In this chapter, we experimentally highlight the importance of keeping the load-contribution ratio of each organization at a consistent level in the event of inter-domain contention. To do so, we evaluate the system admission ratio, that is, the ratio between successfully admitted requests and the total number of requests submitted to the shared

resources that are federated and managed directly by a central broker. As we do not yet employ the token-exchange incentive scheme, the objective is to reveal the potential difficulties of employing a centralized scheme to manage inter-domain contention in this chapter.

To simulate inter-domain contention, we employ the fast fractional gaussian (FGN) model to generate the arrival profile of requests for the entire community of users. We use the FGN (instead of those from real supercomputer workload traces as used in chapter 4) in order to characterize different degrees of resource contention on the basis of correlations in request arrival time. The correlations in arrival times of requests simulate the scale of concurrent resource usage amongst all organizations. Doing so, we can better evaluate the impact of inter-domain resource contention on resource management. In particular to the centralized approach, we observe the size of the broker's queue on three parameters: the load-contribution ratio of each organization, degree of participation and the extent of correlations in request arrivals for all organizations.

The initial set of experiments confirm the importance of maintaining the load-contribution ratio of all organizations at a specified level especially if the degree of participation increases. We then show that even if slack is introduced to the execution schedule, the improvement in system admission ratio is not effective if the degree of contention due to the correlation of request arrivals and lack of the computing resources remain high. This further highlights the importance of admission control on the broker's queue. Admission control involves a combined policy to keep the load-contribution ratio of all participating organizations constant while keeping the queue size below a certain threshold. We show that the system admission ratio becomes increasingly inconsistent when workloads with differing statistical characteristics are introduced.

This chapter is organized as follows: In section 5.2, we introduce the experimental framework employed to assess the impact of various parameters that lead to inter-domain contention. In section 5.3, a theoretical study on the impact of the load-contribution ra-

tio on request admission management is presented. We first compare the impact of the load-contribution ratio on the system admission ratio with respect to the degree of participation. Then, we introduce the allowance of delay to requests in the event of contention to observe the improvement on the system admission ratio. In section 5.4, we then present a centralized approach for admission control to counteract the negative effects of queue storms on resource management. The framework for request management will also be presented in this section. It explains the details of the various sub-modules that make up the centralized admission control mechanism to address inter-domain contention. In the last section, we present the summary and conclusions of this chapter.

5.2 Inter-domain Contention on Shared Resources

As discussed, the key motivation behind addressing inter-domain contention is the likelihood of correlations in arrival times of requests submitted by users in the entire community. This phenomena was originally investigated by Kleban et al. when they observed how concurrent request submissions that occur intermittently, can cause the broker's queue of a set of computing resources grows to an extent that the underlying scheduling mechanism is unable to cope [55]. This case becomes even more important in a VO since organizations need to leverage on each others computing resources so as to improve their overall usage capacity when needed. In the event that all organizations are competing for computing resources, the admission control mechanism must intervene so that the admitted workload of each organization is proportional to their resource contribution.

We first introduce the basic elements that are employed in the experiments to validate the significance of regulating the load-contribution ratio of all organizations. This is because it will be translated into a policy that will be exploited to manage inter-domain contention.

5.2.1 Request Submission

The frequency of request arrivals was simulated based on the fast fractional gaussian noise generator (FGN). This mathematical workload has been used to model long-range dependencies on job inter-arrival times on a variety of supercomputers [55]¹. FGN is modelled using the synthesis of a high frequency Gaussian-markov process and a low frequency Gaussian process. The parameter that dictates the extent of request arrival correlations resulting from long-range dependencies is the H or ‘Hurst’ parameter. The value of H ranges between 0 and 1. A higher value of H reflects a higher degree of correlations in request arrivals.

We first use the FGN workload model to obtain the community workload intensity profile. The community workload profile is essentially a trace indicating the degree of competition for resource at specific time intervals. The normalization is performed by dividing each value (representing the total number of arrivals for a given time epoch) with the highest number of request arrivals for all values of H between 0.6 to 0.8. The community workload intensity profile defines the degree of resource competition of each organization.

We investigate the scenario where all organizations concurrently compete for the same amount of resources – which, as we identified is the cause of inter-domain contention. This scenario is simulated by generating an equal number of requests for each organization for a given epoch t . The equation $n(t) = w_H(t) \times L \times N$ denotes the total number of requests that is submitted by users belonging to all organizations, at epoch t . $0 \leq w_H(t) \leq 1$ is the community workload intensity profile² for a given H value. L is the peak resource demand (number of requests) of each organization and N represents the total number of participating organizations. The requests generated at each epoch are divided equally into N partitions, one for each organization, and are labelled according to their respective partitions. For each epoch, the requests from each organization are shuffled to randomize

¹Refer to Appendix B for the formulation of the FGN workload model

²The derivation of the community workload profile is shown in Appendix B

Figure 5.1: Schematic of the Simulation Framework for Centralized Policing Scheme to Address Inter-domain contention

the order at which they are submitted to the broker.

5.2.2 Simulation Model

The diagram in Figure 5.1 shows a schematic of the simulator designed for the following experiments in this chapter. It is a simplified version of the simulation model discussed in Chapter 2 because emphasis is given to the design of centralized policies at the broker to perform admission control. Because workload trace is built to represent the entire community of users, requests are submitted to the broker without the use of the consumer agents. Requests submitted to the broker are first placed in the broker's queue. They are then serviced by the broker using a first-come-first-serve (FCFS) policy. The broker in turn, submits the requests to a resource manager using the round-robin policy. Each organization's resource manager is responsible for allocating resources to the request it receives as long as there are available computing resources. Otherwise, the broker will be informed so that another resource manager can be selected.

5.2.3 Queue Storm

A queue storm is a condition when the broker's queue becomes excessively large because it is unable to service the request effectively due to a lack of available computing resources. From a queueing theory perspective, this happens when the arrival rate of requests exceeds the overall servicing rate of the system. With respect to the workload profiles, the arrival rates are not stochastically stable due to the presence of correlations in request arrivals. As such, queue storms occur in bursts, each of which lasts for a period time. However, the size and duration of a queue storm is strongly influenced by the resource contributions of each organizations. We therefore measure the size of queue storms to assess the level of inter-domain contention between participating organizations.

5.3 Impact of Load-contribution Ratio on Inter-domain Contention

The load-contribution ratio is basically a number indicating the relative size of workload each organization generates with respect to its contribution. As mentioned, in a centralized system, all requests submitted by users arrive at the broker's job queue. Hence, we associate the degree of inter-domain contention with the size of a queue storm.

In the first experiment, we first observe the effect of each organization's load-contribution ratio and the degree of participation (number of organizations) on the occurrence of queue storms. In the second experiment, we further demonstrate that even with the introduction of slack or delay time on admitted requests to reduce the size of queue storms, the load-contribution ratio still has a greater impact on resource overload.

Before presenting the results, we first introduce the parameters and performance metrics that are used throughout the experiments in this section.

VO Parameters

The following parameters are used in the analysis:

- N – Represents the degree of participation or the total number of organizations to be simulated.
- LC – The load-contribution ratio of each organization, is the ratio between the size of average number of request arrivals per organization (L) and the total number of computing nodes contributed by each organization (C). Again, in order to examine the situation where all organizations concurrently compete for shared resources, we assign the same LC ratio to each organization. Similar to the experiments in chapter 4, we use LC to calculate C by dividing L with LC .
- H – This value is proportional to the extent of request arrival correlations. We choose two distinct values ($H = 0.6$ and $H = 0.8$) for the experiments. As shown in Figure 5.2, $H = 0.6$ exhibits less correlations in request arrivals when compared with $H = 0.8$. Notice that when $H=0.6$, the profile resembles a white noise function. When $H=0.8$, strong correlations in request arrivals is observable. An example is the time interval between 400 and 620 as shown in Figure 5.2. We see a gradual increase in the degree of competition or intensity of request arrivals for $H = 0.8$. In such conditions, the broker's queue may be overwhelmed with unscheduled requests – thus resulting in a queue storm. The correlations are contributed by the low frequency component that is used to build the FGN workload trace.
- T_s – Defines the total number of simulation intervals, where each interval is assumed to be 1 day. T_s was set to 1000 for all experiments.

Request Size

Request size is a measure of the request's execution length multiplied by the number of computing nodes required. In the simulations, each request is assumed to require one

(1)

(2)

Figure 5.2: Community Workload Intensity Profile for (1) $H=0.6$ and (2) $H=0.8$.

CPU. The length of requests were randomly generated based on a normal distribution with a mean value of 3 days with standard deviation of 2 days. This statistical profile of the workload trace was the same as that used by Kleban et al. [55].

Performance Measures

- Queue Storm (Q)

A queue storm is a value that is obtained from a Cumulative Distribution Function (CDF) of the broker's queue length, q , for an entire simulation run. This value is expressed as $q > Q$ at $p = 5\%$. This expression accounts for queue lengths q that are greater than Q at 0.05 probability for the entire simulation run.

- System Admission Ratio

It is the ratio of the number of requests that meet their required execution deadlines and the total number of submitted requests.

- System Admission Failure Ratio

It is the ratio of the number of requests that do not meet their required execution deadlines and the total number of submitted requests. It can also be computed by taking the difference between 1 and the system admission ratio

5.3.1 Analysis of VO Parameters on Queue Storms

The three VO parameters considered are the load-contribution ratio of each organization, the degree of participation and the correlation in inter-arrival times of requests within each organization. In order to analyze the impact of the VO parameters on queue storms, we measure the size of queue storms for different values of N , LC and H . Table 5.1 gives the size of queue storms for each combination of load-contribution ratio and number of participating organizations. It can be observed that when $H = 0.6$, there is a gradual increase in queue storm if N is increased. However, as LC is increased, the size of queue storms become very large with increased participation. When $H = 0.8$, the size of queue

storms are even larger when LC is increased when compared with $H = 0.6$. From the results, LC has the most profound impact on inter-domain contention.

For further illustration, we make use of the results in Table 5.1 to obtain MRC_N and MRC_{LC} . Figure 5.3 shows the results of the analysis. MRC_N and MRC_{LC} measure the change of Q with respect to N and LC respectively, for two different degrees of workload burstiness (i.e., $H = 0.6$ and $H = 0.8$). We can observe from MRC_N that provided if all organizations are consistently cooperating to contribute resources at a given LC , the change of queue storms would first increase to a certain peak value before dropping to a very low value when N is further increased. This means that once the level of participation (N) exits the critical point, further increment in participation does not degrade the performance of the system. Also, notice that with improved load-contribution ratios (smaller values), this transition becomes less prominent.

Figure 5.3 also shows the change of queue storms for MRC_{LC} by varying LC . It can be seen that if all organizations (at worst case) were to increase their LC ratios, that is to say, if they were to either lower their contributions or increase their average loads alone, then the change by queue storms would increase considerably. It is especially so if N is large, the contrast in performance between $H=0.8$ and $H=0.6$ also becomes more prominent. MRC_N on the other hand shows that as N increases to a large value, the size of queue storms would not increase indefinitely. Therefore, the size of a queue storm is influenced more significantly by each organization's LC ratio than any of the other VO parameters. If unacceptable queue storms subsequently do occur due to changes in average load for some organizations, especially when N or H increases, it is necessary to limit the LC of each organization.

Figure 5.3: Marginal Effect of N (MRC_N top) for $H=0.6$ (left) and $H=0.8$ (right) and the Marginal Effect of LC (MRC_{LC} bottom) for $H=0.6$ (left) and $H=0.8$ (right) on Queue Storms

Table 5.1: The Effect of Varying N and LC on Queue Storms

$q > Q$ at $p = 5\%$, $H = 0.6$									
LC	$N=2$	$N=4$	$N=6$	$N=8$	$N=10$	$N=12$	$N=15$	$N=20$	$N=30$
1.2	1	2	2	4	5	6	7	9	11
1.3	1	3	5	9	11	12	15	16	18
1.4	1	4	10	15	20	23	30	36	40
1.5	3	9	18	22	35	42	60	72	80
1.6	3	12	40	65	90	130	140	151	159
$q > Q$ at $p = 5\%$, $H = 0.8$									
LC	$N=2$	$N=4$	$N=6$	$N=8$	$N=10$	$N=12$	$N=15$	$N=20$	$N=30$
1.2	1	2	4	7	9	11	14	15	17
1.3	2	5	10	12	20	25	38	43	52
1.4	2	8	18	30	45	50	71	82	102
1.5	6	23	50	78	120	140	170	245	321
1.6	7	50	150	270	380	480	550	605	665

5.3.2 The Effect of Introducing Delay to Requests and its Impact on Queue Storms

In this experiment, we explore the benefits of assigning delay to the service instance of each request's starting time in order to improve the system admission ratio in the presence of queue storms. Allowing delay gives additional time for a request to remain in the queue, and thus, increases its chance of being scheduled for execution by the resource management system during contention.

We use a slack tolerance factor (STF)³ to denote the maximum allowable delay on a request when it is admitted by a resource manager (provider agent). We varied STF from 0.1 to 0.5 for both $H=0.6$ and $H=0.8$. We fixed LC at 1.54 so that the resulting queue storm Q during each simulation run, would vary smoothly between 20 and 160 when N increased gradually from 2 to 12. For each queue size and value of STF, the system admission ratio was measured.

The effectiveness of introducing slack is significant only for larger queue sizes when $H=0.6$. Referring to Figure 5.4, it can be observed that when the queue size is increased

³STF is the same as $\alpha(r_i)$, the maximum allowable slack ratio of on request r_i as defined in chapter 2.

Figure 5.4: Impact of Slack on System Admission Ratio in Presence of Queue Storms for $H=0.6$ and $H=0.8$

Table 5.2: A Measure of Significance of Slack on Queue Storms

Queue Storm (Q)	Significance	
	H=0.6	H=0.8
20	0.25	0.25
40	0.9	1.125
65	0.95	1.0
90	0.93	0.94
130	1.0	0.75
160	1.1	0.58

from 20 to 130, the system admission decreases from 0.9 to 0.55 when STF is 0.1. In contrast, when STF is 0.5, deterioration in system admission ratio is much less as the size of queue storms is increased. When $H = 0.8$, we get similar results except that, when the size of queue storms is increased, the results for $STF = 0.5$ deteriorates quite considerably.

We now analyze the significance of improving the system admission ratio as a result of introducing slack to requests for different degree of queue storms. We introduce another performance measure, which is termed as significance, by taking the ratio between the largest difference in performance of system admission ratio and the range of STF, for a given queue size. It accounts for the improvement to the system admission ratio for every unit increase in STF. It is computed as follows:

$$\frac{\max(JS(Q)) - \min(JS(Q))}{STF_{\max} - STF_{\min}} \quad (5.3.1)$$

$JS(Q)$ returns the system admission ratios for a given queue size, $STF_{\max}$ is 0.5 and $STF_{\min}$ is 0.1. The results for both $H = 0.6$ and $H = 0.8$ is shown in Table 5.2. Larger values indicate that introducing slack to requests contribute to greater improvement to the system admission ratio.

When $H = 0.6$, the effectiveness of assigning delays to requests is significant with larger queue storms ($Q \geq 130$). For the case of $H = 0.8$, the significance of assigning delays can be divided into three parts. First, when the queue storm size is less than

20, the significance is 0.25, giving a value as low as the case for $H = 0.6$. When queue storm size is between 40 and 90, we can observe that the significance improves tremendously and even outperforms $H = 0.6$. This is a critical region at which the assignment of STF can have a significant impact on the system admission ratio. Finally, when queue storm size is greater or equal to 130, the significance dropped to about 0.75. The cause of this drop is largely due to the existence of unusually large spikes in the arrival of requests, thus causing a larger proportion of requests to miss their execution deadlines since they cannot be admitted. When $H=0.8$, if the queue size is large, then the system admission ratio performance remains poorer than $H = 0.6$ ($Q \geq 130$) regardless of the amount of slack introduced.

