A Novel Scheduling Model for Computational Grid Economy Systems

by

Ömer Ozan Sönmez

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical & Computer Engineering

Koç University

August, 2005

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Ömer Ozan Sönmez

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assist. Prof. Attila Gürsoy

Assoc. Prof. Ceyda Oğuz

Assist. Prof. Öznur Özkasap

Date:

To my parents

ABSTRACT

Computational grids have emerged to exploit geographically distributed resources such as clusters or idle personal computers to solve large-scale computational and data demanding scientific problems. It has been considered that developing computational grid economy systems in which users pay for using resources or services, would motivate people to share their resources making the computing power economically available that the communities require. In this thesis, we present a novel economic-based job scheduling heuristic to be used in such a grid system. The heuristic basically tries to complete a sequential workflow or a parameter sweep application using one or more optimization strategies (cost, time or time-cost) according to the deadline and budget constraints of the user. The experimental results reveal that our heuristic outperforms the related heuristics in the literature. Besides, we present two market models, a commodity market and a combinatorial double auction model, that are expected to meet the requirements of the resource owners and users in the economic respect and ensure efficient scheduling in a computational grid economy system. We performed simulation experiments to compare the market models, and the experimental results demonstrate that the models have both advantages and drawbacks in terms of achieving social welfare in the market.

ÖZETÇE

Sayısal Şebeke (Grid) sistemleri atıl halde duran kullanıcı bilgisayarları yada yüksek başarım bilgisayarları gibi coğrafi olarak dağıtılmış kaynakları bir araya getirerek daha yüksek bir hesaplama gücü ortaya çıkarmayı hedefler. Bu yüksek hesaplama gücü büyük ölçekte hesaplama ve veri gerektiren bilimsel problemlerin çözümünde kullanılabilir. Bu sistemlerde piyasa ekonomisi modellerinin kullanılması sayesinde, kişilerin kar maksadıyla daha çok kaynağı paylaşıma açacağı düşünülmektedir. Bu tezde, böyle bir sistemde kullanılmak üzere yeni bir iş zamanlama sezgisel (heuristic) yöntemi geliştirdik. Yöntem kısaca kullanıcının işlerini belirtilen zaman ve para kısıtlarında bitirmeye çalışıyor. Bu amaçla, mevcut zaman ve para miktarını dikkate alarak zaman, para yada her ikisini birden en iyi şekilde kullanmaya çalışıyor. Bu yöntemi benzer çalışmalardaki yöntemlerle, simülasyona dayalı deneylerle karşılaştırıp başarılı sonuçlar elde ettik. Bunun yanı sıra, kaynak sahiplerinin ve kullanıcıların ekonomik yönden beklentilerinin karşılanması ve etkin iş zamanlama sonuçları üretebilmek için mal piyasası ve açık arttırma modellerine dayanan piyasa modelleri geliştirdik. Yaptığımız deneysel çalışmalar bu modellerin birbirlerine karşı olan avantaj ve dezavantajlarını ortaya çıkardı.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor Assist. Prof. Attila Gürsoy, whose experience, understanding and patience added considerably to my graduate experience. I would like to thank the members of my thesis committee for critical reading of this thesis and for their valuable comments. I would also like to thank to my cousin M. Süleyman Mermut, who is a finance specialist, for his contributions to my thesis. I am grateful to all my friends from Koç University, for their close friendship, to name some, Zülküf Genç, Utkan Öğmen, Tayfun Elmas, Bengi Mizrahi, Gökçe Görbil, and Selim Aytuna. I must give immense thanks to my very special friend Feray Dor and to my parents for their continuous support during the course of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

QoS	Quality of Service
WAN	Wide Area Network
BSP	Bulk Synchronous Parallel
MIPS	Million Instructions Per Second
PPMI	Price Per Million Instructions
SPEC	Standard Performance Evaluation Corporation

Chapter 1

INTRODUCTION

Grid computing [1] systems have emerged in the mid-1990s with the aim of enabling the sharing, selection, and aggregation of geographically distributed resources such as clusters, supercomputers, storage systems and data resources to solve large-scale computational and data demanding problems in science, engineering, and industry.

In fact, the name "grid" has been chosen as an analogy to the electric power grids that supply consistent, persistent, and transparent access to electricity, regardless of its source [2]. Similarly, grid systems virtualise the heterogeneous resources, which belong to different administrative domains and are managed by different policies, to the end users [1]. In other words, grid is a distributed environment where users can run their jobs or use services without knowing where the resources are or even who owns them.

To construct a grid system it is significant to have a common middleware, and to benefit from the grid several issues have to be addressed such as site autonomy, heterogeneity, resource allocation or co-allocation, online control and delivery of nontrivial QoS [2]. Globus toolkit [3] is the de facto standard in the grid computing world that addresses many of these issues. Globus toolkit provides a software infrastructure, *i.e.* a middleware, that provides uniform and secure environment for accessing the resources. The toolkit comprises a set of components that implement fundamental services, such as security, resource location, resource management, data management, resource reservation, and communications. Today, several resource management applications and tools such as MPICH-G [4], Cactus [5], Nimrod-G [6], and Condor-G [7] use the Globus toolkit as an infrastructure to be a grid-enabled system.

1.1 Computational Grids

Basically, three types of architectural models have been offered for grid computing [1]; data grids, service grids, and computational grids. In this thesis, we concentrate only on the computational grids.

Computational grids are designed to aggregate the processing power from a distributed collection of processing resources in order to solve distributed parallel computing applications or applications that need to be executed many times with different parameters. A typical example of a computational grid is the SETI@home project [8] in which idle cycles of the personal computers, dispersed over the world, are combined to create a computational grid used to analyze radio transmissions received from outer space.

The main concern in a computational grid is to determine which resource should run or service the job submitted to the system [2]. To make use of a computational grid, a scheduling mechanism is needed that can gather and compare dynamic and static information of the resources and assign each job to the most appropriate resource considering the QoS parameters of the grid users.

Scheduling in the computational grids can be classified as conventional and market-based scheduling [2, 9]. In general, conventional scheduling methods use heuristics to assign jobs to the grid resources; whereas, market-based methods, *i.e.* computational markets, let the software agents exchange resources in a market-like way. Besides, we can classify the work on the computational markets into two main groups [10]: Resource allocation studies where the market is used for maximizing (or minimizing) some global measure, and implementation of real markets where the computational agents represent people or companies.

For the first type of studies, economic markets have been thought as a very natural way to manage large distributed resource allocation problems. However, attaining the optimal resource allocation has not been considered as the only motivation behind applying market-based mechanisms in grid computing. Since there is no strong motivation to make people or companies share their resources in the current grid systems, it has been considered that economic models in which users pay for using resources or services, would give people incentive to share their resources making the computing power economically available that the society requires [11]. This is the mainstream behind developing real market-based grid scheduling and resource management systems or namely grid economies.

1.2 Contributions

In this thesis, we present a novel job scheduling heuristic to be used in a computational grid economy system. It is primarily designed for scheduling sequential workflow type of applications [12] using cost, time and time-cost optimization strategies at each stage considering the deadline and budget constraints. In addition, treating the entire job set as a single stage workflow application it can also deal with parameter sweep applications [13]. Using the GridSim toolkit [14], we performed simulation based experiments in order to compare the proposed heuristic with the similar heuristics that are being used in the Nimrod/G [6] system for scheduling parameter sweep applications.

We also present two market models (a commodity market and an auction model), that are expected to meet the requirements of the resource owners, typically CPU providers, and users in the economic respect and ensure efficient scheduling in a computational grid economy system.

For the commodity market model we consider three pricing policies that the resource owners may use; a fixed pricing policy that actually does nothing but preserve the initial prices, a dynamic pricing policy that adjusts prices according to the supply-demand dynamics, and a stochastic approximation based policy [15], which was actually proposed to be used in the e-commerce applications. In addition, the user preferences are represented by the proposed job scheduling heuristic.

Regarding the distributed system field, in most of the computational market studies which make use of auction models [16, 17, 18, 19], resource owners auction themselves using a central auctioneer, and users desire only one commodity (resource) in each auction session. However, a central mechanism is a point of failure in a distributed system, and generally users need multiple commodities at the same time for their applications (*i.e.* co-allocation) in a computational grid system. Considering these factors, we present a combinatorial double auction model that enables resource owners to perform auctioning on their own, and users to get multiple commodities concurrently.

1.3 Organization

Chapter 2, overviews the grid scheduling subject presenting several conventional and economic based grid scheduling studies, including theoretical and real applications. Chapter 3 introduces the economic-based job scheduling heuristic and presents the experimental results with a detailed discussion. Chapter 4 describes the market models and presents the related experimental results with a detailed discussion. Thesis finalizes by a chapter for conclusion, Chapter 5, and an Appendix section that contains information about the simulation toolkit, and the pseudo codes of the compared heuristics.