If inter-domain contention occurs as a result of correlations in request arrivals, assigning delays alone may not be effective in dealing with queue storms. From the experiments, we can observe that even if each request takes up only a single computing node and pre-emption⁴ to executing requests is allowed, inter-domain contention can still adversely affect the performance of the system.

5.3.3 Discussion

Amongst the various VO parameters, the load-contribution ratio (LC) has the biggest impact on the degree of inter-domain contention. Furthermore, contention becomes more severe with higher correlation in job arrival time. Merely increasing the allowable slack on jobs will have little improvement on the system admission ratio if the load-contribution ratio of each organization becomes very large. This points out the need for admission control to cope with overload conditions when organizations do not contribute sufficient resources to the community.

⁴For a concrete definition, refer to chapter 4 in page 68.

5.4 Admission Control on the Broker's Queue

For the time being, we disregard the use of the pricing framework to administer admission control in order to examine the critical difficulties of addressing inter-domain contention by employing a centralized admission control mechanism at the broker. A centralized approach requires a policy to adaptively control the size of the broker's queue so as to reduce the size of queue storms. To do so, we need to find an optimal threshold on the queue size. We expect this optimal value to exist because due to sudden surges in request arrivals, those with longer execution times, when admitted, may inadvertently slow down the servicing time of the system. Since requests cannot be delayed indefinitely, admission control on such requests may significantly improve the global performance of the system admission ratio. On the other extreme, there must be a limit imposed because otherwise, no requests will be executed. In the following experiments, we first demonstrate the existence of an optimal queue size when queue storms occur. We then explore the feasibility of the centralized request admission control policies on the system's performance.

5.4.1 Request Management Framework

Figure 5.5 presents the request management framework at the resource broker. It has three general levels of management. The bottom layer is the *Resource Allocation* module. This module handles each user's requests by submitting them to organization-level resource managers. Each resource manager that receives a request from the broker returns a candidate service. The *Resource Allocation* module uses the admission policy to choose a candidate service and assigns it to the current requests. The selected *Resource Manager* will then be informed to schedule and execute the request.

Above the *Resource Allocation* module, are the request admission control modules (surrounded by the dotted box). The *Load Management* module performs two functions. First, it controls the maximum permissible load (request submissions) by limiting the size

Figure 5.5: The Admission Control Framework for Centralized Policy

of the queue using the optimal threshold value obtained from the *Request Queue Threshold Control* module. The latter employs the *Request Property Monitoring* subsystem to compute the value of the optimal threshold for request queue size using the statistical properties of submitted requests at regular time intervals or epoches. The statistical properties of requests are calculated using the information compiled from the *Request Accounting Database*.

The *Load Management* module also employs the admission control policy to maintain equitable resource access by admitting requests as long as their organization's load-contribution ratios do not exceed the pre-defined threshold. The *Admission-contribution Monitoring Agent* is employed to supply the admitted workload and resource contribution information in order to calculate the load-contribution ratio of each organization. The admitted workload of each organization is measured by counting the total number of requests that are successfully submitted for a given epoch. Similarly, the resource

contribution of each organization is measured by counting the total capacity consumed by requests serviced in the same epoch.

5.4.2 Experiments

To demonstrate the existence of the optimal threshold on the broker's queue size, we varied LC between 1.2 to 1.6 and fixed $N=15$ for both $H=0.6$ and $H=0.8$. We choose $N = 15$ because at this level of participation, the MRC_N just crosses the critical point discussed in section 5.3.1. The results are shown in Figure 5.6. The y-axis presents the system admission failure ratio which quantifies the proportion of requests that either had not been allocated a resource or had failed to execute within their specified deadlines.

It can be seen that an optimal queue size exists between 50 and 60 for all values of LC and H because at this point, the system admission failure ratio is at the lowest value. This result is important since controlling the queue size can lead to marked improvements to the system admission ratio. The threshold is also independent of the VO parameters considered in the experiments. In other words, the optimal threshold for the broker's queue during contention does not depend on H , LC and N ⁵.

However, we noted that the threshold was instead influenced by the statistical properties of the request mix. We define the statistical structure of a request mix as follows:

$$(t_{avg}, \sigma_{t_{avg}}, STF_{avg}, \sigma_{STF_{avg}}) \quad (5.4.1)$$

t_{avg} and STF_{avg} are the average execution time and average allowable slack together with their corresponding standard deviations. In the experiments, we focus on the impact of STF_{avg} on the optimal queue size threshold. t_{avg} on the other hand, is dependant on the combined workload of users – which is likely to fluctuate over time. As such, we do not study t_{avg} influence on the queue size threshold.

⁵We do not show the results for varying N because is already obvious from the earlier experiments that LC has a much more profound impact on the size of the queue storms

Figure 5.6: Impact of Queue Size Threshold on System Admission Failure Ratio for different LC ratios for $H = 0.6$ (top) and $H = 0.8$ (bottom)

Figure 5.7 shows the results of the optimal threshold for different values of STF. Each value for the system admission failure ratio is computed as average from LC ratios ranging between 1.2 and 1.6 when $N = 15$. Note that the optimal threshold for different values of STF can vary quite substantially, ranging from about 55 when STF is 0.1 to 120 when STF is 0.5.

Figure 5.7: Performance of Queue Size Thresholding for Different STF

To cope with the resulting variations of optimal thresholds due to STF, we employed both static and adaptive thresholding for the admission control policies. In the static-based approach, we assigned the queue size threshold to be 60. With respect to Figure 5.7, this value gives the best average system admission ratio performance for the entire range of STF.

Due to the diversity of users' quality-of-service requirements, we anticipate the existence of fluctuations in the request mix characteristics because of seasonal usage coming from different groups of users. We also expect these fluctuations to occur as a result of

Table 5.3: Workloads to Examine Effect of Queue Size Thresholding for Request Management

No	Description	Specification
1	Low STF	$STF_{avg}=0.3$ $\sigma_{STF_{avg}}=0.2$
2	High STF	$STF_{avg}=0.6$ $\sigma_{STF_{avg}}=0.2$
3	Rand1 STF	Alternate 25 Days (1) $STF_{avg}=0.2$ $\sigma_{STF_{avg}}=0.1$ (2) $STF_{avg}=0.6$ $\sigma_{STF_{avg}}=0.1$
4	Rand2 STF	Alternate 50 Days (1) $STF_{avg}=0.2$ $\sigma_{STF_{avg}}=0.1$ (2) $STF_{avg}=0.6$ $\sigma_{STF_{avg}}=0.1$

Figure 5.8: The Admission Control Algorithm

```

1  On Event Request  $p$  Arrival
2   $K = GetThreshold()$ 
3  if  $Queue\_size > K$ 
4     $j = Membership(p)$ 
5     $AC_j = \frac{ReqAvgLoad(j, \Delta t)}{ReqAvgSrv(j, \Delta t)}$ 
6    if  $AC_j < \Upsilon_{opt}$ 
7       $Enqueue(p)$ 
8    else
9       $ReplyFail$ 
10   endif
11 else
12    $Enqueue(p)$ 
13 endif

```

varying levels of participation throughout the entire course of the VO's existence. We therefore designed an adaptive mechanism to dynamically compute the optimal threshold according to the properties of submitted requests that is monitored at distinct epoches.

We observed the effects of thresholding based on the system admission failure ratio, for four distinct classes of workloads traces with requests having different statistical properties. They are listed in Table 5.3.

The pseudocode in Algorithm 5.8 gives an overview of the request admission control process. The admission control policy is triggered only when a new request arrives and before it is admitted to the broker's queue. In step(2), the queue size threshold will be

obtained from *GetThreshold()*. If a static scheme is employed, a constant value will be returned. But if an adaptive scheme is used, *GetThreshold()* will communicate with the *RequestPropertyMonitoring* agent and apply a mapping function to compute the actual queue threshold from the statistical parameters obtained. If the current queue size is greater than the threshold, only then admission control will be administered (step(3)). Otherwise, the request will be admitted into the broker's queue (step (12)). In step (4), the request's originating organization (that is, j) will be obtained by *Membership()*. AC_j is the instantaneous admission-contribution ratio of organization j . In step (5), AC_j is calculated by taking ratio of the average admitted requests of each organization using *ReqAvgLoad()* and the contribution by calculating the total number of executed requests by the organization using *ReqAvgSrv()*. Δt represents the time interval where measurements of the submitted and admitted requests are made to compute *ReqAvgLoad()* and *ReqAvgSrv()* prior to the arrival of the new request. At step (6), if the load-contribution ratio AC_j is smaller than the stipulated ratio denoted by Υ_{opt} , then the request is admitted and will be pushed into the broker's queue (step (7)). Otherwise, the request is rejected (step (9)). We calculate Υ_{opt} by taking the ratio between the totalled value of *ReqAvgLoad()* and the totalled value of *ReqAvgSrv()* for all organizations⁶.

The results for the system admission failure ratio with respect to the predefined workloads, are shown in Figure 5.9. The purpose of showing these results is to demonstrate the merits and demerits of both static and adaptive approaches for finding the optimal threshold for the queue size during contention.

For request mix 1, a large proportion of requests are not delayed beyond their execution deadlines with the application queue size thresholding. This results in a significant difference in system admission ratio performance between the schemes with and without thresholding. The improvement is less prominent with request mix 2 because the higher STF that we assigned to each request means that they can better tolerate delays. Also, adaptive thresholding is more successful with request mix 1 than 2. This is because the

⁶The rationale behind doing this will be explained in chapter 6.

Figure 5.9: System Admission Failure Ratio Performance for Different Request Mixes given Different Queue Size Thresholding Methods

nominal value to define the threshold for the static scheme is much closer to the optimal threshold of request mix 1 than 2.

The adaptive scheme is capable of updating the optimal threshold by tracking the statistical properties of the request mix. Request mixes 3 and 4 are used to demonstrate the effectiveness of adaptive management of the optimal threshold when subjected to workload traces with changing statistical properties. It can be seen that if fluctuations occur at a lower frequency as in case for request mix 4, the difference in performance between the static and adaptive scheme is more significant. When the frequency of fluctuations is increased, the adaptive scheme loses its effectiveness.

5.5 Summary

We observed the consequence that not keeping load-contribution ratio of each organization controlled can be detrimental to the system admission ratio – especially if all organizations concurrently compete for shared computing resources during contention.

We also found that participating organizations must keep the aggregate workloads of their users proportional to their resource contributions. This is to ensure that their load-contribution ratios are kept below a predefined threshold, so that increased participation will have little impact on the size of queue storms.

Assigning additional delays to requests is feasible only after overload conditions have been effectively dealt with. This is especially true when the degree of participation and the presence of request arrival correlations is likely to be high. We also showed that controlling the broker's queue size can significantly improve the performance of the system admission ratio during resource overload conditions. With regards to the dynamic nature of the community of users, information on the statistical properties of workloads was periodically captured and used in the admission control policy to improve the system admission ratio.

Taking into account the fluctuating statistical properties of submitted requests to compute an optimal threshold for admission control is critical to the success of the centralized scheme to manage inter-domain contention. This includes using a pre-defined load-contribution ratio as a throttle to prevent overloading due to unusually high number of request arrivals from any particular organization.

Although straightforward in terms of implementation, the experiments demonstrate the practical difficulty of using a centralized management to treat inter-domain contention. The statistical characteristics of the grid workload is likely to change over time. As such, further analysis is required to observe other factors that influence the extent of workload statistical fluctuations so that an adaptive scheme for finding the optimal threshold for admission control can be designed. However, the centralized scheme cannot be used to address user-initiated contention because a pricing framework, which is distributed in nature, has already been designed to cope with the diversity of users' quality-of-service requirements. Therefore, in the next chapter, we integrate our earlier work on the pricing framework into the token-exchange incentive scheme and demonstrate

how it can be used to manage inter-domain contention.

Chapter 6

Employing the Token-exchange Incentive Scheme to Manage Inter-domain Contention

6.1 Introduction

In this chapter, we explore the use of the token-exchange incentive scheme for managing inter-domain contention. As reflected in chapter 3, existing work on the token-exchange incentive scheme takes for granted that the equitable access to shared resources can be achieved. This is because many existing work relies on the token-exchange incentive scheme as a means to reduce free riding amongst participating organizations.

However, because we apply the token-exchange incentive scheme to our conceptual model of the grid resource management framework for admission control, it is necessary to first quantitatively define equitable resource access with respect to our problem domain.

We design a metric, which is termed as fairness to show how the token-exchange incentive scheme can achieve equitable resource access without the need to monitor request workload information at the broker as required for the centralized admission control scheme explored in chapter 5.

The grid is a decentralized system where users independently specify their own quality-

of-service requirements to make use of the shared resources. In this chapter, we present an analysis on how the token-exchange incentive scheme can be employed to achieve fairness. We make several assumptions about the resource management system to demonstrate the conditions necessary for fairness to be attained.

We show that many of the assumptions cannot be enforced in a realistic grid computing environment. This is especially so if the pricing framework designed in chapter 4 has to be incorporated into the token-exchange incentive scheme for admission control. Therefore, the gist of this chapter is to make use of the analysis to give clarity on the key reasons that an optimal value for fairness cannot be attained in practice. This points out that participating organizations must be prepared to tolerate some degree of unfairness when using the shared resources.

This chapter is structured as follows: In section 6.2, we discuss in detail the derivation of the metric for fairness. In section 6.3, we highlight the problem associated with centralized policing of resource usage by relying solely on the resource broker, and then introduce the application of token exchange incentive scheme into the resource management framework to resolve the issues pertaining to the centralized policing strategy. In section 6.3.3, we introduce the properties necessary for the token-exchange incentive scheme to force an optimal value for fairness. In section 6.4, we further examine the additional conditions necessary for the token-exchange incentive scheme when augmented with the pricing framework to achieve fairness. We use this section to clarify that an optimal measure for fairness may not exist and henceforth, the administrative intervention is required to deal with this issue. In section 6.5, we give a summary on various topics addressed in this chapter.

6.2 A Metric for Managing Inter-domain Contention

Our experimental study in chapter 5 has shown that correlations in request arrivals can cause the broker's queue to grow enormously thus resulting in queue storms. The pres-

ence of queue storms can then cause a large proportion of requests to miss their execution deadlines. This condition worsens when organizations do not contribute sufficient computing resources to the community.

We demonstrate from experimental results that, LC , the load-contribution ratio of each organization has a major impact on the size of queue storms. This shows that in order to improve the servicing capacity of the broker, it is necessary to perform admission control according to a stipulated threshold value. However, it is generally difficult to find a good value for the threshold due to the possibility of fluctuating workload statistics.

Furthermore, by employing a centralized admission control scheme, accounting information is needed to reflect each organization's total number of admitted user's requests and the total number of requests serviced over a given epoch in order to calculate their load-contribution ratios. In chapter 5, we devised a policy to keep the load-contribution ratio of each organization below or equal to a prescribed threshold. Instead of pre-determining the threshold and to avoid the centralized admission control, in this chapter we employ the token-exchange incentive scheme to indirectly enforce this policy. Therefore, a metric is required to assess the extent at which the objective of reducing inter-domain contention is met (see section 3.2).

To do so, we first present the basic definitions that are used to construct the metric. Next, we show the significance of using the community's average utility-contribution ratio (as we have done in chapter 5) as the threshold for admission control. We then make use of this information to derive the metric.

6.2.1 Measuring Utility and Resource Contribution

We use $\mu(r_i, s_j)$ to measure the value gained by r_i if it has been assigned s_j . The value returned by $\mu(r_i, s_j)$ depends on the quality-of-service assigned to r_i through s_j . It is measured as follows:

$$\mu(r_i, s_j) = \begin{cases} 0 & \text{if } T(r_i) \neq T(s_j) \\ \vee PE(r_i) \neq PE(s_j) \\ \max\left(0, 1 - \left(\frac{\alpha(s_j)}{\alpha(r_i)}\right)\right) & \text{otherwise} \end{cases} \quad (6.2.1)$$

The above equation returns 0 if the quality-of-service requirements of a request cannot be met. Otherwise, $\mu(r_i, s_j)$ will return a value that is linearly and inversely proportional to the ratio of the assigned slack and the maximum allowable slack of the request. $\mu(r_i, s_j)$ is actually a refined definition of the valuation function of each user, presented in chapter 3. This is because it takes into account the delay assigned to each request by the scheduling mechanism managed by the provider agent. If measurements on the total utility are made at distinct time intervals or epoches, then the total utility gained by organization k at a specific epoch t is given by $\mu_{k,t} = \sum_{\forall r_i \in R_{k,t}} \mu(r_i, s_j)$, where $(r_i, s_j) \in M^{RS}$. $R_{k,t}$ defines all requests submitted during time epoch t . Only requests that are admitted and are assigned to their corresponding service instances are accounted for by this equation.

Also, with different quality-of-service requirements for each service instance, $C(s_j)$ computes the actual measure of resource capacity provided to each service by an organization. It is defined by:

$$C(s_j) = PE(s_j) \times T(s_j) \quad (6.2.2)$$

If $S_{k,t}$ defines the total services provided by organization k to the community during time epoch t , $C_{k,t} = \sum_{\forall s_j \in S_{k,t}} C(s_j)$ measures the resource contribution of organization k during time epoch t .

6.2.2 The Community Utility-contribution Ratio

In this subsection, we explore the rationale for electing the community utility-contribution as a threshold (Υ_{opt}) for the centralized-based scheme for admission control.

To assess the token-exchange scheme's ability to satisfy the conditions for equitable access to shared resources, we establish the conditions for inter-domain contention because it is a direct consequence of free riding behavior by all organizations.

First, $\forall r_i \in R_{k,t}$, we define an expected service instance s_j^* that could be assigned to request r_i , such that it satisfies quality-of-service requirements demanded by r_i .

The conditions for the cause of inter-domain contention are as follows:

- $\forall k, \sum_{\forall r_i \in R_{k,t}} C(s_j^*) \geq C_k \times \Delta T,$
- $\forall k, (C_k - \nu) \times \Delta T < C_{k,t} < C_k \times \Delta T$

The first condition shows that the aggregate resource capacity demanded by all users within an organization at epoch t always exceeds the resource capacity contributed by their organizations. The left hand side summation computes the total resource capacity demanded (ideally) by all requests in organization k . The right hand side computes the actual capacity of computing resources contributed by organization k . It is measured by taking the product of C_k , (i.e., the total number of computing nodes) and ΔT (i.e., the time interval of epoch t). The second condition shows that during contention, the contributed resources of all organizations will be fully utilized. The value of ν is the number of CPUs of the smallest job.

Based on the requirements set out in chapter 3, to achieve equitable resource access among participating organizations, the ratio of the total utility gained by each organization and the organization's resource contribution should be limited by the threshold Υ_{opt} . Recall that this requirement, as shown in eqn 6.2.3, is reflected in the admission control policy for equitable resource access if centralized policing is administered (chapter 5).

$$\frac{\mu_{k,t}}{C_{k,t}} \leq \Upsilon_{opt} \quad (6.2.3)$$

Let $\mu_t^{tot} = \sum_k \mu_{k,t}$ be the total utility attainable by all organizations when inter-domain contention occurs. Given the above conditions for inter-domain contention, the total utility gained by an organization ($\mu_{k,t}$), relative to the total utility of all organizations (μ_t^{tot}) must be equal to the ratio between its resource contribution and the total resource contribution of all organizations ($\frac{C_{k,t}}{C_t^{tot}}$, where $C_t^{tot} = \sum_k C_{k,t}$). Eqn 6.2.4 shows the total utility gained by organization k if equitable access to shared resources is to be enforced during inter-domain contention.

$$\mu_{k,t} \approx \frac{C_{k,t}}{C_t^{tot}} \times \mu_t^{tot} \quad (6.2.4)$$

With respect to eqn 6.2.3 and eqn 6.2.4, an appropriate value for Υ_{opt} should then be $\frac{\mu_t^{tot}}{C_t^{tot}}$, the ratio between the total utility and total contribution of all organizations. Recall that this ratio is the value for Υ_{opt} used in the centralized-based admission control policy in chapter 5.

Figure 6.1 depicts our case for establishing $\frac{\mu_t^{tot}}{C_t^{tot}}$ as the threshold for managing full-load contention. Both $\mu_{k,t}$ and $C_{k,t}$ are represented by a single sector in the pie chart, for a total of 5 organizations. Hence, the inner (shaded) and outer (white) circle represent μ_t^{tot} and C_t^{tot} respectively. The white circle is divided into sectors according to the resource contribution of each organization. This also divides the shaded circle accordingly. The dotted lines are sectors that define the total expected utility of all users within the same organization. It is defined as $\mu_{k,t}^e = \sum_{r_i \in R_{k,t}} \mu(r_i, s_j^*)$. Note that in all cases on the pie chart, $\mu_{k,t}^e$ is greater than $\mu_{k,t}$. It follows that upon admission control, the utility-contribution ratio for each organization ($\frac{\mu_{k,t}}{C_{k,t}}$) must be equal to the community utility-contribution ratio ($\frac{\mu_t^{tot}}{C_t^{tot}}$) if equitable access to shared resources is to be achieved.

6.2.3 Defining Metric for Fairness

Since for all organizations k , $\frac{\mu_{k,t}^e}{C_{k,t}} > \frac{\mu_{k,t}}{C_{k,t}}$, when inter-domain contention occurs, $\frac{\mu_{k,t}}{C_{k,t}}$ has to be equal to $\frac{\mu_t^{tot}}{C_t^{tot}}$ if admission control is performed to achieve fairness. This gives rise

Figure 6.1: An Illustration on Full-load and Partial contention on Shared Resources

to eqn 6.2.5 that is to be minimized. It is defined to take into account all organizations and time epochs. T refers to the total number of epochs and $|U|$ is the total number of organizations. Optimizing fairness requires the value of Φ to be zero.

$$\Phi(k, t) = \begin{cases} \Phi'(k, t), & \text{if } \Phi'(k, t) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\Phi = \sum_{t=0}^T \sum_{k=0}^{|U|-1} \Phi(k, t) \quad (6.2.5)$$

where,

$$\Phi'(k, t) = \frac{\mu_{k,t}}{\mu_t^{tot}} - \frac{C_{k,t}}{C_t^{tot}} \quad (6.2.6)$$

6.3 Employing the Token-exchange Scheme to Achieve Fairness

In this section, we discuss how the token-exchange scheme is used to manage full-load contention. We first reiterate on the weakness of the centralized policy for admission control. Although the centralized scheme is seemingly straightforward, it poses the following difficulties:

1. Arrival patterns of requests and resource availability of different organizations may vary with time. Hence, it is difficult to justify an appropriate time interval between time epoches where $\Upsilon_{opt} = \frac{\mu_t^{tot}}{C_t^{tot}}$ is to be computed.
2. The system has to rely on historical data to compute Υ_{opt} , as we have done in chapter 5. This is because it not possible to measure μ_t^{tot} and C_t^{tot} for the current time epoch t until it ends. While Υ_{opt} may be extrapolated to find the current values for both μ_t^{tot} and C_t^{tot} using statistical prediction strategies, there is generally no optimal solution given the heterogeneous nature of grid workloads [92].
3. Finally, maintaining accounting information to calculate both μ_t^{tot} and C_t^{tot} poses a practical difficulty because the amount of data collected at the broker is dependent on the degree of participation and the time interval of each epoch.

We therefore exploit the token-exchange incentive scheme to perform admission control to achieve fairness without the need to employ Υ_{opt} given the conditions for inter-domain contention. In this way, we can avoid measuring μ_t^{tot} and C_t^{tot} . Using the token-exchange incentive scheme, each organization is required to limit their use of shared resources because a fixed amount of tokens are assigned with respect to their contribution of resources to the community.

6.3.1 Conceptual Resource Management Framework with Token-exchange Incentive Scheme

With reference to the conceptual resource management framework (see section 2.2 and section 2.5), consumer agents at each organization will submit requests for resources to the broker on behalf of their users. Each provider agents will create a candidate service for the request received from the broker. The broker will then match the appropriate candidate service that can best meet the request's quality-of-service requirements and then notify its consumer agent about the assignment.

The following operators define the additional functions that are incorporated into the conceptual resource management framework when the token-exchange incentive scheme is applied.

- $permit(CA_k, r_i, s_j)$ (step 1)¹

This operator represents the first stage of admission control for each request at the consumer agent of organization k (represented by CA_k). The policy $p_i^r \leq B_{k,t}$ is applied to each request so that it will only be submitted to the broker if the request price p_i^r is less or equal to $B_{k,t}$, the total available budget or the total amount of tokens held by organization k at epoch t .

- $request(CA_k, r_i, p_i^r)$ (step 2)

A request operation is submitted to the broker by consumer agent CA_k , together with price p_i^r . This price is the maximum amount an agent is willing to pay in order to be assigned a service instance. p_i^r is therefore the bid in anticipation of an available service instance given its request specification contained in r_i .

- $provide(PA_k, s_j, p_j^s)$ (step 4)

It denotes a single execution of the resource provider agent (PA_k) attempting to share its resources by publishing a service instance s_j to the resource broker. The

¹Denotes operation sequence of the simulator's protocol for admitting request as discussed in chapter 2

resource provider agent has to decide on the price (quantity of tokens) along with a newly generated candidate service instance to be published. s_j is essentially an offer to service r_i made by organization k in exchange for p_j^s tokens.

- $match(r_i, s_j)$ (step 5)

This operator defines the actual transaction to map a request r_i to a candidate service instance s_j by the broker. The two conditions for a suitable match are $p_i^r \leq p_j^s$ and $\mu(r_i, s_j) > 0$. That is, the request price defined by the consumer agent must be greater than or equal to the admission price for the request to be successfully executed. In addition, the quality-of-service assigned must meet the resource specification of r_i .

Based on the above operations, it can be seen that the broker's responsibility is only to assign requests to candidate services. In contrast with centralized policing, it no longer has to perform any admission control operations on incoming requests. This frees the broker of the need to maintain any historical data pertaining to workload information.

6.3.2 An Analysis on the Necessary Conditions to Achieve Fairness

The following analysis shows how a token-exchange incentive scheme can eliminate the need to use Υ_{opt} to achieve fairness. As illustrated in eqn 6.3.1, the introduction of trade essentially chains the process of keeping resources usage bounded by $\frac{\mu_{k,t}}{C_{k,t}}$ through the use of $B_{k,t}^+$ and $B_{k,t}^-$. We first show how an upper bound of $\frac{\mu_{k,t}}{C_{k,t}}$ is autonomously established for each organization k . In other words, agents of each organization cannot unlimitedly increase the utility of their users without making sufficient contribution. We then use these results to show how fairness can be achieved provided that certain properties are enforced.

$$\frac{\mu_{k,t}}{C_{k,t}} \approx \frac{\mu_{k,t}}{B_{k,t}^-} \times \frac{B_{k,t}^+}{C_{k,t}} \quad (6.3.1)$$

$B_{k,t}^-$ denotes the total expenditure at epoch t . The expenditure is a sum of all payments. Given that each service instance s_j has a corresponding admission price p_j^s , the expenditure of organization k at epoch t is $B_{k,t}^- = \sum p_j^s$, given that $(r_i, s_j) \in M^{RS}$ and $r_i \in R_{k,t}$. On the other hand, earnings are represented by $B_{k,t}^+ = \sum p_j^s$, where $s_j \in S_{k,t}$. Requests are submitted continuously for the entire simulation. During contention, we therefore assume that $B_{k,t}^+ \approx B_{k,t}^-$ because an organization needs to use up tokens as soon as they are earned.

Execution of the resource usage policies is decentralized because the agents perform the role of managing their own workloads and resources. $\frac{\mu_{k,t}}{B_{k,t}^-}$ is managed by consumer agents because they autonomously decide their expenditure $B_{k,t}^-$ leading to the aggregate utility $\mu_{k,t}$ gained by users from their executed workloads $R_{k,t}$. Correspondingly, $B_{k,t}^+$ represents the earnings that provider agents make because they initiate the trading of instantiated services based on their contributed resources $C_{k,t}$. The following observations indicate that there are upper bounds for both $\frac{\mu_{k,t}}{B_{k,t}^-}$ and $\frac{B_{k,t}^+}{C_{k,t}}$. So, from eqn 6.3.1, there must be an upper bound for $\frac{\mu_{k,t}}{C_{k,t}}$.

Observation 6.3.1. $\forall k, t$ an upper bound exists for $\frac{\mu_{k,t}}{B_{k,t}^-}$

A consumer agent that wishes to maximize $\frac{\mu_{k,t}}{B_{k,t}^-}$ can increase $\mu_{k,t}$ or/and reduce $B_{k,t}^-$. Intuitively, a consumer cannot increase its utility without incurring any expenditure. The necessary conditions for pricing a service is that $p_j^s > 0$ and $0 < \mu(r_i, s_j) \leq 1$. Let $\delta_{i,j} = \frac{\mu(r_i, s_j)}{p_j^s}$ be the utility-expenditure ratio for an arbitrary request. From the pricing framework in Chapter 4, we know that as demand for a fixed contribution of resources increase then the prices of available services would also increase. The minimum ratio $\delta_{i,j}^{min} \rightarrow 0$ when the price $p_j^s \rightarrow \infty$ with increased competition. On the other extreme, $\delta_{i,j}^{max} = \frac{1}{p_{min}}$, where p_{min} is the smallest possible price for any successful transaction. We expect $\delta_{i,j}$ to be bounded as follows: $\delta_{i,j}^{min} < \delta_{i,j} < \delta_{i,j}^{max}$. If $\delta_{i,j}$ is bounded, the upper bound on $\frac{\mu_{k,t}}{B_{k,t}^-}$ must exist.

Observation 6.3.2. $\forall k, t$, an upper bound exists for $\frac{B_{k,t}^+}{C_{k,t}}$

$S_{k,t}$ is a set of service instances that are created based on the current contribution $C_{k,t}$. Let the price of an average service instance be λ and $n = \frac{C_{k,t}}{|S_{k,t}|} > 1$ be a ratio between contribution and the total number of service instances (i.e., the average number of computing nodes each service instance is assigned). This gives $B_{k,t}^+ \approx \lambda |S_{k,t}|$ and therefore, $\frac{B_{k,t}^+}{C_{k,t}} \approx \frac{\lambda}{n}$. So, n cannot decrease indefinitely due to the minimum resource requirements imposed by consumers. λ cannot be indefinitely increased because the broker is configured to select the lowest priced candidate service that gives the highest payoff² to the request. Essentially, $|S_{k,t}|$ can only be increased or decreased if an organization increases or decreases $C_{k,t}$. Therefore, by this observation $\frac{B_{k,t}^+}{C_{k,t}}$ has an upper bound.