Chapter 2

A SURVEY ON GRID SCHEDULING

2.1 *Overview of Grid Scheduling*

Scheduling, is the assignment of tasks to resources over time [20]. In grid computing these tasks refer to computational jobs such as required by protein modeling or docking applications, Monte Carlo simulations, 3D modeling and fluid dynamics, and resources refer to data and processing units such as clusters and supercomputers.

Scheduling has been widely studied in distributed systems before grid systems have emerged; however, grid concept has introduced specific requirements that make grid scheduling different than conventional distributed system scheduling. To begin with, a grid environment spans multiple administrative domains each with its own schedulers and management policies; hence, grid scheduling requires additional mechanisms to find appropriate resources, interact with different local policies and manage execution of jobs on those diverse administrative domains [2]. Second, in cluster or single site scheduling which can be classified under conventional distributed system scheduling, the main goal of the users is to minimize the total execution time, *i.e.* makespan, of a set of jobs; however, in a grid environment users may have various goals in addition to minimizing the makespan, more specifically, they may require a specific operating system, they may have data and memory requirements, they may pay attention to deadline more than minimizing the makespan and so forth [21]. Consequently, a system framework, *i.e.* a complicated software architecture, is needed to handle scheduling in grid systems since local schedulers are far away to realize all those needs single-handedly.

2.1.1 Issues in Grid Scheduling

Scheduling in grid systems is a complex task due to a variety of factors such as geographical distances, site autonomy, domain size and dynamic characteristics of resources [22]. In this section, we explain all these issues, including their possible solutions.

Grid resources may be connected with relatively slower WANs in which transferring large input and output files can cause network latency. If this is the case for a particular grid environment, intelligent data management schemes such as replicating frequently used data or caching techniques, should be a part of the scheduling framework [23].

Conventional scheduling systems, which refer to the local schedulers in grid environments, have a complete control on the resources that they belong; however, in a grid system placing a central control for those resources is rather difficult since resources are distributed across autonomous domains. Therefore, a grid scheduling system should conserve site autonomy allowing local policies to run as a part of the grid scheduling framework. A decentralized scheduling mechanism is required not only since the resources are dispersed across different domains but also because there exist hundreds of sites, thousands of users and resources in a typical grid environment.

Another problem that grid schedulers should cope with, is the dynamic characteristics of the grid resources [23]. The availabilities and capabilities of resources vary rapidly in a grid environment; as a remedy, grid schedulers should choose *online* strategies instead of *offline* ones which assume constant availability of resources. These strategies are mentioned in section 2.2.

2.1.2 A Generic Grid Scheduling Process

In general, scheduling is performed in four steps in a grid environment: In the fist step, a user submits the jobs and QoS parameters and then grid scheduler performs a resource discovery operation to find the resources that fit to the request. In the second phase, dynamic information (e.g. load on a resource) of those resources are gathered to specify if they are available to execute jobs in that time frame. In the third phase scheduling strategy is applied to choose the best mapping, *i.e.* assigning jobs to the resources in the most appropriate way [24, 25]. And in the final phase, which is optional, execution of the jobs are monitored to make further improvements or to recover from failure.

2.1.3 Grid Scheduling Models

The structure of the scheduler affects the structure and scalability of the resource management system [22]. The current scheduling systems can be categorized as *centralized*, *decentralized*, and *hierarchical* models.

In a centralized model, all jobs are submitted to a central scheduler that is in charge of scheduling them on the appropriate resources. A central scheme is a single point of failure in distributed systems and it yields poor scalability in a grid environment where the resource and user quantity can be extremely high. Moreover, this scheme cannot ensure site autonomy, which is a major requirement for grid computing.

A decentralized model addresses several significant problems such as fault-tolerance, scalability and site autonomy. However, decentralized schedulers have to coordinate with each other using some complex protocols that will affect the system's overall scalability [26].

In a hierarchical model, the schedulers are organized in a hierarchy in which higher level schedulers control larger set of resources and lower level schedulers control small set of resources. This model is a combination of the centralized and decentralized models.

2.2 Conventional Grid Scheduling Strategies

The problem of finding an optimal scheduling in high computing distributed domains has been shown to be NP-complete [27, 28]; therefore, heuristics have been developed to get near optimal scheduling solutions. Existing scheduling heuristics can be grouped into two classes: online (dynamic) and offline mode (static) heuristics [29, 30].

Online mode heuristics assign jobs to the resources as soon as they arrive at the scheduler; however, in the offline mode heuristics jobs are collected into a set that is processed at predetermined intervals for mapping. Also, online mode heuristics map a job to a resource only once; however, offline mode heuristics may remap a job at each evaluation interval until it starts execution [29, 30].

Offline mode heuristics theoretically make better decisions than online mode heuristics since they have the chance to deal with a group of jobs rather than a single one [29]; however, this scheme has some drawbacks. As it is mentioned in section 2.1.1, availabilities and capabilities of the resources vary rapidly in a grid environment. Thus, the dynamic information about resources that an offline heuristic considers can be quickly out of date. Moreover, resources may go offline, and in the next interval the heuristic has to reprocess jobs that have been already mapped to the previously online resources. It is apparent that this condition will dissatisfy the job owners who pay attention to makespan minimization. Actually, online and offline heuristics should be tested before deciding to place into a scheduling framework, since grid domains may present differences and may have different characteristics.

Both online and offline mode heuristics assume that execution time of the jobs are known a priori. Methods for estimating execution times based on task profiling and analytical benchmarking are argued in [31].

Various heuristics in the literature of distributed system scheduling have been proposed to be used in the computational grids, yet we present the important ones briefly:

Online Mode Heuristics

- **FCFS:** *First Come First Served* strategy arbitrarily assigns jobs to the resources according to the arrival order.

- **OLB:** *Opportunistic Load Balancing* [32, 33, 34] is a simple strategy that assigns each job in arbitrary order to the next available resource without regarding the expected execution time on that resource. The intuition behind OLB is to keep all resources as busy as possible.

- **MET:** *Minimum Execution Time* [32, 34], assigns each job in arbitrary order to the resource that gives the best expected execution time for that job, regardless of the current load on that resource.

- **MCT:** *Minimum Completion Time* [32, 34] assigns each job to the resource that offers the minimum completion time considering its next available time. The idea behind MCT is simply to join the benefits of OLB and MET.

Offline Mode Heuristics

- **Min-min:** *Min-min* heuristic has two phases: In the first step, the minimum completion times are calculated for all jobs, next, from all jobs, the job with the minimum overall completion time is selected and assigned to the corresponding resource. This process repeats until all the jobs are assigned to a resource.

- **Max-min:** *Max-min* heuristic applies the first step as Min-min; however, it assigns the job with the maximum completion time to the corresponding resource in the second phase. The Max-min heuristic is beneficial in a condition where completion time for the jobs varies significantly.

- **Sufferage:** The idea behind *Sufferage* heuristic is that a resource is allocated to a job that would 'suffer' most in terms of expected completion time if that particular resource is not allocated to it. The sufferage value of a job is defined as the difference between its second best and its best estimated completion times. After the sufferage values of the jobs have been calculated, they are assigned correspondingly to the resources that offer the best completion times.

In reference [32] a set of 11 heuristics, OLB, MET, MCT, Min-min, Max-min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu and A*, has been implemented and compared by simulation studies for independent job scheduling, *i.e.* jobs with no inter-task data dependencies, in the heterogeneous distributed computing systems. In this study, the performance of the heuristics is based on minimizing the execution of the metatask. As the outcome of the simulations, Min-min heuristic outperforms the other techniques.

Maheswaran et al. [30] also compare heuristics in their study and state that the selection of the heuristics to use in distributed system scheduling, depends on parameters such as heterogeneity among jobs and resources, and the arrival rate of the jobs.

In addition to comparison of heuristics, some other works aim to develop new heuristics: In reference [35], Sufferage heuristic is improved as *XSufferage* to be capable of scheduling parameter sweep applications with file and I/O requirements. In [36], *k-windows* scheduling heuristic is proposed which schedules independent tasks on the clusters regarding the parameters such as available CPU load, free memory and the number of remaining jobs. In [29], the QoS guided Min-min heuristic is presented in which the conventional Min-min heuristic is enhanced in order to take QoS criteria of grid users into account.