Noting that the token-exchange based resource management framework can keep $\frac{\mu_{k,t}}{C_{k,t}}$ controlled in a decentralized setting, we now illustrate how the utility-contribution ratio can be kept around $\frac{\mu_t^{tot}}{C_t^{tot}}$, in order to guarantee fairness. Given the conditions for inter-domain contention as established earlier in section 6.2.2, then the optimal value is achieved if eqn 6.2.5 is 0.

The following three properties are assumed in order to prove that the system can intrinsically ensure that eqn 6.2.5 is minimized to 0. These properties are conceptual mappings in form of functions joining the aggregate utility gain by users and the resource contribution of each organization. The mappings are conceived from the flow diagram in Figure 6.2. It shows that transitional connection between the utility gained by users from the resources contributed by their respective organizations. Both consumer and provider agents are in the center of the decision process to trade resources on behalf of their users.

Property 6.3.1. Linearity in Utility on Expenditure

Utility on expenditure is a function $f_k : \mathbb{R}^+ \rightarrow [0, 1]$ for each consumer agent. Formally,

²The payoff to a request will be defined in section 6.4.1

Figure 6.2: An Illustration Of the flow of aggregate utility gained by participating organizations based on the Token-exchange scheme

the function returns the aggregate utility $\mu_{k,t}$ gained by users for the total expenditure of tokens ($B_{k,t}^-$), in order to successfully execute its own requests $R_{k,t}$. Since f_k is linear, $\forall k, \frac{\mu_{k,t}}{B_{k,t}^-} = \varepsilon$ and if $B_{k,t}^- = 0 \Rightarrow \mu_{k,t} = 0$.

Property 6.3.2. Linearity in Expenditure Propensity on Earnings

We define this function as $g_k : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ for each consumer agent. The function defines the amount of budget that an organization is willing to spend ($B_{k,t}^-$) given the amount it is currently earning ($B_{k,t}^+$). Again, $\forall k, \frac{B_{k,t}^-}{B_{k,t}^+} = \varsigma$ and if $B_{k,t}^- = 0 \Rightarrow B_{k,t}^+ = 0$.

Property 6.3.3. Linearity in Earnings on Contribution.

The earnings on contribution is a function $h_k : \mathbb{N} \rightarrow \mathbb{R}^+$ for each provider agent. It returns the earnings ($B_{k,t}^+$) yielded from the resource contribution ($C_{k,t}$). The linearity property implies that the system is able to award earnings linearly proportional to the

measure of contribution of shared resources. , such that, $\forall k, \frac{B_{k,t}^+}{C_{k,t}} = \gamma$ and $C_{k,t} = 0 \Rightarrow B_{k,t}^+ = 0$.

Theorem 6.3.1. *If properties 6.3.1, 6.3.2 and 6.3.3 are true then $\Phi = 0$*

Proof: We show the proof by deriving a composition of f_k , g_k and h_k to establish a relationship between utility gained and contribution. With that, we show that the composition is also linear. Consider the composition $\rho_k = f_k \circ g_k \circ h_k$ such that $\rho_k : \mathbb{N} \rightarrow [0, 1]$. We use this composition to show that for all organizations, the utility-contribution ratio is constant and equal to the total utility and total contribution ratio. $\frac{\mu_{k,t}}{C_{k,t}} = \frac{\rho_k(C_{k,t})}{C_{k,t}} = \frac{\varepsilon\gamma C_{k,t}}{C_{k,t}} = \varepsilon\gamma$. Similarly, $\frac{\mu_t^{tot}}{C_t^{tot}} = \frac{\sum_k \mu_{k,t}}{\sum_k C_{k,t}} = \frac{\sum_k \varepsilon\gamma C_{k,t}}{\sum_k C_{k,t}} = \varepsilon\gamma$. Hence, we show that $\forall k, \frac{\mu_{k,t}}{C_{k,t}} = \frac{\mu_t^{tot}}{C_t^{tot}}$. This implies that $\frac{\mu_{k,t}}{C_{k,t}} - \frac{\mu_t^{tot}}{C_t^{tot}} = 0$, and therefore, $\Phi = 0$.

6.3.3 Discussion

The purpose of the theoretical analysis on application of the token-exchange scheme for resource management is to observe the necessary conditions and mechanisms that are required to guarantee fairness. To conclude this section, we first give a general overview on the design characteristics to be embodied by the resource management framework.

- **Agent Homogeneity**

The first prerequisite to achieve fairness is that both consumer and producer agents must be homogeneous. Properties 6.3.1 and 6.3.2 shows that all consumer agents must have the same expectation of utility on their expenditure with respect to their current earnings as reflected by functions f_k and g_k . Similarly, property 6.3.3 shows the necessity for all provider agents to price services using the same function h_k with respect to demand conditions and the quantity of their resource contribution. The linearity of f_k and h_k are influenced by the pricing framework that is used by both the consumer and provider agents to use and provide the shared resources

respectively. The linearity of g_k , on the other hand, is determined by $permit()$. The need for agent homogeneity therefore points out the fact that the same pricing framework must be enforced at all organizations. In the next section, we examine the pricing framework's influence on the linearity of f_k and h_k when it is integrated into the token-exchange incentive scheme.

- **Load-based Pricing of Shared Resources**

The linearity in earnings on contribution (Property 6.3.3) means that an organization will always attain its earnings proportional to its contributions. This property can be realized by having all provider agents to implement a load-based pricing framework (e.g., the pricing scheme in chapter 4) to compute the admission price for requests. The broker must first employ a policy to select candidate services in such a way that it balances the utilization of resources among the participating organizations. As described in chapter 4, the level of utilization or load ratio is proportional the admission price for a newly generated service instance. Furthermore, the pricing framework takes into account the opportunity cost for the discrepancy of quality-of-service demanded by all requests. This means that, if the load ratio at all organizations are balanced, then a larger contribution of resources also imply that the combined capacity of resource used by the scheduled service instances will also be larger. Consequently, the earnings of each organization will also proportional to their resource contributions. We justify this postulation by means of a proof in the next section.

6.4 Managing Inter-domain Contention using the Pricing Framework

Augmenting the token-exchange incentive scheme with the pricing framework in chapter 4 imposes some degree of restriction to the design of the admission control framework. These restrictions may interfere with the system's ability to attain an optimal measure for

fairness even with the earlier conditions defined for the cause of inter-domain contention. Hence, the motivation of this section is to investigate in detail, by means of a mathematical analysis, to demonstrate the pricing framework's influence on the token-exchange incentive schemes ability to achieve fairness.

Policing against unfair usage of shared resources by means of the token-exchange incentive scheme is achieved by linking the sum of earnings ($B_{k,t}^+$) from services to the expenditure ($B_{k,t}^-$) so that consumer agents can leverage on the organization's earnings to generate their bids for requests to gain access to shared resources. Hence, in the first part, we examine how the provider agents, together with the broker, are organized to achieve property 6.3.3, the linearity in earnings on contribution. We then demonstrate the difficulty of attaining both property 6.3.1 (utility on expenditure) and 6.3.2 (expenditure propensity on earnings) as a consequence of applying the pricing framework. This does not mean that the pricing framework will defeat the functional purpose of the token-exchange incentive scheme. But, we make use of the analysis to show that an ideal value for fairness ($\Phi = 0$) is non-existent. This points out the fact that resource administrators from different organizations have to agree on the degree of unfairness that they are willing to tolerate when a realistic system is deployed.

6.4.1 Achieving Linearity of Earnings on Contribution using the Pricing Framework

We assume that all agents are homogenous; meaning that, the resource scheduling and allocation modules, the pricing decisions are the same at all participating organizations. When a request arrives at the broker, it will be propagated directly to the provider agents.

In this sub-section, we give a generalized mathematical analysis to show that the admission control mechanism enables organizations to earn tokens proportionally to their active contribution of shared resources. First, the broker's matchmaker (denoted by the *match()* operation) is designed to select among the candidate services returned by the

provider agents, in a manner, so that the load-ratio of all organizations will be balanced. Second, the pricing framework is employed by each provider agent to ensure that the earnings is proportional to the resource contribution made by its organization. Doing so enables the system to achieve linearity in earnings on contribution.

For a given request r_i , let $S^r = \{s_0, s_1, \dots, s_j, \dots, s_{|U|-1}\}$ be a set of candidates serviced, where s_j is returned by the provider agent of organization j . The size of S^r is equal to $|U|$, the total number of participating organizations. For any organization n , if s_j is able to meet the quality-of-service requirements of r_i , then $\mu(r_i, s_n) > 0$, $s_j \neq \text{Null}$ and $p_j^s > 0$. p_j^s is the admission price of the service instance. The corresponding admission price of s_n Otherwise, $\mu(r_i, s_j) = 0$ and $s_j = \text{Null}$. The broker's matchmaking policy $match()$, will choose the candidate service that gives the highest payoff to the request, on top of the requirements set out in section 6.3.1. It is measured by taking the difference between the utility ($\mu(r_i, s_j)$) gained by the request and the corresponding normalized admission price ($\hat{p}_j^s$). The normalization is performed on the admission price of all candidates services to so that they range between 0 and 1. The matchmaking policy to select the candidate service that maximizes the payoff to r_i as follows:

1. Select $s_k \in S^r$ and add it to S^p , if $\forall s_n \in S^r, \mu(r_i, s_k) - \hat{p}_k^s \geq \mu(r_i, s_n) - \hat{p}_n^s$.
2. If the size of S^p is greater than one, then select s_k if $\forall s_n \in S^p, k < n$.

where,

$$\hat{p}_k^s = \frac{p_k^s}{\sum_{j=0}^{|U|-1} p_j^s}$$

In (1), the policy chooses a set of candidate services that return the highest payoff. In (2), the policy selects the candidate service with the lowest index.

Lemma 6.4.1. *The matchmaking scheme always chooses the candidate service with the lowest price*

Proof:

- Case 1 (Without time delay) :

If $\forall m, n$, where m and n are indices to a set of candidate service in S^r , created by organization m and n respectively. For a given request r_i , $\mu(r_i, s_m) = \mu(r_i, s_n)$ is always true. This is a trivial case where no time delay is introduced to requests that have been successfully admitted. Therefore, $\forall s_j \in S$, $\mu(r_i, s_j)$ always returns 1 if s_j can meet the quality-of-service requirements of the request. The remaining variable in consideration is the admission price of each candidate service, whereby, the lowest one will be selected if the matchmaking policy is enforced.

- Case 2 (With time delay) :

For the matchmaker to select the lowest price candidate service, then the condition $\mu(r_i, s_n) \geq \mu(r_i, s_m) \Leftrightarrow \hat{p}_m^s \leq \hat{p}_n^s$, must also be true. With reference to chapter 4, in eqn 6.2.1, the slack ratio (delay) assigned to each service is proportional to the instantaneous load-ratio. This implies that, $\alpha(s_m) \leq \alpha(s_n) \Leftrightarrow \frac{\Delta C_m}{C_m} \leq \frac{\Delta C_n}{C_n}$. Based on eqn 6.2.1, we can deduce that $\mu(r_i, s_n) \geq \mu(r_i, s_m) \Leftrightarrow \alpha(s_m) \leq \alpha(s_n)$. Therefore, $\mu(r_i, s_n) \geq \mu(r_i, s_m) \Leftrightarrow \frac{\Delta C_m}{C_m} \leq \frac{\Delta C_n}{C_n}$. The pricing framework in eqn 4.4.4, implies that, for a given request, $\frac{\Delta C_m}{C_m} \leq \frac{\Delta C_n}{C_n} \Leftrightarrow \hat{p}_m^s \leq \hat{p}_n^s$ because the admission price of a service is proportional to the load-ratio. Hence, we have $\mu(r_i, s_n) \geq \mu(r_i, s_m) \Leftrightarrow \hat{p}_m^s \leq \hat{p}_n^s$. ■

Lemma 6.4.2. *The system ensures that if m and n are indices of any two different organizations, there must exist a maximum bound of $|\frac{\Delta C_m}{C_m} - \frac{\Delta C_n}{C_n}|$.*

Proof: Let s_m be a chosen candidate service from organization m . Given lemma 6.4.1, s_m has the admission lowest price. This means that, $\forall n$ where $n \neq m$, $p_n^s \geq p_m^s \Rightarrow \frac{\Delta C_n}{C_n} \geq \frac{\Delta C_m}{C_m}$. Upon servicing s_m , the load-ratio of organization m increases to $\frac{\Delta C_m + PE(s_m)}{C_m}$. If $\forall n$, $\frac{\Delta C_m + PE(s_m)}{C_m} < \frac{\Delta C_n}{C_n}$, subsequent requests will still be serviced by m . However if $\exists n$, $\frac{\Delta C_n}{C_n} < \frac{\Delta C_m + PE(s_m)}{C_m}$, then m will not be selected when the next request is serviced.

Hence, this implies that the load-ratio for all organizations will be balanced, leading with a maximum bound of $|\frac{\Delta C_m}{C_m} - \frac{\Delta C_n}{C_n}|$. Furthermore, this bound will approach zero as more requests are serviced. For a given organization k , the load ΔC_k , will increase as more requests are admitted. Hence, the change in ΔC_k will reduce as more services are created. As such, $\forall n$ and m , the maximum bound of $|\frac{\Delta C_m}{C_m} - \frac{\Delta C_n}{C_n}|$ will reduce to approach 0. ■

Theorem 6.4.1. *The system achieves linearity in earnings on resource contribution.*

Proof: From the results of the above lemma, the load-ratio for all organizations will be balanced as contention increases. This means that between any organization m and n where $m \neq n$, $C_m \geq C_n \Leftrightarrow \Delta C_m \geq \Delta C_n$. To generalize the analysis, we use the same quality-of-service specifications for all services. Hence, if all services are represented by s_j , let $T_j = T(s_j)$ and $PE_j = PE(s_j)$. We establish the relationship $\Delta C_m = \zeta_m PE(s_j)$ and $\Delta C_n = \zeta_n PE(s_j)$ where both ζ_m and ζ_n are multiples of $PE(s_j)$ to the load of organization m and n respectively. This also implies that $C_m \leq C_n \Leftrightarrow \zeta_m \leq \zeta_n$.

From pricing framework, the price for the first and subsequent services for any organization k are as given:

$$p_0^s = \frac{PE_j T_j (\omega + PE_j)}{2\omega\tau C_k}$$

$$p_n^s = \frac{n (PE_j)^2 T_j}{2\omega\tau C_k} + p_0^s$$

The total earnings gained by organization k with respect to its load-ratio or utilization is $B_k^+ = \sum_{i=0}^{\zeta_k-1} p_i^s$. This gives eqn 6.4.1³.

$$B_{k,t}^+ = \zeta_k \left\{ \frac{(PE_j)^2 T_j}{2\omega\tau C_k} \left[\frac{\zeta_k - 1}{2} \right] + p_0^s \right\} \quad (6.4.1)$$

Since $\forall k$, $\frac{\zeta_k}{C_k}$ is a constant, we let $\frac{\zeta_k}{C_k} = \sigma$. Also, because ζ_k is an estimation of the total number of instantiated services, we let $\zeta_k \gg 1$. Hence an approximation to the earnings that organization k gets is,

$$B_{k,t}^+ \approx \zeta_k \left\{ \frac{\sigma (PE_j)^2 T_j}{4\omega\tau} + p_0^s \right\}$$

³A full derivation of this equation is given in Appendix C

We have earlier established that for any distinct organizations m and n , $C_m \leq C_n \Leftrightarrow \zeta_m \leq \zeta_n$. Hence, the above equation shows that the earnings that each organization receives is proportional to its resource contributions.

The full-load earnings is the total earnings that an organization would get when all its resources are fully used. We compute an organization's full-load earnings by substituting $\frac{C_k}{PE_j}$ into ζ_k . Doing this, we get eqn 6.4.2.

$$B_{k,t}^+ \approx \frac{T_j}{4\omega\tau} C_k + \frac{T_j(\omega + PE_j)}{4\omega\tau} \quad (6.4.2)$$

■

By observing eqn 6.4.2 and taking into account that C_k much greater than both $\frac{T_j}{\tau}$ and PE_j , an organization's full-load earnings is linearly proportional to its contribution of physical resources.

6.4.2 Achieving Linearity of Utility on Earnings based on the Pricing Framework

Linearity of utility on earnings (i.e., $\frac{\mu_{k,t}}{B_{k,t}^+}$) means that the ratio between the aggregate utility gained by users and the total earnings at any time epoch, is the same for all organizations. For this property to be achieved, both property 6.3.1 and 6.3.2 must be satisfied.

To guarantee property 6.3.1, each admitted request should have the same utility-price ratio. As illustrated in eqn 6.4.3, the utility over expenditure ratio at epoch t can be approximated by the utility gained by an arbitrary request r executed by s with admission price p . This means that the utility gained from a successfully executed request divided by its admission price must be a constant.

$$\frac{\mu_{k,t}}{B_{k,t}^-} \approx \frac{\nu\mu(r, s)}{\nu p} \approx \frac{1}{p} \quad (6.4.3)$$

Because slack is not assigned to a large number of service instances (less than 50hrs of execution time), the utility of their corresponding requests is 1.0. $\mu(r, s)$ can therefore be approximated to 1.0 because the majority of services are not assigned delays. The remaining variable p is the admission price that is to be paid in order for r to be executed.

Since the price of services and users' resource requirements are likely to vary considerably, the consumer agents must be designed to adaptively generate bids for requests while keeping their expenditure of tokens controlled. To do so, each consumer must be equipped with some mechanism to estimate the admission price of the request before it is submitted to the broker.

Therefore, p , which is computed using the pricing framework, is also influenced by the number of demanded computing nodes, $PE(s)$, execution time $T(s)$, and $\frac{\Delta C_n}{C_n}$, the load-ratio of the organization (n) that executes r . This poses a difficulty for each consumer agent to obtain a near constant utility-price ratio for all requests. This is because the admission price of a request is influenced by the quality-of-service requirements and the degree of resource contention on the organization's resources, whose candidate service is selected by the broker. Hence p , cannot be the same all the time.

As for property 6.3.2 (linearity in expenditure propensity on earnings), the ratio between expenditure and earnings for given epoch is dependent on the internal policies employed by each agent to decide on the amount of tokens to use ($B_{k,t}^-$) with respect to its earnings ($B_{k,t}^+$). Since $B_{k,t}^-$ is also influenced by other factors (e.g., the total number of requests submitted at epoch t), guaranteeing property 6.3.2 is beyond the control of the consumer agent.

6.4.3 Discussion

The analysis in this section shows that an optimal measure for fairness is non-existent if the token exchange incentive is augmented with the pricing framework in chapter 4. This is because the resource requirements of user's requests subjects the system to situations

at which the properties 6.3.3, 6.3.1 and 6.3.2 to deviate from perfect linearity.

A practical difficulty in achieving linearity in utility on earnings (property 6.3.3) occurs when one or more organization's resources do not meet the resources requirements (e.g., CPU speed, available memory and storage) of a majority of users. This is a potential problem because users have the autonomy to specify resource requirements on their own. If they have preferences for computing resources belonging to any specific organization, the broker will not be able to balance the load on shared resources to achieve linearity of earnings on contributions.

As discussed in the previous subsection, deviations from linearity for both property 6.3.1 (utility on expenditure) and 6.3.2 (expenditure propensity on earnings) is highly possible due to the discrepancy in admission price for requests with different quality-of-service requirements.

6.5 Summary and Conclusions

The work in this chapter establishes the foundational justification for employing the token-exchange scheme to unify the process of managing both user and inter-domain contention. The major benefit of employing the token-exchange scheme to manage inter-domain contention is that the need to employ a stipulated utility-contribution ratio to perform admissions control on requests submitted by users can be avoided. Due to the dynamic nature of workloads, it is difficult to estimate the community utility-contribution ratio. By employing the token-exchange incentive scheme, we can avoid the need to compute this value. The secondary advantage is that, this reduces the amount of accounting information that is needed to be maintained at the broker for admission control.

We first establish fairness; a metric required to quantitatively assess the effectiveness of the token-exchange incentive scheme to manage inter-domain contention. We then establish the system properties that are required to obtain an optimal value for fairness.

An in depth analysis shows how the pricing framework, together with the matchmaking policy of the broker is applied to obtain a partial solution for fairness (linearity of earnings on contribution). However, this is provided that all organizations are able to meet the qualitative requirements of organizations. Furthermore, the linearity of utility on earnings requires that the admission price of all admitted requests are to be the same. However, the admission price of requests are likely to vary because they are strongly influenced by the quality-of-service requirements set by their users and the instantaneous level of contention.

In this chapter, we therefore, show that the token-exchange incentive scheme when augmented with the pricing framework does not allow for the ideal value of fairness metric to be attained. As mentioned, this does not mean that the token-exchange incentive scheme is no longer feasible but it is necessary for administrators at each organization to tolerate the prevailing degree of unfairness unless some form of administrative interventions are introduced. An example is for the various administrative domains to increase their contribution of resources. In chapter 7, we observe that the the load-contribution ratio strongly influences the degree of fairness even with the use of token-exchange incentive scheme.

Chapter 7

Adaptive Policies for Managing Inter-domain Contention

7.1 Introduction

The primary aim of admission control in the context of managing contention is to jointly achieve the best admission ratio while ensuring that resources are shared fairly between organizations with respect to their contributions. However, a potential obstacle to successful application of the token-exchange scheme for admission control is with the difficulty of finding an appropriate quantity of tokens to be initially assigned to each organization with respect to their contributions. This is because the admission ratio and fairness are performance trade-off for different quantity of tokens assigned. A straightforward reasoning is that, if the initial budget assigned is too high, the total utility gained by all users in an organization may not be proportional to the organization's resource contribution, since the amount of assigned tokens to an organization will have very little influence on the limit that users can gain access to shared resources. This would therefore cause the token-exchange incentive scheme to lose its effectiveness to achieve fairness. On the other hand, if the initial assignment is too low, agents cannot utilize shared resources sufficiently due to a lack of available tokens in circulation, thus resulting in poor system admission ratio. In this chapter, we therefore design policies to be incorporated into the

pricing framework (introduced in chapter 4) employed by both consumer and provider agents. The policies are built to reduce the sensitivity of the initial token assignment on the performance trade-off between fairness and the system admission ratio.

The chapter is structured as follows – In section 7.2, we survey pricing algorithms under various settings that resemble our approach for adapting the pricing framework to cope with the problem addressed in this chapter. In section 7.3, an overview of the pricing mechanism that is employed to cope with the above problem is discussed. In section 7.4, we describe the experimental procedures. In section 7.5, we present the experimental results and discussion on various policing schemes. This is then followed by summary and conclusions in the last section.

7.2 Related Work

Extension of the pricing framework requires a strategy for assimilating information from diverse sources into a rule-base that is to be used dynamically to modify the demand and supply policies for trading resources. Hence, in this section, we give an overview of the algorithms that are capable of factoring multiple parameters into a decision process where dynamic schemes are subsequently employed to relate perceived data from these parameters to pricing strategies in order to achieve the decision goal.

In the area of computational finance, automated trading algorithms generally employ some degree of intelligence by introducing heuristics into the decision mechanism of agents to generate trades. Chan et al. demonstrated the use of bayesian trading agents that make use of current and historical market information to update their beliefs about the state of the economy to make predictions on future prices of commodities [20]. The pricing mechanisms take into account changes in the market conditions on the premise that recorded events on market behavior have an impact on the future strategies undertaken by agents when competing for resources.

The advantage of relying on past information to estimate future prices is also em-

ployed in the design of software agents in online auctions [47, 91]. One example is the Trading Agent Competition (TAC) [87], where agents are built to compete in a variety of real-time simulated online auctions. The agents make human-level trading decisions on transactions similar to those of electronic auctions on the Internet (e.g., e-Bay). A prominent example is the work of He et al., where they employed a fuzzy inference engine that draws information from a wide range of sources to formulate a high-level pricing strategy to guide lower level algorithms to generate the actual bids for resources [44]. Wurman et al. used a decision representation scheme to capture information regarding the strategies taken by other agents and the market. The combined information, which takes the form of a vector is mapped the combined information into a set of distinct states [91]. Between the states are transitions that are linked to a set of actions, to define the bidding strategies of agents to compete for resources. The decision representation scheme is constructed as a stochastic state machine called a Markov Decision Process (MDP). Feasible actions are chosen to maximize a predefined objective function. In situations where the state transitions are not defined, reinforcement learning is employed to find the transitions in the form of policies to construct the decision mechanism for the agent's pricing strategy [9, 18].

In our work, we do not employ techniques that attempt to dynamically find a set of actions to optimize a long-range objective function at each organization. This is because the agents in our case, requires some degree of homogeneity in order to satisfy the conditions for the system performance metrics. For this reason, we apply the same trading policy that maps the observed data to a set of actions for all agents.

7.3 Design of Trading Policies

Figure 7.1 shows a schematic of each organization's decision mechanism for computing bids (demand) and admission prices (supply). The pricing decision mechanism is generally divided into three parts – the accounting ratios, a decision block for each agent, and the

Figure 7.1: Schematic of Each Organization's Decision Mechanism.

pricing framework. The accounting ratios are a set of input parameters to the decision blocks. Each decision block generates actions according to a set of policies to manipulate the price index for each agent. The pricing framework is reused to compute the actual demand¹ or supply² price by both consumer and provider agent by taking into account the price index.

7.3.1 Accounting Ratios

The accounting ratios are information sources that are factored into each agent's decision making process. The policies used in each decision block map the values of the accounting ratios to a set of actions. The actions manipulate the upward or downward movement of the price index (σ_c and σ_p for consumer and provider agents respectively) which is applied to the pricing framework to compute the actual number of tokens to be used or earned

¹The demand price refers to the request price or bid issued by consumer agents.

²The supply price refers to the admission price of a candidate service issued by provider agents.

by a request or service respectively. The ratios are updated at regular time intervals. Following an update epoch, the decision block of each agent responds to changes in the accounting ratios by switching between the predefined actions to update their respective price indices. In our experiments, the time interval between two consecutive updates of the accounting ratios is 1 hour for both consumer and provider agents.

The accounting ratios are used to assess the degree of contention and economic status of each organization. The three basic parameters that are used to compose the ratios are average demand price (D_t^{avg}), average local supply price and average current budget (B_t^{avg}) (or the average total number of tokens held by an organization). The average demand price at each organization estimates the aggressiveness of each consumer agent's bidding strategy for shared resources. It is calculated by taking the average value of all submitted bids by each consumer agent. The average local supply price estimates the degree of contention on computing resources at each organization because it is proportional to the instantaneous utilization due to the pricing framework. It is calculated by taking the average service price of all admitted requests. The accounting ratios are composed from these parameters as given in the definitions below.

- Demand-budget (DB) Ratio

It measures the ratio between the average demand price and the available budget ($B_{k,t}$) owned by an organization. This ratio makes a comparison between the intensity of competition by consumer agents relative to the economic status of their respective organizations.

- Supply-budget (SB) Ratio

It measures the ratio between the average local supply price and the available budget ($B_{k,t}$). This ratio gives an indication of the organization's gauge of its average anticipated earnings relative to its current economic status.

- Supply-demand (SD) Ratio

It is computed by taking the ratio between the local average supply price and

average demand price. It is used to compare the differential in demand and supply pricing conditions.

The above ratios are normalized to the range between 0 and 1 by applying the following standard functions. For DB and SB, we use eqn 7.3.1. B_t^{avg} is the average budget between any two consecutive epoches. Λ is a variable that is substituted with the actual ratio of either DB or SB. β defines the rate at which the two ratios vary between 0 and 1 with respect to their actual range.

$$I_{\Lambda} = \begin{cases} 0, & \text{if } B_t^{avg} = 0 \\ 1 - e^{-\beta\Lambda}, & \text{otherwise} \end{cases} \quad (7.3.1)$$

For SD, eqn 7.3.2 is used. D_t^{avg} represents the average demand price at epoch t . When the average supply and demand is balanced, that is, when $SD = 1.0$, the value is 0.5. When $SD > 1.0$, the value converges to 1.0 and when $SD < 1.0$, it converges to 0.0.

$$I_{SD} = \begin{cases} 1.0 - 0.5e^{-\beta(SD-1.0)}, & \text{if } D_t^{avg} \neq 0, SD \geq 1.0 \\ 0.5e^{\beta(SD-1.0)}, & \text{if } D_t^{avg} \neq 0, SD < 1.0 \\ 1.0, & \text{otherwise (i.e., } D_t^{avg} = 0) \end{cases} \quad (7.3.2)$$

The actual values of β for SB, DB and SD are chosen (see Table 7.3.1) so that their normalized values are mapped to three distinct states; namely, LOW, MED and HIGH as shown in Table 7.2. The range of values are assigned as follows: We choose $DB = 0.25$ to distinguish between the MED and HIGH state. On the average, at each organization, 3 to 4 requests are submitted at each time epoch. If the amount of tokens (budget) is to be proportionally divided amongst the requests, then the amount of tokens assigned to each request is approximately 25% of the available budget³. We arbitrarily use 5% to distinguish the interval between LOW and MED. We assign the same range of values to SB based on a same concept. As for SD, MED is assigned if the average supply price ranges between 75 to 125% of the average demand price.

³Again, we use the average submission rate of all workloads to find the value that distinguishes the discrete states for DB in a similar way of defining competitive consumers in chapter 4.

Table 7.1: Values of β for Accounting Ratios

Ratio	$\beta =$
SB	5.991
DB	5.991
SD	2.878

Table 7.2: Mapping of Accounting Ratios to Distinct States

Ratio	Range of Values	State	Remarks
I_{SB}	$0 \leq I_{SB} \leq 0.25$	LOW	The average supply price is 0 to less than 5% of average budget
	$0.26 \leq I_{SB} \leq 0.75$	MED	The average supply price is between 5 to 25% of average budget
	$I_{SB} \geq 0.75$	HIGH	The average supply price is greater than 25% of average budget
I_{DB}	$0 \leq I_{DB} \leq 0.25$	LOW	The average demand price is 0 to less than 5% of average budget
	$0.26 \leq I_{DB} \leq 0.75$	MED	The average demand price is between 5 to 25% of average budget
	$I_{DB} \geq 0.75$	HIGH	The average demand price is greater than 25% of average budget
I_{SD}	$0 \leq I_{SD} \leq 0.25$	LOW	The average supply price is less than 75% of average demand price
	$0.26 \leq I_{SD} \leq 0.75$	MED	The average supply price is between 75 to 125% of average demand price
	$I_{SD} \geq 0.75$	HIGH	The average supply price is greater than 125% of average demand price

7.3.2 Policy Design

As mentioned, the weakness of the token-exchange scheme is that the performance trade-off between fairness and the system admission ratio is highly sensitive to the initial budget assignment. To address this problem, we do not try to experimentally find an optimal initial token assignment that gives a good joint performance for both fairness and the system admission ratio. This is because the optimal budget is influenced not only by the pricing framework employed by both consumer and provider agents, but also by other environmental factors. Alternatively, we try to lessen the sensitivity by incorporating additional policies to the pricing framework to cope with different degrees of contention via the pricing indices (σ_c and σ_p). Each consumer and provider agent has a decision block that implements the policies to manipulate their respective price indices on the following set of actions.