Beside heuristics, mathematical models, such as divisible load theory [37], have also been applied to solve scheduling problems in grid computing. In references [38, 39, 40], divisible load theory is proposed to solve problem of allocating and scheduling computing resources on a grid environment for excessive number of independent tasks from large number of users. Divisible load scheduling can be applied to the optimization of distributed computing problems where both communication and computation load can be divided arbitrarily among a number of processors and links [41]. The aim of divisible load scheduling, which uses a linear mathematical model, is to reduce the total execution time of the jobs on the computational resources. However, divisible load theory based studies have a restriction of arbitrary-sized divisible load assumption, which is not necessarily the case in the real world. In reality, jobs and data are usually measured as integer number of units, and the integer programming problem is NP-complete [21]. Therefore, the theoretical optimization that divisible load theory presents may loose its validity in a real world problem.

In most of the conventional grid scheduling studies, especially in simulation based ones some common assumptions can be noticed that force us to discuss them in the context of grid concept. First, there are only a few studies that consider scheduling dependent jobs [42], typically, jobs are assumed to be independent that no inter-task dependencies exist; however, a grid domain should support both type of jobs. Second, in general, a few QoS criteria are considered; in most cases reducing makespan is the only goal. And last, no site autonomy is allowed, that is local sites could not run their own scheduling policies. Besides, there are also two well organized studies; [23] and [43] that consider most of the aspects that we mentioned above. They allow site autonomy by dividing scheduling into two phases; external and internal scheduling. External schedulers use site selection strategies to dispatch jobs and internal schedulers organize those jobs over local resources using heuristics. Moreover, [23] allows priority to local policies and considers dynamic data replication as part of the scheduling problem, and [43] considers dependent tasks and allow users to specify deadline for their jobs.

In addition to the theoretical and simulation works, a number of application level scheduling systems have been developed for grid computing [4, 5, 44, 7, 45], and they are explained and compared in several studies [9, 22, 26]. Among them, we explain the AppLeS [44] and Condor/Condor-G [7, 46] systems briefly.

The Application Level Scheduling System (AppLeS) provides a methodology, application software, and software settings for adaptively scheduling and deploying applications in heterogeneous, multi user grid domains. AppLeS employs decentralized scheduling and uses services of Network Weather Service (NWS) [47] to observe changes in performance of resources dynamically [2]. In AppLeS, applications are embedded into agents that perform resource scheduling based on static and dynamic resource information provided by NWS. Also, these agents can use the services of some core grid middlewares such as Legion [48], Globus [3] and NetSolve [49].

Condor is a resource management system primarily designed for utilizing idle resources on a network. Condor employs a centralized scheduling scheme in which a user submits jobs to a central scheduler which is responsible for finding the appropriate resources for the execution of the jobs [50]. The scheduler can transfer a running job among machines until it is completed. Each machine in the Condor system advertises its resources and reports their availabilities to the central scheduler. Condor-G, the complementary tool for Condor, merges the inter-site resource management protocols of the Globus and intra-site resource and job managing schemes of Condor to let users exploit multi-site resources as though they all exist in a single site.

2.3 Market-based Grid Scheduling Strategies

2.3.1 The Economic Problem

The economic problem exists only when the resources are scarce and the participants, namely consumers and producers, maximize their utility by deciding among needs that cannot be concurrently satisfied and lead to different utility levels [51]. If resources are reachable, whenever required, and their utilizations result the same in value, there would be no problem of allocating the resources for utility maximization, or in other words there would be no economic problem.

Considering the statement above, we can assert that economic problem exists in the computational grids. Although the amount of resources are not few, resources appear scarce due to heavy loads of the users, and allocation of the resources leads to different utility levels in the users' point of view since grid resources are heterogeneous. Therefore, scheduling in grid computing is very much related to the economics that is the study of how limited resources, goods, and services are allocated among competing uses [52].

2.3.2 Economic Models for Grid Scheduling

In the real-world markets, various economic models are used for setting the price of a commodity or a service, based on the supply-demand and their value to the users [11]. Some of these models are convenient to design a real grid economy system or solving the resource allocation problems.

Commodity Market Model

In a commodity market model, resource owners competitively set their service price and charge consumers according to the amount of resource they consume [11]. The pricing policy of the commodity market model can be flat or variable depending on the supply and demand on the resources. In the flat price policy, price of a resource is fixed for a certain period and it remains the same regardless of the demand or service quality, whereas in a variable pricing policy, prices change frequently according to the supply and demand variations. As one of the main principles of economic theory, the increase of demand or the decrease of supply, increases the prices, and the decrease of demand or the increase of supply, lowers the prices until there exists an equilibrium between supply and demand.

Bargaining Model

In the bargaining model, consumers bargain with resource owners for lower service prices and longer usage periods [11]. Both consumers and resource owners have their own utilization functions and they bargain with each other until they agree on a price or one of the sides is not willing to bargain any further. As a drawback, if the consumers start negotiation offering relatively low prices then it may take considerable amount of time to reach an agreement causing waste of time.

Tender/Contract-net Model

Tender/Contract-net model is based on the contracting mechanism used by businesses to manage the exchange of goods and services [11]. The trading protocol of this model is simply as follows: Consumers announce their requirements such as memory, architecture and deadline whenever they require to commerce. In the second step, the interested resource owners evaluate the announcements and reply with their bids. And finally, the announcement owner evaluates bids and start trading with the resource owner of the most appropriate bid. As a disadvantage, a less capable resource may be selected if a more capable one is busy at the announcement time and moreover, consumers are not obliged to inform the participating contractors that an agreement has already been made.

Auction Model

Auction by definition is the process of selling a property to the highest bidder [11]. This model provides a one-to-many negotiation scheme between resource providers and consumers. The typical steps involved in an auction process are as follows: A resource owner announces its services and invites bids in the first step. Then, in the second step, consumers offer their bids. This step continues until no consumer is willing to bid a higher price or the resource owner ends if the minimum price value is not met. Auctions can be carried out as open or closed depending on whether the consumers can see each other's offers or not. Additionally, consumers may raise the offered bid and auctioneers may update the offered sale price.

Community Model

In a community model, market players share each other's resources to form a cooperative computing environment [11]. This model is suitable when those players have to be both producers and consumers. The trading can be realized by credits that are given proportionally in response to the amount shared by the resource owner.

Monopoly/Oligopoly

Monopoly is a market in which there is only one seller of a product or service [11]. Consumers cannot affect the prices of the services and they have to buy the service at the given price. This may be the case in a grid environment if there is a single owner of a particular service. In addition, market situation can be an oligopoly if a small number of producers control the market and set the prices.

2.3.3 Market-based Scheduling Studies in Conventional Distributed Systems

Adapting market-based approaches to scheduling problems in distributed computing systems is not a new idea; a number of systems have been developed before grid systems have evolved. All those early studies make use of economics to obtain effective resource allocation solutions. Some of the important systems are explained in this section.

The *Enterprise* task scheduler [53] utilizes the concepts of contract-net model to exploit idle resources in the distributed network. The scheduling procedure in the Enterprise system simply proceeds as follows: A client publicizes a task indicating relative priorities, description of required resources and information for machines to estimate the execution time. Subsequently, idle machines offer bids to the requests specifying the estimated execution time of the tasks. Finally, the machine that provides the shortest execution time is chosen by the client in order to process the task.

Actually, Enterprise system does not utilize a real market approach; it uses artificial priorities instead of a price mechanism which, in fact, can eliminate such a need [51]. Hence, clients are unable to make decisions between fast and slow resources to submit tasks. Moreover, machines have no incentive to compete with each other in order to obtain a job by giving the best offer, since any utility maximizing behavior has not been defined for the machines.

The *Walras* [54] system is an application of *general equilibrium theory* to management of distributed systems. The *tatonnement* process is implemented to attain the equilibrium price for the resources. In the tatonnement process, there is a central auctioneer that receives utility functions of the consumers and calculates the equilibrium prices using the Smale's theorem [55]. Besides, for each resource it initiates a distinct auction, and after the calculation, the resulting prices are sent to all consumers [20]. Consumers can respond to the new prices by adjusting their utility functions. This process is repeated until the prices reach to a steady state.

Spawn [17] is a scheduling system designed as a market economy that is composed of interactive consumers (buyers of CPU time) and resource owners, to run in a distributed computing environment. As the economic model; Vickrey auction is used to handle the sales. In addition to independent tasks, Spawn also supports concurrent tasks which are assumed to be tree-based. Each node in the tree which refers to a subtask, associated with a manager and has a funding rate shared by the root node. The managers of the nodes participate in an auction independently from each other. After the trading finishes, they inform the root manager and deliver the results.