- PLUS/MINUS

This action creates a step increase/decrease in the price index. Both σ_c and σ_p are essentially factors applied to the demand and supply price based on the actual

token-exchange scheme.

- ZERO

This is the default action where the agent keeps the price index unchanged.

The basic principle used to cope with the performance trade-off between fairness and admission ratio is as follows:

- If contention is relatively low, bring the supply prices as low as possible so that consumer agents are able to increase their chance of gaining admission to shared resources with less expenditure so as to improve the admission ratio.
- When contention starts to build up, increase the admission price progressively. Doing this will increase the effect of the token-exchange incentive scheme on each agent's trading behavior to ensure that fairness can be administered.

We rely on the provider agent to achieve the above by mapping SB to a set of actions on its supply price index, σ_p . We use SB because it combines the measure of competition for shared resources and the organization's financial position (size of current budget) to use shared resources in the current and next time epoch.

The COOPERATIVE scheme, as shown in Figure 7.2 (top), defines the policies implemented for each provider agent. The values obtained from each accounting ratio are classified into three discrete values – *LOW*, *MED* and *HIGH*, as given Table 7.2. η , a constant value fixed at 0.01, is used to change the values for both σ_c and σ_p .

When SB is *LOW*, the average supply price may be low, indicating that there are not many current services in execution. It may also imply that there are sufficient tokens for future expenditure. The COOPERATIVE scheme attempts to increase resource utilization by discounting the admission (supply) price of a service so as to increase the provider agent's probability of admitting more requests.

```

if  $SB = LOW$  then
   $\sigma_p = \sigma_p - \eta$ 
else if  $SB = HIGH$  then
   $\sigma_p = \sigma_p + \eta$ 
else if  $SB = MED$  then
  if  $SD = LOW$  then
     $\sigma_p = \sigma_p + \eta$ 
  else if  $SD = HIGH$  then
     $\sigma_p = \sigma_p - \eta$ 
  end if
end if
if  $\sigma_p < 0.01$  then
   $\sigma_p = 0.01$ 
end if

```

```

if  $SB = LOW$  then
  if  $Avg\_Supply = 0.0$  then
     $\sigma_p = \sigma_p - \eta$ 
  else
     $\sigma_p = \sigma_p + \eta$ 
  end if
else if  $SB = HIGH$  then
   $\sigma_p = \sigma_p - \eta$ 
else if  $SB = MED$  then
  if  $SD = LOW$  then
     $\sigma_p = \sigma_p + \eta$ 
  else if  $SD = HIGH$  then
     $\sigma_p = \sigma_p - \eta$ 
  end if
end if
if  $\sigma_p < 0.01$  then
   $\sigma_p = 0.01$ 
end if

```

Figure 7.2: Policy Schemes, COOPERATIVE (top) and COMPETITIVE (bottom) for Provider Agents

When SB is *HIGH*, the average supply price has grown larger because either utilization of resources has increased or the available tokens for future use are running out. So, the provider agent attempts to increase its opportunity to gain more tokens on its remaining resources by increasing σ_p . However, since the matchmaker tries to choose the candidate service that gives the highest payoff to the request (as already explained in chapter 6), the provider agent must reduce its price index when SB drops.

SD is used to determine the movement of the supply price index when SB is *MED*. SD quantifies the ratio between the average demand and supply price. Hence, it judges the difference in the degree of contention between consumer and provider agents. The objective is to reduce the differential, so that consumer and provider agents can adaptively reach an equilibrium needed to estimate the global state of contention for resources. By employing SD , provider agents try to match their price index as close as possible to the prices indices of consumer agents. When SD is *HIGH*, it indicates that the provider has overestimated the level of contention. In such case, the supply price index must be reduced to improve the chance that consumer agents can bid above the supply price. When SD is *LOW*, it suggests that the average demand price or bid is higher than the average supply price. This is an indication to the provider agents so that they can further increase their earnings by raising their supply price index. When SD is *MED*, σ_p remains unchanged.

The COMPETITIVE scheme, as shown in Figure 7.2 (bottom), is a reverse strategy of the COOPERATIVE scheme. It is introduced to give additional emphasis on the need to apply the policing rules based on COOPERATIVE in order to achieve the desired performance results. Provider agents adopting the COMPETITIVE scheme are primarily aimed at maximizing their earnings.

When SB is *LOW*, each provider agent greedily raises σ_p with the aim of increasing its earnings. A potential problem arises when the average supply price is zero because it has not serviced any requests in the previous time epoch. Under this situation, if

a provider agent continues to increase the price index, it will further lower its chance to service requests. We prevent this from occurring by decreasing σ_p when the average supply price is 0.0. When SB is *HIGH*, it reduces σ_p to improve the provider agent's chance to service submitted requests.

Similar to the COOPERATIVE scheme, when SB is *MED*, the price index will be increased when SD is *LOW* – indicating that the average supply price index is lower than the average demand price index and therefore the provider agent should try to match the demand conditions by increasing σ_p . When SD is *HIGH*, the provider agents does the opposite. And when SD is *MED*, σ_p remains unchanged.

For both COOPERATIVE and COMPETITIVE schemes, when $\sigma_p < 0.01$, the algorithm will bring σ_p back to 0.01 so that the supply price of request do not become zero or negative.

The main objective of the consumer agent is to maximize the number of successful admission of requests submitted by its users given the available budget. To do so, it needs to find a bid for each request that is just sufficient to surpass the admission (supply) price generated by the provider agents. The pseudocode in Figure 7.3 explains how each consumer agent adaptively manipulates its price index to serve this purpose.

If DB is *LOW*, it suggests that there is sufficient budget for future use, and thus the demand price can be scaled higher to improve the consumer agent's chance of beating the admission prices generated by the provider agents. This signals the policing scheme to increase σ_c . If DB is *HIGH*, the opposite action is taken because the consumer agent must try to limit its expenditure either because the average demand price index is too high or there is a likelihood of insufficient tokens. If DB is *MED*, the consumer agent uses SD to match the supply conditions. If SD is *LOW*, it is indicative that the average supply price is below the demand price. In this case, the price index will be decreased. If SD is *HIGH*, the opposite action is taken.

```

if  $DB = LOW$  then
   $\sigma_c = \sigma_c + \eta$ 
else if  $DB = HIGH$  then
   $\sigma_c = \sigma_c - \eta$ 
else if  $DB = MED$  then
  if  $SD = LOW$  then
     $\sigma_c = \sigma_c - \eta$ 
  else if  $SD = HIGH$  then
     $\sigma_c = \sigma_c + \eta$ 
  end if
end if
if  $\sigma_c < 0.01$  then
   $\sigma_c = 0.01$ 
end if

```

Figure 7.3: Policy Scheme for Consumer Agents

Similar to the policy scheme for provider agents, when $\sigma_c < 0.01$, the consumer agent must bring the index back to 0.01 so that the bids for all requests will have a positive value.

With these policies in place, information on the degree of competition (SD) and relative available budget (DB and SB) is used as a feedback to facilitate the process of adapting the price indices in order to regulate the circulation of tokens among participating organizations.

7.3.3 Formulation of Bids and Admission Price

Consumer agents formulate the bid price for each request by attempting to track the overall degree of contention for shared resources via σ_c . Each consumer agent substitutes $\frac{\Delta C}{C}$ in eqn 4.4.4 with σ_c which results in eqn 7.3.3.

$$\frac{PE_j T_j}{\omega \tau} \left[\sigma_c + \frac{1}{2C} (\omega + PE_j) \right] \quad (7.3.3)$$

On the other hand, for each provider agent, the price index is employed as a factor that is multiplied with the actual pricing framework to re-scale the admission price. This

is done so that the overall degree of competition for resources based on the data obtained from the accounting ratios can be reflected. Hence the extended pricing framework for each provider agent is as shown in eqn 7.3.4

$$\sigma_p \left[\frac{PE_j T_j}{2\omega\tau C} (2\Delta C + \omega + PE_j) \right] \quad (7.3.4)$$

7.4 Experimental Procedure

7.4.1 Performance Metrics

- System Admission Ratio

It is calculated by taking the ratio between the total number of admitted requests divided by the total number of submitted requests – the same as the one defined in chapter 4.

- Fairness

We use the same metric as eqn 6.2.5 already defined in chapter 6.

7.4.2 Variables

- Load-contribution ratio (LC)

The load-contribution ratio (LC) is the same as the one defined in chapter 4.

- Degree of Participation (N)

The degree of participation, N, represents the total number of participating organizations in the community for each simulation run.

- Budget ($B_{k,0}$)

The initial quantity of tokens that are assigned to organization k at $t = 0$. In the experiments, the amount of assigned tokens is the same for all organizations.

We use the KTH SP2 Skew-02 (refer to Table 4.3) workloads for all experiments.

7.5 Results

Figure 7.4 shows the impact of the initial budget on the system admission ratio for the different schemes under different load-contribution ratios and degree of participation. The UNIFORM scheme is based on the original token-exchange incentive scheme where σ_p is kept at 1.0 throughout the entire simulation. For the UNIFORM scheme, when $LC = 0.4$ for both $N = 2$ and $N = 5$, the system admission ratio, as shown in Figure 7.4, improves with increasing amount of tokens assigned to each organizations. When the budget is increased from 50 to 1000, the system admission ratio increases from 0.35 to 0.62. When $LC = 2.0$, the same trend is observed for the system admission ratio. However, the overall performance is weaker than that of $LC = 0.4$ for both $N = 2$ and $N = 5$. This is because there are less resources shared to the community.

In terms of fairness, as shown in Figure 7.5 a reverse trend is observed when compared with Figure 7.4. When $LC = 0.4$, it deteriorates (becomes larger) when the budget is increased from 50 to 1000. The situation is more severe when $LC = 2.0$. In addition, fairness is also influenced albeit weakly by the degree of participation (N). When $N = 2$, the value increases from 0.95 to 1.15, when budget is increased from 50 to 1000. When $N = 5$, the values are between 1.4 to 1.8. It is generally difficult to achieve fairness when there is lower degree of resource contribution because there is a higher chance for any organization to dominate the use of shared resources especially when the initial assigned amount of tokens is high (e.g. when budget = 1000). From the results, it can be observed that the optimal results, with regard to the joint performance of both fairness and admission ratio, occurs when budget is between 200 to 500.

The COOPERATIVE scheme on overall, gives the best and most consistent performance for fairness and the system admission ratio. Consistency⁴ is the largest absolute difference in values obtained from either fairness or system admission ratio between the entire range of initial tokens assigned for a given load-contribution ratio (LC) and degree

⁴We measure consistency in the same way as in chapter 4, page 91.

Figure 7.4: Impact of Initial Budget on System Admission Ratio for LC=0.4 (top), N=2 (left) and N=5 (right) and LC=2.0 (bottom), N=2 (left) and N=5 (right)

Figure 7.5: Impact of Initial Budget on Fairness for $LC=0.4$ (top), $N=2$ (left) and $N=5$ (right) and $LC=2.0$ (bottom), $N=2$ (left) and $N=5$ (right)

of participation (N). Hence, lower values reflect better consistency. At $LC = 0.4$, the consistency in terms of the system admission ratio, is 0.03. UNIFORM performs better (0.25) than COMPETITIVE (0.3) for both $N = 2$ and $N = 5$. When $LC = 2.0$, COOPERATIVE still gives the best overall performance for admission ratio when $N = 2$ and $N = 5$. However, the consistency is now 0.07. This indicates that COOPERATIVE is less effective when resource contributions are lowered.

In terms of fairness, COOPERATIVE also gives the best and most consistent results when $LC = 0.4$. The maximum value does not exceed 0.1 irrespective of the initial budget allocated. Furthermore, the consistency is almost zero for both $N = 2$ and $N = 5$. The performance difference for fairness between COOPERATIVE and the other schemes also becomes less significant when $LC = 2.0$. Although the overall performance results for different assigned budgets are still better than UNIFORM, COOPERATIVE does more poorly in terms of consistency compared to the case when $LC = 0.4$. These results demonstrate the effectiveness of the scheme to adapt to different degrees of contention when applying the token-exchange incentive scheme for resource management.

The COOPERATIVE scheme becomes less effective in preserving performance consistency with less contribution of resources. This is because with increased competition for resources, it creates a higher likelihood for some organizations to gain an advantage over others. This has adverse effects on both the system admission ratio and fairness.

By reversing the strategies for manipulating the price indices of each provider agent, as in the case for COMPETITIVE, we show that it gives performance results that are in general even poorer than UNIFORM for both the system admission ratio and fairness when $LC = 0.4$. The results for fairness deteriorates significantly when budget is 1000 for both $N = 2$ and $N = 5$. Because each organization attempts to maximize its own earnings, it renders one or more organizations, at any one time, the ability to dominate the use of shared resources when they have an opportunity to amass a significant amount of tokens. When $LC = 2.0$, the trend that COMPETITIVE performs worse than UNIFORM

still persists but becomes less prominent. Both have poor performance in this case.

The movement of the price indices, under the COOPERATIVE scheme, are strongly dependent on both the initial assigned budget and the load-contribution ratio. The budget constitutes the degree of contention perceived by both provider and consumer agents because it is accounted by both SB and DB , which affects the demand and supply price index. The load-contribution ratio, on the other hand, has an influence on the likelihood of physical contention of the shared resources, and therefore, has a larger impact on the supply price index. Figure 7.6 and Figure 7.7 show the movement of price indices across an entire simulation run for both consumer and provider agents with initial budget of 50 and 1000 assigned to each organization with respect to different load-contribution ratios.

When budget is 50 and $LC = 2.0$ (Figure 7.6), that is, with a higher likelihood of contention, provider agents compete intensely. This is evidenced by a higher frequency of price movements with larger magnitudes. In contrast, when $LC = 0.4$, ‘spikes’ with relatively lower magnitudes and shorter time intervals are observed. With less tokens in circulation, consumer agents also tend to be risk averse on their expenditure by attempting to drive their demand price index down. As indicated, for both $LC = 2.0$ and $LC = 0.4$, the demand price index rarely exceeds 0.5.

As shown in Figure 7.7, when the initial assigned budget is 1000, the demand price index does not fluctuate as much compared to the case when the budget is 50. When $LC = 0.4$, they inflate almost linearly with respect to the simulation time. This is because with increased contribution of shared resources, earnings need not be replenished since provider agents try to bring the supply prices index to a very low level. The inflation results from the fact that DB and SB remains *LOW* at most of the time. However, when $LC = 2.0$, there is first an initial inflation, which is then followed by oscillations in the demand price index. This is because more tokens are exchanged when the likelihood of contention increases with less shared resources. The oscillations are

LC=0.4

LC=2.0

Figure 7.6: Demand and Supply Price Indices of COOPERATIVE for $N=5$ and Budget = 50, with $LC=0.4$ (top) and $LC=2.0$ (bottom)

LC=0.4

LC=2.0

Figure 7.7: Demand and Supply Price Indices of COOPERATIVE for $N=5$ and Budget = 1000, with $LC=0.4$ (top) and $LC=2.0$ (bottom)

caused by the fluctuating availability of budget when tokens are in circulation among different organizations.