2.3.4 Market-based Scheduling Studies in Grid Systems

Assuming that the scheduling problem must be decentralized in a grid system, market approaches can offer several advantages [56]. First, markets are naturally decentralized, and self-interested agents may yield global optimum solutions by focusing on only their own good. Second, communication is limited to the exchange of bids and prices between agents, and a pricing mechanism has the potential to differentiate resources more dynamically compared to a priority mechanism [51]. Considering these advantages, several market-like systems (not real economies) have been proposed to attain optimal resource allocation solutions in computational grids as in the conventional distributed systems.

In [20], a contract-net based model is proposed for scheduling parallel jobs in grid environments. In this work, simulations have been carried out to compare the proposed market-based method with the conventional FCFS-Backfilling heuristic. The results revealed that their method outperforms the conventional one in terms of average-weighted-response-time and average-weighted-wait-time metrics. Besides, in [57] and [58] the continuous double auction protocol is used for the same purpose. In [57], the continuous double auction outperforms the conventional FCFS and Minimum Completion Time (MCT) heuristics in terms of job completion ratios and in [58] the protocol outperforms the conventional Round Robin heuristic in terms of weighted-completion-time metric. In addition to these studies that compare market-like approaches with the conventional ones, in [59] the tatonnement process is compared with the Vickrey auction in terms of grid wide price stability, market equilibrium, application efficiency and resource efficiency. Simulation results point that the tatonnement process is more successful than the auction model in controlling grid resources in terms of these criteria.

Market mechanisms can also be considered to build real computational grid economies to prompt individuals or companies share their resources and let the users benefit from the extensive resources in return for a fee.

There would be, naturally, two main players in a real computational grid market:

Providers (resource owners) and consumers (resource users) [2]. Resource owners would like to maximize their utilities, *i.e.* make profits or recover their costs, by charging the users for accessing or using their resources, and users would like to solve their problems by using the resources that satisfy their utilities *i.e.* budget, deadline and other QoS criteria. Moreover, resource owners would like to offer a competitive service access price in order to attract users, and users would have the right to select the resources that best satisfy their needs. Thus, in the computational grid economies, a user would be in competition with other users and a resource owner with other resource owners as in all other real markets.

User applications may have various resource demands depending on computations performed and methods used in solving the problems [2]. Some applications can be CPU intensive while others can be data intensive or a mixture of two. Therefore, in economy grids, commodities can be a variety of resources such as CPU slots, memory, storage, software and network usage. To charge the users for using these resources accounting services and payment mechanisms have to be deployed into a computational grid economy. References [60, 61, 62] address this subject proposing accounting infrastructures and automatic payment mechanisms for computational grids.

Unfortunately, market-based real grid economy systems have not been commercialized yet; however, there are ongoing projects at the academic level. Among them we present Nimrod/G and CPM systems below.

Nimrod/G [6] is a market-based grid resource management and scheduling system, designed to be established in the real grid environments. It employs the Globus toolkit's services for dynamic resource discovery and shipping jobs over the grid, and employs Grid Architecture for Computational Economy (GRACE) framework [60] for the trading services.

There are five major components in the Nimrod/G architecture (Fig. 2.1). The *Client* or *User System* component operates as a user interface to control and manage jobs; users can specify time and cost constraints to affect the behavior of the scheduler in resource selection phase, and they can monitor the status of the jobs. *Parametric Engine* is the central component that manages the entire application,*i.e.* set of jobs. Parameterization of an application, creation of jobs, maintenance of job status, and interacting with clients, schedule advisor, and dispatcher are the main responsibilities of this component. Moreover, it records the states of the entire experiment to restart it in case of failure. The *Scheduler* component is responsible for resource discovery, resource selection and job assignment. The resource heuristics [2, 63] employed in the scheduler interact with a grid-information service directory, identify the list of available and appropriate machines, and keep track of resource status information. In addition, heuristics are responsible for selecting resources that meet the deadline and cost constraints. The *Dispatcher* component starts the *Job Wrapper* component that will initiate the execution of a job on the selected resource.

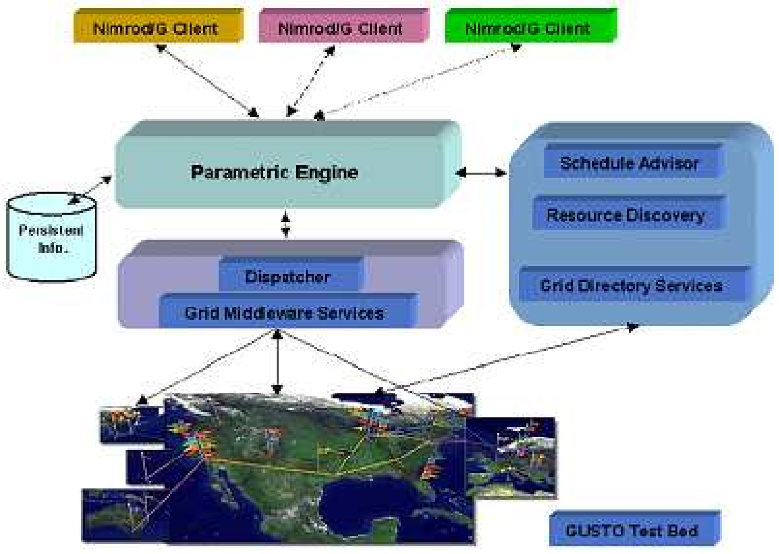

Figure 2.1: Nimrod/G Architecture [6]

Compute Power Market (CPM) [64] is another market-based resource management and job scheduling system designed for grid computing. It transforms the distributed computing environment into a computational market in which users can solve problems by renting computational power.

The CPM basically composed of markets, resource consumers, resource providers and their interactions. It provides various economic models such as commodity market model, contract-net/tender, and auction for resource pricing and scheduling.

The architectural components of the CPM are illustrated in Fig. 2.2. Market is the point of interaction for both consumers and producers. Consumers and producers have to register with a market to rent or sell computational power. A market basically provides

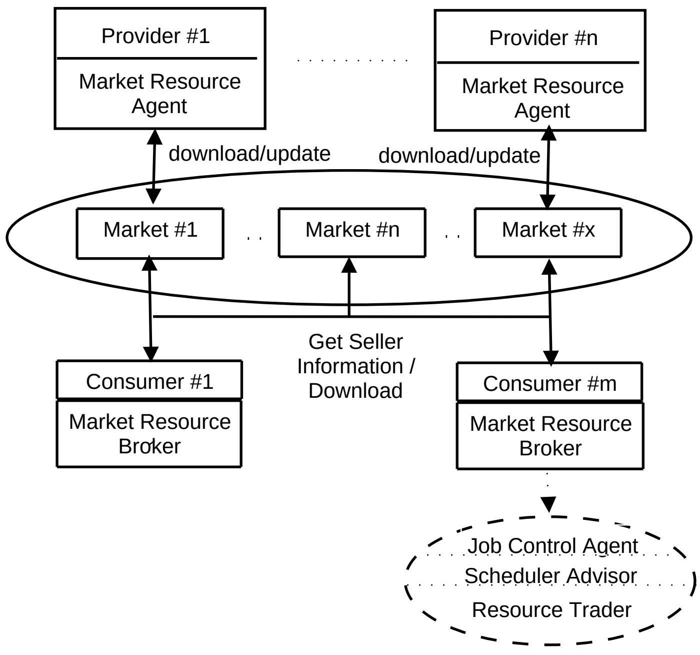

Figure 2.2: The Components of CPM Architecture [64]

repository of information on providers, agents for consumers and providers, mechanisms for updating the information and interaction with other markets.

Resource providers get a Market Resource Agent (MRA) from the market that is responsible for updating the market with the most recent information about their resources and accepting, deploying and launching the job. MRA works on a push-pull mechanism with the pull unit extracting information such as available memory and number of processes, whereas the push unit sends this data into the market through communication unit. Correspondingly, resource consumers get a Market Resource Broker (MRB) that is responsible for locating the proper provider based on the information supplied by the market. MRB comprises several sub-components such as Job Control Agent, Market Explorer, Resource Trader, and Scheduler.

Job Control agent is responsible for guiding a job through the system by interacting with other components of the broker. Economy computations in the CPM grid is managed by schedule advisor, trade manger and trade server. The schedule advisor uses market explorer for resource discovery, trade server for negotiating costs and scheduling algorithms for deciding on mappings between jobs and resources.

Chapter 3

A JOB SCHEDULING HEURISTIC FOR COMPUTATIONAL ECONOMY GRIDS

In a computational grid economy system, we need effective schedulers not only to minimize the makespan but also to minimize the costs that are spent for the execution of the jobs. To this end, we present the Stage Focused Time-Cost Optimization (SFTCO) heuristic to be used in a commodity market model in which the prices are determined according to the supply and demand in the market.

3.1 Characteristics

The SFTCO [65] heuristic is designed to complete all the jobs of a user using one or more optimization strategies (cost, time or time-cost) according to the deadline and budget constraints of the user. It is primarily intended for scheduling workflows [12] each represented as a sequence of activities or stages as illustrated in Fig. 3.1. Several computation-demanding applications in the scientific field conform to a sequential workflow pattern such as Bulk Synchronous Parallel (BSP) programs [66, 67], in which a computation has a number of supersteps, each including parallel computational processes that synchronizes at the end of the superstep; or scientific applications [68, 69], in which the output of an activity returned from a grid service becomes the input for the subsequent activity. Furthermore, treating the whole job set as a single stage workflow application, the proposed heuristic can also deal with parameter sweep type of applications that are combination of task and data parallel models which contain large number of independent jobs operating on different data sets [13]. Parameter sweep models can be applied to the methods of drug design, genome sequence analysis, or protein folding in order to observe the outputs of different scenarios and parameters.

In computational grid environments, to improve utilization and throughput of the resources such as clusters or supercomputers, and to reduce the communication overhead among the grid entities, jobs should be handled at the local and grid level separately [23]. As Fig. 3.2 demonstrates, at the grid level, jobs should be mapped to the resources that meet QoS constraints, and at the resource level, local policies should be employed in order to map the received jobs to the processing elements. Correspondingly, the SFTCO is a grid level scheduling heuristic that permits local policies to be employed.

The availabilities and capabilities of the resources vary frequently in grid environments; thus, schedulers should be capable of dealing with the dynamic nature of these resources. The SFTCO is a dynamic scheduling heuristic such that mapping decisions are made during the execution of a schedule, considering the instant load of the resources along with the cost and processing speed criteria.

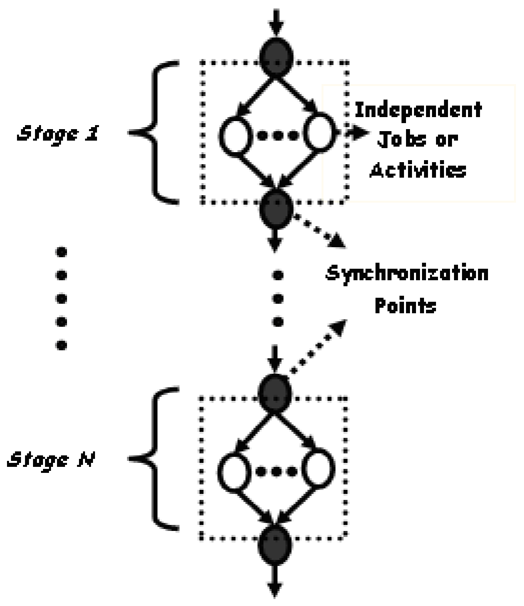

Figure 3.1: A sequential workflow schema

3.2 Assumptions

The assumptions of the SFTCO heuristic are as follows:

- A grid information service is available that can be queried for the availabilities and capabilities of the grid resources.

- A resource, which may be considered as a cluster or supercomputer, composed of identical processing elements (*i.e.* parallel processors).

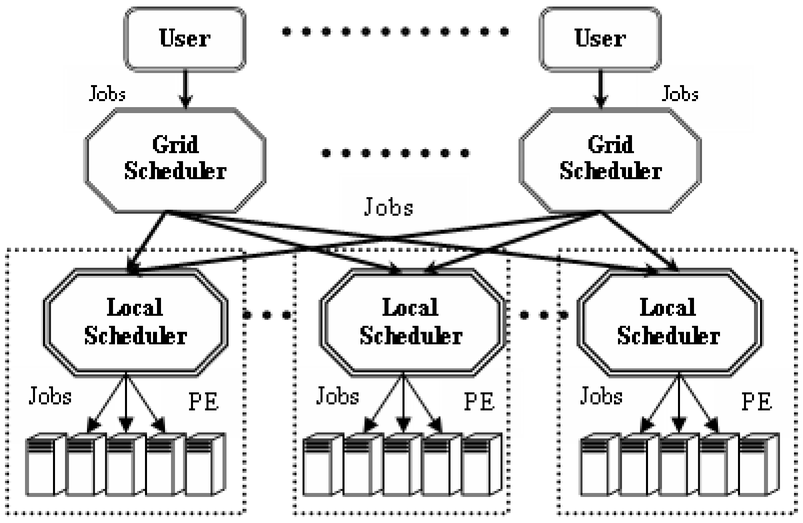

Figure 3.2: Interactions among grid components

- Resources are capable of presenting their dynamic state such as load.
- Speed of a processing element is defined in Million Instructions per Second (MIPS) rating (in accordance with SPEC benchmark [70]).
- Size of a job is known a priori and represented in Million Instructions (MI) rating. Most research on distributed system scheduling has dealt with the problem when the jobs and interprocessor communication costs are entirely known [71].
- Jobs are independent; they do not communicate within a stage. Parameter sweep and BSP applications compose of independent jobs.

3.3 Methods

The SFTCO heuristic, presented in Algorithm 1 (refer to Table 3.1 for the notation), takes three input parameters: An application, represented as a workflow, a budget that the user is willing to spend for the application, and a duration in which the results of the application must be returned.

Initially, the currently running stage is assigned portions from the total budget and total duration (*i.e.* the difference between the beginning time of the scheduling and the deadline) in proportion to its total job size, and the surpluses or debts that are emerged from the previous stage are added (line 4-5). The motivation behind distributing the budget and duration among the stages is to optimize the cost and time. Since the demand on the resources can vary frequently in a grid environment, we may need to change our optimization strategy and adapt to the new conditions while proceeding over stages. Then, the maximum expense that the stage may consume is estimated via multiplying the total job size of the stage by the Price per Million Instructions (PPMI) value of the most expensive resource (line 6), and the maximum duration that may elapse to process all the jobs in the stage is estimated via dividing the total job size by the MIPS rating of the slowest resource (line 7). These estimations indicate how much time and fund is required to process all the jobs in the worst case. Next, the jobs of the stage are sorted in the decreasing order of job size (line 8), and one of the three optimization strategies is selected after comparing the alloted budget and duration with the estimated ones (line 9-15). If the actual budget is sufficient but the actual duration is insufficient in terms of these comparisons, time optimization strategy is used in which faster resources are given precedence over slower ones. On the other hand, in case the actual duration is sufficient but the actual budget is insufficient, cost optimization strategy is used in which cheaper resources are given precedence over expensive ones. Finally, in case both of them or none of them are sufficient in terms of the comparisons, time-cost optimization strategy is used in which relatively faster and cheaper resources are given precedence. This relativity is determined by the MIPS/PPMI ratio of the resources.

After sorting the resources according to an optimization strategy, for each resource in order, jobs are submitted until the corresponding resource reaches to the desired load factor, DL_j (line 16-22). We define the load factor as the ratio of the total number of jobs that a resource is currently executing, to the total number of processing elements it has. DL_j, is calculated as it is illustrated in Eq. 3.1. $Duration_i$ is the duration alloted to the i^{th} stage, $MIPS_j$ is the MIPS rating of the interacted resource, and $maxMI_i$ is the size of the largest job in that stage. This method is used to execute as many jobs as possible on the desired resource, without exceeding the duration assigned to the stage; however, with the assumption of no other user would pull up the load any time further. It is of course not a guaranteed way but a reasonable approach.

$$DL_j = \frac{Duration_i * MIPS_j}{maxMI_i} \qquad 1 \le i \le n,\ 1 \le j \le k. \tag{3.1}$$

If the interacted resource has a space-shared management architecture (in which a

Table 3.1: Notation for the SFTCO Heuristic

n :	Number of stages in the workflow
k :	Number of resources
TrB :	Transfer budget
TrD :	Transfer duration
$Budget$:	Total budget
$Budget_i$:	Budget alloted to the i^{th} stage
$Duration$:	Total duration
$Duration_i$:	Duration alloted to the i^{th} stage
S_T :	Total job size of the workflow
S_i :	Total job size of the i^{th} stage
$maxEE_i$:	Max. estimated fund required for the i^{th} stage
$maxED_i$:	Max. estimated duration required for the i^{th} stage
$\overline{PPMI}$:	Price Per MI value of the most expensive resource
$\underline{MIPS}$:	MIPS rating of the slowest resource
DL_j :	Desired load factor for $resource_j$
CL_j :	Current load factor of $resource_j$
$workflow_i$:	Job list of the i^{th} stage
$resources$:	List of available resources
$resource_j$:	The j^{th} resource in the $resources$ list
$RealExpense_i$:	The amount spent in the the i^{th} stage
$DurationElapsed_i$:	Duration elapsed in the i^{th} stage

processing element can run only a single job contrary to the time-shared architecture), the DL_j is simply set to 1 (neglected in the pseudo code). After that, if there remains any unassigned jobs, (*i.e.* all the resources are overloaded), a single job is assigned to a single resource in order until all the jobs are dispatched (line 23-27).