For the same load-contribution ratio, a lower budget incurs higher degree of fluctuations on the supply price indices of all organizations. In Figure 7.6 (top), when the budget is 50 and $LC=0.4$, there are more ‘spikes’ as compared to the case when the budget is 1000. Similarly, when $LC=2.0$, the fluctuations in the supply price indices when the budget is 50 are much larger as compared to the case when the budget is 1000.

Figure 7.6 and Figure 7.7 demonstrate how provider and consumer agents can autonomously reach an equilibrium in terms of manipulating their prices with respect to different initial budgets and load-contribution ratios. This outcome is instrumental to the COOPERATIVE scheme’s ability to maintain performance consistency for both the system admission ratio and fairness.

7.6 Summary and Conclusions

As demonstrated from our simulation results, one practical difficulty with the token-exchange incentive scheme is its performance sensitivity on the trade-off between fairness and the system admission ratio due to the initial quantity of tokens that is assigned to each organization. This is because the traditional token-exchange incentive scheme is not designed to handle fluctuating demands (as observed from supercomputer workload traces) for resources which results in sporadic resource contention.

We resolve this problem by employing accounting ratios as a means to gauge the instantaneous degree of contention at each organization. Policies are built into each agent to respond to the values of these accounting parameters to manipulate their respective price indices of either the demand and supply price. From our results, we show that the trade-off between fairness and admission ratio becomes less sensitive to different quantity of tokens assigned to each organization if the COOPERATIVE scheme is adopted. This is provided that each organization contributes sufficient resources to the commu-

nity. Finally, all organizations must adhere to the same policies for trading resources. This is necessary to prevent agents from adopting strategies such as the COMPETITIVE scheme, because it can potentially lead to performance results that will be even poorer than UNIFORM (employing the token-exchange incentive scheme without any price adaptation).

Chapter 8

Conclusions

8.1 Summary of Contributions

The primary aim of this thesis is to address the problem of resource contention in a virtual organization at both the user and inter-domain level. First of all, because users can have diverse quality-of-service requirements, there is a need to reduce the likelihood that any user can potentially deprive resource access to a large number of other users. Secondly, the aggregate demand for resources by the users from an organization can lead to inter-domain level contention when the organization does not contribute sufficient computing resources. These issues arise as a consequence of free-riding behavior amongst users that must be addressed by employing an admission control mechanism for resource management.

Before exploring strategies to address the above two issues, we developed a conceptual model for the grid giving focus to two critical features: quality-of-service and advance reservation mechanisms. Quality-of-service is needed since the grid has to support a large group of users that are likely to have diverse resource requirements. An advance reservation is also required to support resource allocation for users' applications that may require concurrent access to computing resources at different geographical sites during their execution lifetime.

From the literature review, we showed that user-initiated and inter-domain contention management are traditionally treated as two separate issues, each of which can be independently dealt with by a host of techniques. As such, we also discuss the plausibility of the token-exchange incentive scheme as a means to unify the process of managing both user-initiated and inter-domain contention.

The main research work of this thesis deals with the design of a pricing framework that extends the token-exchange incentive scheme so that it can be incorporated to support the administration of both user-initiated and inter-domain level contention.

The pricing framework is capable of preventing users from oversubscribing for shared resources because the admission price generated is relative to the quality-of-service demand of the user's request and the instantaneous degree of contention (utilization) on the provider's shared resources. Since organizations are assigned a fixed amount of tokens proportional to their resource contribution, their respective consumer agents will autonomously limit their expenditure of tokens on each request. The experimental analysis on competitive consumers (i.e., consumers that limit their expenditure within a fraction of their budget) show that by incorporating the pricing framework to support the admission of advance reservation requests, the system is also capable of achieving more consistent performance on the system admission ratio when compared with other benchmarks (e.g., MSB, MRT and QOPS).

Our preliminary analysis on the VO parameters that are critical to the cause of inter-domain contention has demonstrated the significance of each organization's load-contribution ratio and its influence on the degree of contention in the VO. The token-exchange incentive scheme has a clear advantage over the centralized approach because it eliminates the need to maintain the accounting records of executed jobs in order to administer the policies for keeping the load-contribution ratio of each organization below a pre-defined threshold. It resolves this problem by ensuring that the earnings of each organization is consistently proportional to its underlying contribution of resources. Our

mathematical analysis of the token-exchange incentive scheme has shown that this is achievable provided that all provider agents use the same pricing framework to admit requests. In addition, the matchmaking policy at the broker must select the candidate service that gives the maximum payoff to the request. Finally, to seek a balance between fairness and the system admission ratio, we introduced mechanisms to dynamically adapt the bid and admission prices of requests and services respectively, so that the trade-off between the performance metrics for user-initiated and inter-domain level contention become less sensitive to the initial number of tokens assigned to each organization.

8.2 Concluding Remarks and Future Directions

8.2.1 The Pricing Framework and its Current Limitations

The key to unifying the management of user-initiated and inter-domain contention lies with the pricing framework, which constitutes the main extension of the token-exchange incentive scheme so that it can operate in a generic grid computing infrastructure. In this section, we will discuss several unexplored areas that has deliberately been isolated from the analysis for the purpose of generalizing our solution to the problem tackled in this thesis.

Resource Heterogeneity

The pricing framework only takes into account the instantaneous utilization of resources as a means to quantify the degree of contention of each provider agent's underlying resources. It does not employ any additional features to account for a grid with computing nodes having different processing capability. Resource heterogeneity is inherent of a grid infrastructure because resources are typically contributed by different organizations. The presence of resource heterogeneity is a challenge because the utilization of contributed resources depends on the consumer workload demands, which in turn have a significant impact on the system admission ratio and fairness. To further elaborate, we consider 3

organizations that contribute the same number of computing nodes but with different processing capability. We assume that both organizations 1 and 2 have contributed resources that are able to meet the resource specifications (e.g., in terms of processor speed, memory, data storage) of a majority of users. On the other hand, resources contributed by organization 3 are not utilized due to the lack of certain functional requirements to meet the needs of a majority of users. The end effect is that, since the resources of organization 3 are unlikely to be utilized even in the event of contention, it will not earn sufficient tokens for its users to gain any access to the shared resources. As such, the users of organization 3 will be automatically forced out of participation.

While the above illustration serves to demonstrate the utility of the token-exchange incentive scheme to promote fair share amongst participating organizations, the main point of emphasis is that resources heterogeneity presents itself as a social externality that can only be addressed by human intervention. Referring back to the above illustration, if organization 3 wants to improve the opportunity of its users to gain access to resources, the administrators must upgrade their resources to meet the quality-of-service requirements of the majority of users.

At large, managing contention is a combined process of admitting and distributing workloads to achieve a set of VO-wide performance metrics. The presence of heterogeneous resources either creates additional constraints or restrictions to the extent at which a pre-defined set of policy structures can achieve the desired goals for resource management. Hence, a possible future extension of this work is to explore how different facets of resource heterogeneity can impact VO-level performance goals on managing resource contention.

Non-deterministic Resource Availability

So far, the analysis carried out has been focused on a fixed contribution of resources. In a large-scale distributed computing infrastructure, the grid is prone to node-level

unavailability for reasons arising from network failure, system downtime either due to node failure or maintenance. In such circumstances, it is not possible for each organization to maintain the same quantity of resource contribution at all times. As such, the system is faced with several resource management challenges. If resource availability becomes non-deterministic, each site-level resource manager must be able to predict ahead of time the amount of available resources with a certain degree of accuracy, so that, the system is able to re-allocate the remaining resource to jobs in the event of node failure. For this reason, the granted quality-of-service parameters to each admitted request has to be stochastic. Each provider agent must take this factor into account when generating the admission price for requests since the granted quality-of-service of some requests has to be re-negotiated.

Therefore, there is also a possibility that the token-exchange incentive scheme needs to be further improved so that discounts are given to users that are willing to give up a portion of resources that have already been allocated to them. Conversely, users may also need to bear an additional premium if they do not permit any re-negotiation on their quality-of-service requirements in the event of contention. Hence, another interesting extension to this work is to study the impact of the quality-of-service negotiations arising from non-deterministic availability of resources on achieving the VO-level objectives.

Unequal Resource Contribution

In chapter 7, we reduced the trade-off between fairness and the system admission ratio, due to the sensitivity of the initial number of tokens assigned to each organization. However, by doing so, the earnings gained by an organization may not always be proportional to its contribution. This is likely to weaken the token-exchange incentive scheme's effectiveness to achieve fairness. Hence, additional analysis is required to further assess the degree of unfairness faced by organizations that contribute more resources relative to other organizations. However, it is difficult to draw a clear line on this issue because there is a possibility of improving the system admission ratio when some degree of unfairness

can be tolerated by the participating organizations. If in the event that the prevailing level of unfairness creates a conflict amongst participating organizations, an effective way to treat this problem is to establish internal policies to limit the range at which the price indices can deviate from their default value (i.e., 1.0).

8.2.2 Relationship between VO-level Objective and System Architecture Design

Our analysis in chapter 6 has shown several properties that the conceptual grid framework must adhere to in order to achieve the optimal measure for fairness. These properties assert that both consumer and provider agents must be homogeneous because all agents must comply with the same policies for generating prices to trade computing resources. In chapter 7, we have characterized agent homogeneity in the following perspectives:

1. Consumer agents of each organization place their bids as a fraction of the available tokens for expenditure.
2. Provider agents employ the same pricing framework to administer the process of admitting requests to utilize their underlying computing resources.
3. Both provider and consumer agents rely on the same set of accounting ratios and policies to generate a set of actions to manipulate their respective price indices.

Finally, the broker's matchmaking policy is configured to choose from a set of candidate services generated by all organizations, that gives the request the highest payoff.

The above requirements demonstrate the fact that the need to achieve fairness has a significant influence on the overall system architecture of the grid infrastructure. In a consortium of physical organizations that collaborate to share their computing resources, the above characteristics are relatively easy to enforce since participation is likely to be on a permanent basis.

However, in a large scale peer-to-peer grid in which the token-exchange incentive scheme has traditionally been employed, such policies cannot be enforced because the system will not scale due to the need to introduce a governing authority for the match-making policies at the broker. As such, the design and analysis of relatively weaker but scalable incentive schemes on peer-to-peer systems, have largely relied on game theoretic techniques to study the emergent behavior of competing agents to evaluate the social outcomes produced by the incentive schemes. To date, there is no concrete solution to achieve fairness without a governing authority.

Appendix A

Derivation of the Pricing Framework

We show how to obtain eqn 4.4.3 based on the arithmetic sum constructed by taking into account the opportunity cost for a provider agent to admit an arbitrary request.

$$\Gamma_{\omega} = \left[\frac{\Delta C + \omega}{C} + \frac{\Delta C + 2\omega}{C} + \dots + \frac{\Delta C + PE_j}{C} \right]$$

First, we group together the series of $\frac{\Delta C}{C}$ by multiplying it with $\frac{PE_j}{\omega}$. On the right hand side, we sum the ratios of the progressive multiples of ω until it reaches $\frac{PE_j}{C}$.

$$\Gamma_{\omega} = \left[\frac{\Delta C}{C} \cdot \frac{PE_j}{\omega} + \frac{\omega}{C} \sum_{i=1}^{\frac{PE_j}{\omega}} i \right]$$

Since PE_j may not be a multiple of ω , we compute the summation by applying the formula for the arithmetic sum on the right hand side. We get the following solution.

$$\Gamma_{\omega} = \left[\frac{\Delta C}{C} \cdot \frac{PE_j}{\omega} + \frac{\omega}{C} \left(\frac{PE_j}{2\omega} + \frac{PE_j^2}{2\omega^2} \right) \right]$$

With further simplification we can then obtain

$$\Gamma_{\omega} = \frac{PE_j}{2\omega C} [2\Delta C + \omega + PE_j]$$

Appendix B

Mathematical Workload Model for Observing the Effect of Job Arrival Correlation on Resource Contention

The request arrival profile for the entire community is as given in eqn B.0.1. The derivation of this workload model is obtained from Kleban et. el. [55].

$$w_H(t) = \frac{X(t, H)}{\max(X(t, H))} \quad (\text{B.0.1})$$

$X(t, H)$ defines the mathematical model of a Fast-fractional Gaussian Noise Generating (FGN) function. It is used to define the extent of correlations in the arrival time of requests submitted by the users. We obtained the profile of the arrival process by dividing it with the maximum number of arrivals within the working range of H . It is termed as the ‘Hurst’ exponent – it is a measure of arrival time correlation of the incoming events. Values of $H > 0.5$ indicate long-range dependencies and values $H < 0.5$ indicate anti-correlations. For our work, we used $0.6 < H < 0.9$ to express the correlations in request arrivals.

The FGN function consists of two components, a low frequency and a high frequency component:

$$X(t, H) = X_l(t, H) + X_h(t, H) \quad (\text{B.0.2})$$

The low frequency component is generated according to a Gauss-Markov process as given by,

$$X_l(t, H) = \sum_{k=1}^N W_k G(t, r(k)) \quad (\text{B.0.3})$$

The parameter N defines the number of Markov-Gauss process to include in the history, $N = \log(QT)/\log(B)$ with $1.1 < Q < 2$ and $B = 20$ being user-defined parameters and T being the length of the time series. The weight for each component is given by:

$$W_k = \sqrt{\frac{H(2H-1)(B^{1-H} - B^{H-1})B^{-2k(1-H)}}{\Gamma(3-2H)}} \quad (\text{B.0.4})$$

The stochastic term is given by,

- $G(t=1, r(k)) = G_k(1)$
- $G(t > 1, r(k)) = r(k)G(t-1, r(k)) + \sqrt{1-r(k)^2}G_k(t)$

with $r(k) = \exp(-B)^{-k}$ and $G_k(t)$ a random Gaussian number. The high frequency component is proportional to a Gaussian process:

$$X_h(t, H > 0.5) = \sqrt{1 - \frac{B^{H-1}}{4(1-H)\Gamma(-2H)}} G(t) \quad (\text{B.0.5})$$

The workload model was compared by Kleban et. al. with jobs logs of computing clusters having a total of 4662 CPUs for over 8,171 jobs over a period of 83 days [55]. From calculations conducted, it was found that the arrival profile of jobs matched very closely to the FGN function.