After completing the job mapping, all the outputs have to be waited for (line 28), since the next stage in the workflow can be initiated after the completion of the current one. On receiving all the outputs, the real expense and duration spent in the stage are computed (line 29), and subsequently, their differences between the alloted ones are calculated (line 30-31), which are going to be reflected to the next stage's budget and duration as surpluses or debts. The heuristic runs until all the stages are completed or an interruption occurs due to a budget or deadline violation. We assume that, the remaining budget and duration is checked independently, (e.g. via a thread) from the implementation point of view.

Algorithm 1 The SFTCO Heuristic

```
1:
2: $TrB, TrD \leftarrow 0$
3: for $i : 1$ to $n$ do
4:     $Budget_i \leftarrow ((Budget/S_T) * S_i) + TrB$
5:     $Duration_i \leftarrow ((Budget/S_T) * S_i) + TrD$
6:     $maxEE_i \leftarrow S_i * \overline{PPMI}$
7:     $maxED_i \leftarrow S_i / \underline{MIPS}$
8:     Sort $workflow_i$ in the decreasing order of job size.
9:     if $Budget_i < maxEE_i$ and $Duration_i > maxED_i$ then
10:        Sort the $resources$ in the increasing order of PPMI value.
11:    else if $Budget_i > maxEE_i$ and $Duration_i < maxED_i$ then
12:        Sort the $resources$ in the decreasing order of MIPS value.
13:    else
14:        Sort the $resources$ in the decreasing order of (MIPS / PPMI) value.
15:    end if
16:    for j:1 to $k$ do
17:        Calculate $DL_j$, Query $CL_j$
18:        while $CL_j < DL_j$ do
19:            Assign a single job to $resource_j$ from $workflow_i$ in order
20:            Query $CL_j$
21:        end while
22:    end for
23:    while there remains unassigned jobs in $workflow_i$ do
24:        for j:1 to $k$ do
25:            Assign a single job, to $resource_j$ from $workflow_i$ in order
26:        end for
27:    end while
28:    Wait until all the submitted jobs are received
29:    Calculate $RealExpense_i$ and $DuarationElapsed_i$
30:    $TrB \leftarrow Budget_i - RealExpense_i$
31:    $TrD \leftarrow Duration_i - DuarationElapsed_i$
32: end for
```

3.4 Simulation Architecture

We have used the GridSim toolkit [14] (see Appendix A for more information about the toolkit) to simulate a grid economy system in which we can explore the capabilities of the SFTCO heuristic for sequential workflows and parameter sweep type of applications.

The GridSim entities that we have used or regenerated are mainly, User, Broker, Grid Resource, and Grid Information Service (GIS) entities. The interactions between these entities are illustrated in Fig. 3.3.

A Grid Resource entity is a union of machines each having one or more processing elements (PE). A Resource can be designed either as a time-shared or as a space-shared system. In time-shared Resources, submitted jobs are scheduled using the Round-Robin algorithm; however, FCFS heuristic is used in space-shared Resources. Additionally, each Resource is assigned a cost value that it charges Users per unit time while processing a job. After a simulation gets started, all the Resource entities register themselves to the GIS entity (Fig. 3.3: *1*).

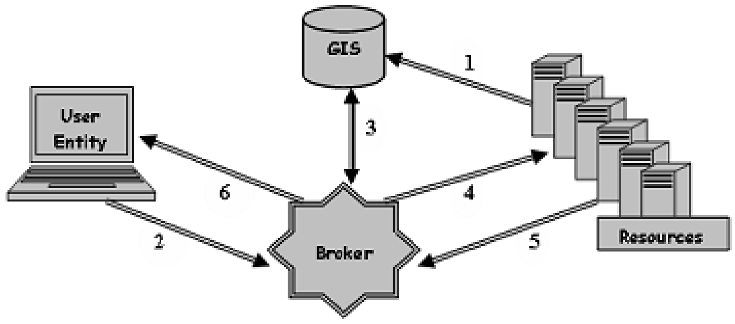

Figure 3.3: Interactions between the simulation entities

A User entity simulates a user who wants his workflow or parameter sweep application to be executed in a grid computing environment within the budget and duration limits. Users pack the application and the requirements and then pass them to the Broker entity (Fig. 3.3: *2*) that will manage the execution of the application on behalf of the User. In GridSim, a job and all the related information such as its size, defined as million instructions (MI), I/O operations and the size of input and output files are packed into a Gridlet object. Thus, an application should be treated as a list of Gridlets. A parameter sweep application is represented as a GridletList object; yet, a workflow is represented as an array of GridletLists that each of its member represents a distinct stage.

In our simulation architecture, each User has a Broker entity that is commissioned to accomplish an application on behalf of its user. On receiving an application along with the requirements, Broker queries GIS entity for resource discovery, and subsequently GIS returns a list of available resources (Fig. 3.3: *3*). Afterwards, Broker starts to perform the scheduling heuristic for allocating appropriate Resources to Gridlets (Fig. 3.3: *4*). During scheduling, Broker can query the Resources to find out their dynamic information such as cost, capability, availability, load and other configuration parameters. On completion of each Gridlet, Resources return them to the Broker (Fig. 3.3: *5*). The processed Gridlets hold the information of their execution cost and duration. Finally, when the scheduling is either completed successfully or interrupted due to a budget shortage or a deadline violation, all the processed Gridlets are sent back to the User entity (Fig. 3.3: *6*).

3.5 Experimental Results

This section presents the results of simulation studies carried out to evaluate the performance of the SFTCO heuristic. Initially, we compare our heuristic with cost [2, 63], time, and cost-time optimization heuristics (see Appendix B for the pseudo codes) of the Nimrod/G [6] system, to observe its performance on parameter sweep type of applications, and next, to observe the heuristic's performance on sequential workflow type of applications, we compare it with a random approach since there do not exist any economy driven workflow scheduling heuristic in the literature.

We simulated a number of time-shared resources running with round-robin local policy. Table 3.2 presents the attributes of these resources. Prices are assigned rationally considering both their processing power and the number of processing elements they have. Besides, we assume that the resource owners apply fixed pricing policy. In the simulation environment, users are charged per time unit on execution of each job; however, brokers translate the price-per-unit-time value to a PPMI value, to be able to compare the relative costs of the resources.

In all of the experiments, users submit applications within a 100 time unit interval, based on a Poisson arrival process, and we assume a user submits only a single application.

Table 3.2: Attributes of the simulated resources

Resource Name	Number of PEs	MIPS Rating of each PE	Resource Management	Price per PE unit time in G$	PPMI in G$
R1	32	10	Time-Shared	9	0.9
R2	36	12	Time-Shared	13	1.083
R3	48	13	Time-Shared	20	1.5384
R4	24	15	Time-Shared	21	1.4
R5	36	20	Time-Shared	40	2
R6	16	12	Time-Shared	12	1
R7	24	11	Time-Shared	9	0.818
R8	40	10	Time-Shared	9	0.9
R9	24	10	Time-Shared	8	0.8

Table 3.3: Variation intervals for job parameters

Experiment Type	Number of Stages	Number of Jobs within a stage	Size of a Job	Input File Size	Output File Size
Parameter-sweep	1	50-150	1000-10000 MI	250-750 B	2.5-7.5 KB
Wokflow	1-10	10-100	1000-10000 MI	250-750 B	2.5-7.5 KB

We repeat the experiments 10 times for each average arrival rate value, λ, that is varied from 10^{-2} to 25.10^{-2} with a 10^{-2} step size, afterwards, we take the average of the results. Moreover, each user has a direct link to each resource with 100 Mb/s connection speed, and the other parameters such as the number of stages in a workflow and number of Gridlets within a stage has assigned randomly from a uniform distribution; their variation intervals are given in the Table 3.3. We assume that parameter sweep and workflow applications consists of independent coarse-grained jobs. Accordingly, we express Gridlets' size in such a way that they are expected to take 500 time-units on the slowest resource and 250 time-units on the fastest resource, on average. The Gridlet (job) size parameters are chosen appropriately to make the resources vary from low to high loaded conditions as the average arrival rate increases. Besides, it is important to note that exactly the same experiments are performed when comparing the heuristics.

In cost optimization comparison, we keep the budget tight and the deadline relaxed for the users. Each user is assigned a duration that is 10 times higher than the duration required to execute all its jobs on resource R1 in sequential order, and a budget is assigned such that each user can afford to execute all its jobs on resource R9, which is the cheapest resource. In time optimization comparison, this time we keep the budget relaxed and the deadline tight such that durations are assigned to be sufficient to execute all the jobs of a user on resource R1, which is one of the slowest resources, and a budget is assigned such that each user can afford to execute all the jobs on resource R5, which is the most expensive resource. Finally, in cost-time optimization comparison users are assigned tight deadline and budgets in the same way as explained for the cost, and time optimization comparisons.

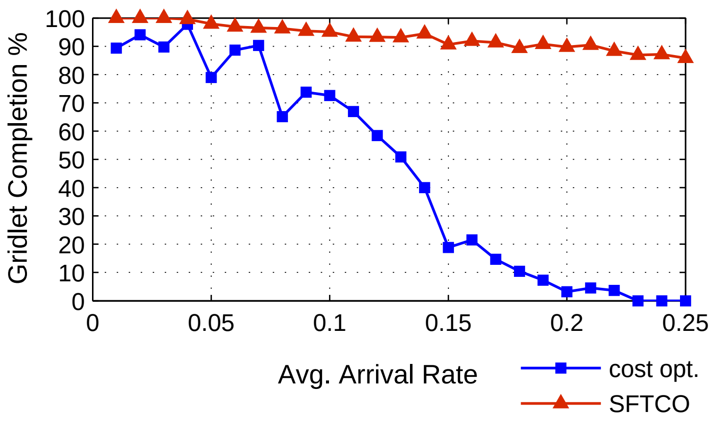

Figure 3.4: Cost optimization comparison

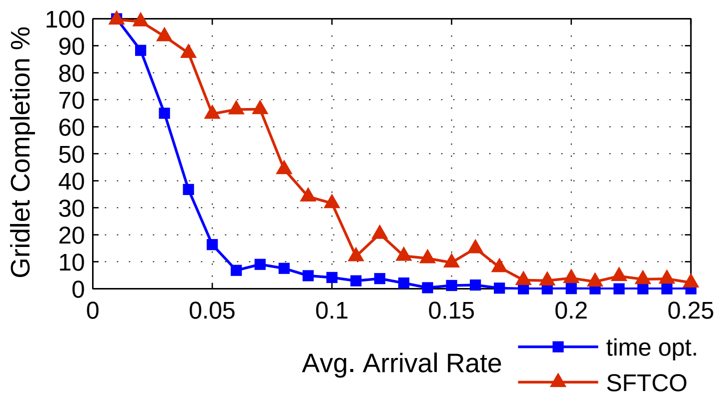

Figure 3.5: Time optimization comparison

Figs. 3.4, 3.5, and 3.6 show the performance results of cost, time, and cost-time optimization comparisons of SFTCO and the heuristics of Nimrod/G, in terms of average

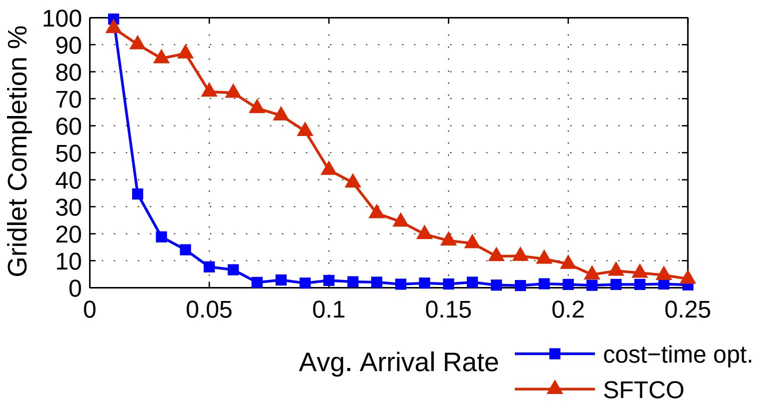

Figure 3.6: Cost-Time optimization comparison

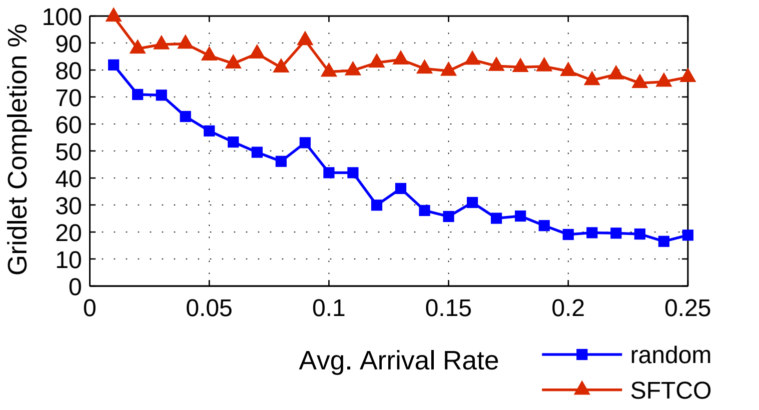

Figure 3.7: Workflow scheduling comparison

Gridlet completion percentages among all users respectively. Average Gridlet completion percentages illustrate how effective the durations and budgets are managed by the heuristics. In fact, *average-response-time* and *average-wait-time* metrics are used in such comparisons; however, using these metrics do not make sense for our experiments since the simulated resources are designed as time-shared (*i.e.* they immediately start to execute the received jobs), and execution of an application can be terminated due to a budget shortage or a deadline violation.

According to the results, it is clear that SFTCO heuristic outperforms the compared heuristics. Actually, resource selection order is same for the heuristics in the cost and time optimization; however, the decisions such as how many Gridlets are going to be submitted to a resource and the way of managing the remaining duration change, hence the performance differences arise. Besides, the cost-time optimization heuristic creates groups containing resources that have the same cost, and then sort the resources within these groups such that faster ones are preferred first. On the other hand, the SFTCO heuristic arranges resources with respect to their MIPS/PPMI ratio.

In Fig. 3.4, the reason of the sharp fall of the cost optimization heuristic, while the arrival rate increases, is interestingly due to deadline violations rather than budget shortages. This heuristic suffers when the demand increases for the resources, since it looses significant amount of time waiting for appropriate conditions; yet those conditions never occurs.

Fig. 3.5, and Fig. 3.6 demonstrates that the time, cost-time, and SFTCO heuristics suffer, as expected, from the increase of demand on the resources. Nevertheless, SFTCO heuristic has completed more Gridlets in the same conditions, which means that it can perform better than the other heuristics in case the resources are extremely loaded.

To explore SFTCO heuristic's performance on sequential workflow type of applications we compared it with a strategy that randomly assigns jobs to the resources. In the experiments, users are randomly assigned budgets and deadlines from a uniform distribution that ranges between tight and relaxed values. Therefore all the optimization strategies may have the chance to be performed. As Fig. 3.7 illustrates SFTCO outperforms the random approach and additionally it does not suffer from the high demand drastically.

Chapter 4

MARKET MODELS FOR A COMPUTATIONAL GRID ECONOMY SYSTEM

In this chapter, we present a commodity market and an auction model that are designed to meet the requirements of the market participants in the economic respect and ensure efficient scheduling in a computational grid economy system. Besides, we performed experiments to evaluate these models in terms of social welfare using the same simulation architecture mentioned in section 3.4.

In the previous chapter, we present the SFTCO heuristic that is designed to be used in a commodity market model. The SFTCO essentially represents the user (consumer) behavior in our commodity market model by making mapping decisions considering the available budget and duration.

In real-world markets producers and consumers desire to maximize their own utility, without considering the global good [72]. However, given sufficient information, independent rational self-interests are believed to achieve optimal resource allocation solutions, both for the individuals and for the society [11]. Correspondingly, in our models self-interested resource owners (producers) desire to profit or recover their costs allocating their resources to the most profitable jobs, and self-interested users (consumers) desire to solve their problems considering their budget and time constraints.

There is no asymmetric information in our market models which means all participants (agents) have the same relevant information, and they are rational in their decisions. Models also motivate participants to reveal their true utility for the commodity; hence the mechanisms are *incentive compatible* [18]. This frees participants from strategic considerations which is particularly significant in a software-based market system in which the strategies are preprogrammed rather than interactively done by people. Besides, decision making is decentralized in the models, since a central mechanism would be a point of failure and a communication bottleneck in such a distributed environment.

We propose these market models for a computational grid system in which resources are personal computers, clusters or supercomputers that comprises identical processing elements (CPUs), and jobs are computation-intensive applications such as workflows or parameter sweep models. In economy grids, commodities can be a variety of resources such as CPU time, memory, storage, software and network usage; however, the only commodity that we consider is the CPU time, and therefore, the prices are determined as per unit time.