Appendix C

Derivation of Linearity in Earnings on Resource Contribution

From pricing framework, we have show that the prices for the first and subsequent services for any organization k are as given:

$$p_0^s = \frac{PE_j T_j (\omega + PE_j)}{2\omega\tau C_k}$$

$$p_n^s = \frac{n (PE_j)^2 T_j}{2\omega\tau C_k} + p_0^s$$

Also, the total earnings gained by organization k is $B_{k,t}^+ = \sum_{i=0}^{\zeta_k-1} p_i^s$. Therefore,

$$\begin{aligned} B_{k,t}^+ &= p_0^s + p_1^s + \cdots + p_{\zeta_k-1}^s \\ &= p_0^s + \left[p_0^s + \frac{(PE_j)^2 T_j}{2\omega\tau C_k} \right] + \cdots + \left[p_0^s + \frac{(\zeta_k - 1) (PE_j)^2 T_j}{2\omega\tau C_k} \right] \\ &= \zeta_k p_0^s + \frac{(PE_j)^2 T_j}{2\omega\tau C_k} \sum_{i=0}^{\zeta_k-1} i \\ B_{k,t}^+ &= \zeta_k p_{k,0}^s + \frac{(PE_j)^2 T_j}{2\omega\tau C_k} \left[\frac{(\zeta_k - 1) (\zeta_k)}{2} \right] \end{aligned}$$

$$B_{k,t}^+ = \zeta_k p_{k,0}^s + \frac{(PE_j)^2 T_j}{2\omega\tau C_k} \left[\frac{\zeta_k^2 - \zeta_k}{2} \right]$$

Re-arranging, we get

$$B_k^+ = \zeta_k \left\{ \frac{(PE_n)^2 T_n}{\omega\tau C_k} \left(\frac{\zeta_k - 1}{2} \right) + p_{k,0}^s \right\}$$

List of Publications

Journals

1. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “Workload Management of Cooperatively Federated Computing Clusters”, accepted by Journal of Supercomputing : Special Issue on New Trends in Parallel and Distributed Computing and Networking. Vol. 36, No. 3, pp. 309–322, Jun 2006.

Conferences

1. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “Adaptive Policing for Token-exchange based Resource Management”, submitted to 6th IEEE International Symposium on Cluster and Grid Computing (CCGrid 2006), Singapore, May 2006.
2. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “Addressing Sporadic Contention on Shared Computing Clusters”, in Procs. of the 8th International Conference/Exhibition on High Performance Computing in Asia-Pacific Region (HPCAsia 2005), Beijing, China, 30 Nov 2005 – 3 Dec 2005.
3. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “Employing Economics to Achieve Fairness in Usage Policing of Cooperatively Shared Computing Resources”, in Procs. of Workshop on Agent-based grid Economics (in conjunction with the CC-Grid2005), Cardiff, UK, May 2005.
4. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “Managing Irregular Workloads of Cooperatively Shared Computing Clusters”, in Procs of 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA’04), pp.625–634, Hong Kong, China, 13-15 Dec 2004 (published by Springer as LNCS, Vol.3358/2004).
5. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “An Analysis of Resource Sharing in Grid Computing Environments”, in Procs. of Workshop on Grid Computing and

PhD Thesis

LIST OF PUBLICATIONS

Applications (in conjunction with 2004 International Conference on Scientific and Engineering Computation), Singapore, Jun 2004.

6. Percival Xavier, Wentong Cai, and Bu-Sung Lee, “A Decentralized Scheduler for a grid-based Clearinghouse”, in Procs. of Workshop on Massively Parallel Processing (in conjunction with 17th International Parallel & Distributed Symposium), Nice, France, Apr 2003.

Bibliography

- [1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. *Proceedings of the IEEE Real-Time Systems Symposium*, 27(2), 1998.
- [2] R. Al-Ali, K. Amin, G. Laszewski, O. Rana, and D. Walker. An ogsa-based quality of service framework. In *Second International Workshop on Grid and Cooperative Computing (GCC2003)(Talk)*, Dec 2003.
- [3] A. Alhusaini, V. Prasanna C., and P. Raghavendra. A framework for mapping with resource co-allocation in heterogeneous computing systems. In *9th Heterogeneous Computing Workshop*, May 2000.
- [4] N. Andrade, W. Cirne, and F Brasileiro. Our Grid: An approach to easily assemble grids with equitable resource sharing. *9th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 53–68, Jun 2003.
- [5] S. Androutsellis-Theokotis. A survey of peer-to-peer file sharing technologies. White Paper , Athens University of Economics and Business, 2002.
- [6] D. Arnold, H. Casanova, and J. Dongarra. Innovations of the NetSolve Grid Computing System. *Concurrency: Practice and Experience*, 14(13–15):1457–1479, 2002.
- [7] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure*, Oct 2004.

-
- [8] H. Aydt. Simulation-aided decision support for cooperative resource sharing. Master of science thesis, KTH Information and Communication Technology, 2006.
 - [9] A. G. Barto and P. Singh. On the computational economics of reinforcement learning. In Morgan Kaufmann, editor, *Connectionist Summer School*, pages 35–44, 1990.
 - [10] J. Basney. Network and CPU co-allocation in high throughput computing environments. citeseer.nj.nec.com/basney01network.html.
 - [11] J. Basney and M. Livny. *High Performance Cluster Computing*, volume 1, chapter 5. Prentice Hall PTR, May 1999.
 - [12] The Beowulf Project Website. <http://www.beowulf.org/>.
 - [13] Berkeley NOW Website. <http://now.berkeley.edu/>.
 - [14] A. Bondavalli, F. Di Giandomenico, and I. Mura. Value-driven resource assignment in object-oriented real-time dependable systems. In *Third IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 97)*, pages 92–99, Newport Beach, California, USA, February 6-7 1997.
 - [15] J. Bredin, D. Kotz, and D. Rus. Utility driven mobile-agent scheduling. Technical Report PCS-TR98-331, Dartmouth University, 2003.
 - [16] M. Brune, J. Gehring, A. Keller, and A. Reinefeld. Managing clusters of geographically distributed high-performance computers. *Concurrency – Practice and experience*, 11(15):887–911, 1999.
 - [17] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.
 - [18] Andrew Bye. A comparison among bidding algorithms for multiple auctions. In *Agent Mediated Electronic Commerce on Agent-Mediated Electronic Commerce IV*,
-

- Designing Mechanisms and Systems (AAMAS '02)*, pages 1–16. Springer-Verlag, 2002.
- [19] J. Cao. *Agent-Based Resource Management for Grid Computing*. PhD Thesis, University of Warwick, Sep 2001.
- [20] N. Chan, B. LeBaron, A. Lo, and T. Poggio. Agent-based models of financial markets: A comparison with experimental markets, 1999. <http://people.brandeis.edu/~blebaron/wps/disagg.pdf>.
- [21] S. Chaplin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in legion. *Future Generation Computer Systems*, 15(5–6):583–594, 1999.
- [22] B. Chun and D. E. Culler. REXEC: A decentralized, secure remote execution environment for clusters. In *Communication, Architecture, and Applications for Network-Based Parallel Computing*, pages 1–14, 2000.
- [23] B. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report CSD1092, University of California Berkeley, Jan 2002.
- [24] B. Chun, F. Yun, and A. Vahdat. Bootstrapping a distributed computational economy with peer-to-peer bartering. In *Proceedings of 1st Workshop on Economics of Peer-to-Peer Systems*, Jun 2003.
- [25] W. Cirne and K. Marzullo. The computational co-op: Gathering clusters into a meta-computer. In *13th International Parallel Processing Symposium*, page 160, Puerto Rico, Apr 1999. IEEE.
- [26] K. Czajkowski, Foster I., and C. Kesselman. Resource co-allocation in computational grids. In *International Conference for High Performance and Distributed Computing*, 1999.

-
- [27] D. Irwin, L. Grit, and J. Chase. Balancing risk and reward in a market-based task service. In *13th IEEE International Symposium on High Performance Distributed Computing*, pages 160–169, Jun 2004.
- [28] e-Donkey website. <http://www.edonkey2000.com/>.
- [29] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. Enhanced algorithms for multi-site scheduling. *GRID 2002*, pages 219–231, 2002.
- [30] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In *9th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 166–182, 2003.
- [31] C. A. Waldspurger et. al. Spawn: A distributed computational economy. *Journal for Software Engineering*, 18(2):103–117, 1992.
- [32] F. Berman et. al. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(4):369–382, 2003.
- [33] D. Feitelson. Parallel workload archive. <http://www.cs.huji.ac.il/labs/parallel/workload/>.
- [34] W. Fellows. Grids: Underpinning utility computing, Jan 2003. <http://searchdatabase.techtarget.com/originalContent/>.
- [35] I. Foster and Iamnitchi A. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems*, 2003.
- [36] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.
- [37] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *8th International Workshop on Quality of Service*, 2000.
-

-
- [38] E. Fulp and D. Reeves. Distributed network flow control based on dynamic competitive markets. In *International Conference on Network Protocols*, pages 13–16, Oct 1998.
- [39] E. Fulp and D. S. Reeves. Optimal provisioning and pricing of Internet differentiated services in hierarchical markets. *Lecture Notes in Computer Science*, 2093:409–420, 2001.
- [40] G. Garimella. Advance cpu reservations with the dynamic soft real-time scheduler, 1999.
- [41] The Globus Project. <http://www.globus.org/>.
- [42] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *Proceedings of the 2001 ACM Conference on Electronic Commerce*, 2001.
- [43] J. Haritsa, M. Carey, and M. Livny. Value-based scheduling in real-time database systems. *The VLDB Journal*, 2(2):117–152, 1993.
- [44] M. He and N. Jennings. Designing a successful trading agent: A fuzzy set approach. *IEEE Transactions on Fuzzy Systems*, 12(3):389–410, 2004.
- [45] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Escrow Services and Incentives in Peer-to-Peer Networks. 3rd ACM Conference on Electronic Commerce*, 2001.
- [46] M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC resource management systems: Queueing vs. planning. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–20. Springer Verlag, 2003.
- [47] J. Hu and M. Wellman. Online learning about other agents in a dynamic multiagent system, 1998.
-

-
- [48] J. In, P. Avery, and S. Ranka R. Cavanaugh. Policy based scheduling for simple quality of service in grid computing. In *18th International Parallel and Distributed Processing Symposium*, Apr 2004.
- [49] M. Islam, Balaji P., Sadayappani P., and D. K. Pandai. QoPS: A QoS based scheme for parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing: 9th International Workshop*, Jun 2003.
- [50] W. Jiang and H. Schulzrinne. Qos measurement of internet real-time multimedia services. In *Proceedings of NOSSDAV*, Jun 2000.
- [51] Jxta Website. <http://www.jxta.org/>.
- [52] Kaaza Website. <http://www.kazaa.com/us/index.htm>.
- [53] M. Karul. *Metacomputing on the World Wide Web*. PhD Thesis, New York University, May 1998.
- [54] K. Kim and K. Nahrstedt. A resource broker model with integrated reservation scheme. *IEEE International Conference on Multimedia and Expo (ICME 2000)*, 2:859–862, 2000.
- [55] S. Kleban and S. Clearwater. Quelling queue storms. In *13th International Conference High-performance and Distributed Computing*, Jul 2003.
- [56] Y. K. Kwok, S. Song, and K. Hwang. Selfish grid computing: Game-theoretic modeling and nas performance results. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, May 2005.
- [57] K. Larson and T. Sandholm. Miscomputing ratio: social cost of selfish computing. In *Second international joint conference on Autonomous agents and multiagent systems (AAMAS '03)*, pages 273–280, 2003.
-

-
- [58] X. Li and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.
- [59] D. Little. Who Needs Supercomputers?, 2002. <http://www.businessweek.com/magazine/content/>.
- [60] IBM’s Loadleveler Website. <http://www-1.ibm.com/servers/eserver/pseries/library/spbooks/loadleveler.html>.
- [61] LSF Website. <http://www.platform.com/products/LSF/>.
- [62] R. Min and M. Maheswaran. Scheduling advance reservations with priorities in grid computing systems. In *Thirteenth IASTED International Conference on Parallel and Distributed Computing Systems (PDCS '01)*, pages 172–176, Aug 2001.
- [63] Napster Website. <http://www.napster.com/>.
- [64] A. Odlyzko. Paris metro pricing for the internet. In *Proceedings of ACM Conference on Electronic Commerce (EC'99)*, pages 140–147. ACM, 1999.
- [65] S. Ontann and E. Plaza. Cooperative case bartering for case-based reasoning agents. In *5th Catalanian Conference on Artificial Intelligence*, volume 2504, pages 294–308, Oct 2002.
- [66] D. Parkes. iBundle: An efficient ascending price bundle auction. In *ACM Conference on Electronic Commerce (EC-99)*, pages 148–157, Nov 1999.
- [67] D. Parkes and J. Rey. Distributed implementations of vickrey-clarke-groves mechanisms. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems*, 2004.
- [68] Portable Batch System Website. <http://www.openpbs.org/>.

-
- [69] K. Ramamritham, J. A. Stankovic, and P. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(2), Apr 1990.
 - [70] M. Rappa. A Review of Technology, Innovation and Management, Nov 1994. North Carolina State University.
 - [71] J. Rashid, A. Al, R. F. Omer, and D. W. Walker. G-QoS: Grid service discovery using qos properties. In *UK e-Science All Hands Meeting*, Sep 2003.
 - [72] O. Regev and N. Nisan. The POPCORN market – an online market for computational resources. In *ICE '98: Proceedings of the first international conference on Information and computation economies*, pages 148–157, 1998.
 - [73] M. Rui and M. Maheswaran. Scheduling co-reservations with priorities in grid computing systems. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, May 2002.
 - [74] H. Sanneck. Packet loss recovery and control for voice transmission over the internet. <http://www.gmd.de/publications/research/2000/008/>.
 - [75] L. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *ACM/IEEE International Symposium on Cluster Computing and the Grid*, May 2001.
 - [76] S. Senapathi, B. Chandrasekaran, and D. K. Panda. QoS middleware for cluster-based servers to support interactive and resource-adaptive applications. In *International Conference High Performance Distributed Computing*, Jun 2003.
 - [77] H. Shan, L. Olikar, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *Supercomputing 2003*, Nov 2003.
 - [78] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *International Parallel and Distributed Processing Symposium*, May 2000.
-

- [79] Solaris MC Website. <http://www.sunlabs.com/research/solaris-mc/>.
- [80] M. S. Squillante, D. Yao, and Z. Li. The impact of job arrival patterns on parallel scheduling. *SIGMETRICS Perform. Eval. Rev.*, 26(4):52–59, 1999.
- [81] Sun Grid Engine 5.3 Website. <http://www.sun.com/software/gridware/sge.html>.
- [82] D. Talby and D. G. Feitelson. Supporting priorities and improving utilization of the IBM SP2 scheduler using slack based backfilling. In *13th International Parallel Processing Symposium*, pages 513–517, Apr 1997.
- [83] C. Tham and Liu Y. Minimizing transmission costs through adaptive marking in differentiated services networks. citeseer.nj.nec.com/tham02minimizing.html.
- [84] K. W. Tse, W. K. Lam, and P. K. Lun. Reservation aware operating system for grid economy. *SIGOPS Operating Systems Rev.*, 37(3):36–42, 2003.
- [85] V. Turner and M. Eastwood. IT Consolidation: Solutions from Hewlett Packard, Jun 2002. <http://mithras.itworld.com/download/HP/IDCitconsolidationWhitepaper1.pdf>.
- [86] Grid Resource Allocation Agreement Protocol Website. <https://forge.gridforum.org/projects/graap-wg>.
- [87] Trading Agent Competition Website. <http://www.sics.se/tac/>.
- [88] B. S. White, M. Walker, M. Humphrey, and A. S. Grimshaw. LegionFS: A secure and scalable file system supporting cross-domain high-performance applications, 2001.
- [89] B. Wilcox-O’Hearn. Experiences deploying a large-scale emergent network. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS ’02)*, 2002.
- [90] W. Wulf. The collaboratory opportunity, Aug 1993.

- [91] M. Wurman. Toward flexible trading agents. In Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 134–140. Stanford, Mar 2001.
- [92] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based CPU load predictions. In *Proceedings of International Parallel and Distributed Processing Symposium*, Apr 2003.
- [93] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *INFOCOM*, pages 679–688, 2001.