4.1 The Commodity Market Model

In this market model each resource owner is free to determine a pricing policy for her own resource, and competitive users take prices as given ignoring any market power they might have, thus eliminating the need to reason tactically about other agents' bidding behavior [73]. Scheduling is done by the invisible hand of economics; or in other words, supply and demand determines where the jobs are going to be executed [74].

The SFTCO heuristic, basically, determines the users' preferences in our commodity market model. The time, cost and time-cost optimization strategies of the heuristic are realistic enough to be the preferences of a computational grid user. Besides, the heuristic takes prices as given and do not perform any strategic behavior against other users' operations.

In the real-world commodity markets, there is no any globally accepted method for price setting or adjustment; each producer or vendor applies an individual pricing policy. Accordingly, we devised a rational dynamic pricing policy (see section 4.1.1) such that resource owners can adjust the prices considering the demand on their resource. Moreover, in [15] a stochastic approximation based pricing policy is proposed to be used in e-commerce applications in order to determine the price of a commodity automatically, depending on past prices and sales. We also consider that this policy can be used by the resource owners in the grid (see section 4.1.2). As another approach, fixed pricing policy may also be preferred by the resource owners. To find out how these policies work in the computational grid environment, we performed comparison experiments, and consequently discuss the results both from users' and resource owners' point of views (see section 4.3.1).

For all pricing policies, we assume that each resource owner individually specifies a minimum price per unit time that she is not willing to execute jobs any price lower than this. The minimum price can be taken as the minimum value required to profit or recover the fixed and variable costs [52]. Besides, setting minimum prices prevents resource owners to undercut their prices to zero as the consequence of a price war [51].

4.1.1 A Dynamic Pricing Policy

In this pricing policy, resource owners can enter the market with a price above than or equal to the minimum price determined in advance, and they adjust the prices over time according to the demand on their resources. The demand on a resource, at a time instant, is assumed to be the utilization of the resource, *i.e.* the fraction of the resource's total capacity that is being used.

Algorithm 2 DynamicPrice($P_{init}, P_{min}, \mu,\ T$)

```
1:
2: Set initial price $p = P_{init}$
3: Set previous utilization rate $ExUr = 0$
4: for $i : 1$ to $\infty$ do
5:    Wait for a period of $T$
6:    $Ur = getCurrentUtilization()$
7:    if $Ur > \mu$ and $Ur > ExUr$ then
8:       $\Delta p = Ur - \mu$
9:       Update the current price $p = p * (1 + \Delta p)$
10:   else if $Ur < \mu$ then
11:      $\Delta p = \mu - Ur$
12:      Update the current price $p = p * (1 - \Delta p)$
13:      if $p < P_{min}$ then
14:         $p = P_{min}$
15:      end if
16:   end if
17:   $ExUr = Ur$
18: end for
```

By Algorithm 2 each resource owner calculates a new price-per-unit-time; p. In the algorithm, Ur $(0 < Ur < 1)$ denotes the utilization and Δp is the price increment or decrement percentage. We assume that resource owners interpret a utilization below than a ratio that they consider (*i.e.* μ, $0 < \mu < 1$) as a decrease and a higher ratio as an increase on the demand.

If the utilization is over μ, owners increase the price only when the utilization changes (line 7-9), *i.e.* when new jobs are accepted or some running jobs are completed to prevent continuous price increase; however, they decrease the price at each predetermined time interval if the utilization is under μ (line 10-12), in order to attract the users. Naturally, price of a resource do not fall below the minimum price that the owner has specified (line 13-14). Moreover, once a job is accepted, it is charged with the agreed price. Further price changes do not affect the running or the queued jobs.

Generally speaking, when there is less demand for resources, the price is lowered and when there is high demand, the price is raised in this policy, which is expected to help in regulating the supply and demand for access to the resources.

4.1.2 A Pricing Policy based on Stochastic Approximation

This policy, presented in Algorithm 3, is based on a stochastic approximation method, which is a general methodology for online function optimization that estimates the maximum of a regression function by testing and obtaining the function values at points of its choice, and gradually converge to the optimum point [15, 75].

Given a reasonable initial price, the policy simply tries to converge to a local optimal price, p, that is expected to maximize the revenues [15]. Using a step size, Δ, determined as a decreasing function of the number of trials so far, I, such as $I^{-1/3}$ (line 5), the policy conducts sales at both prices $p + \Delta$ (line 6) and $p - \Delta$ (line 8) for a certain period of time, T, and based on the number of received jobs during these periods (in its original form this is the number of commodities that has been sold), $S(p + \Delta)$ (line 7) and $S(p - \Delta)$ (line 9), it calculates the profits (revenues) obtained for the respective prices (line 11-12). Then it updates the current price as shown in the line 14. Here, A is the update interval that is set as a decreasing function of the trial number such as $1/I$. The given equations for A and Δ satisfies Eq. 4.1 and Eq. 4.2 that are the necessary conditions to ensure convergence of the obtained prices to the (local) optimum. Finally, the resulting price is prevented to be lower than the minimum price, P_{min} (line 15-16), and further price changes do not affect the running or the queued jobs, in this policy as well.

$$\sum_{I=1}^{\infty} A(I) = \infty \tag{4.1}$$

$$\sum_{I=1}^{\infty} \frac{A(I)^2}{\Delta(I)^2} < \infty \tag{4.2}$$

Algorithm 3 StochPrice($P_{init}, P_{min},\ T$)

```
1:
2: Set initial price $p = P_{init}$
3: Set trial number $I = 1$
4: for $I : 1$ to $\infty$ do
5:     Set step size $\Delta = I^{-1/3}$
6:     For a period of $T$, set the price to $p + \Delta$
7:     Let $S(p + \Delta)$ be the amount of jobs received during this time
8:     For a period of $T$, set the price to $p - \Delta$
9:     Let $S(p - \Delta)$ be the amount of jobs received during this time
10:    Calculate the obtained profit as follows:
11:    $P(p + \Delta) = S(p + \Delta) * (p + \Delta)$
12:    $P(p - \Delta) = S(p - \Delta) * (p - \Delta)$
13:    Set the update interval $A = 1/I$
14:    Update the current price $p = p + \frac{A}{\Delta} * \frac{P(p+\Delta)-P(p-\Delta)}{2T}$
15:    if $p < P_{min}$ then
16:        $p = P_{min}$
17:    end if
18: end for
```

4.2 Auction Models

An auction is a market tradition with an explicit set of rules determining resource allocation and prices based on the bids of market participants [56]. Auctions can be classified in different ways: In *one-sided* auctions an auctioneer accepts bids from multiple buyers (bidders) on behalf of a single seller, on the other hand, in *two-sided* auctions an auctioneer accept bids from multiple sellers and buyers to sell commodities of the same type. In *open* auctions all bids can be seen by the buyers, whereas, in *sealed-bid* auctions bids are kept secret. Moreover, in *first-price* auctions the winner pays the amount she offers, yet, in *second-price* auctions the winner pays the amount of the second-highest bid.

There are five major auction types in the economics literature [76]: English, First-price sealed-bid, Vickrey, Dutch, and Double. In an *English* (first-price open cry) auction the auctioneer begins with the lowest acceptable price and proceeds to ask for successively higher bids from the buyers until no one increases the bid. At the end, the commodity is sold to the highest bidder. In a *First-price sealed-bid* auction each bidder submits only one bid without knowing the others' bids. The highest bidder wins and pays the amount she bid. The *Vickrey* (second-price sealed-bid) auction differs from the First-price sealed-bid auction in the way that the winner pays the amount of the second-highest bid. In a *Dutch* auction the bidding starts with an extremely high price and it is continuously lowered until a buyer takes the commodity at the current price. In a *Double* auction both sellers and buyers submit bids which are subsequently ranked from highest to lowest to create demand and supply profiles by the auctioneer. From the profiles, the maximum quantity exchanged can be determined by matching selling offers (starting with the lowest price and going up) with demand bids (starting with the highest price and moving down).

In a computational grid environment, users need multiple processing elements (commodities) simultaneously for their workflow, parallel or parameter-sweep type of applications. Many of the mentioned auction models are not directly applicable to computational grids, since they allocate a single commodity at once [57]. Thus, a combinatorial auction mechanism is necessary in which users submit bids for a combination of resources. Besides, since the grid is a type of distributed system we intuitively do not want to place a central auctioneer. Instead, each resource owner can perform auctioning on its own. Also, to reduce the number of communication messages sealed-bid auction models should be preferred in which only a single round of communication takes place. In the next subsection, we present a double auction model considering these limitations.

4.2.1 A Combinatorial Double Auction Model

First of all, an auction mechanism can only be used if the resources are space-shared machines. Since several jobs can be run on a time-shared processing element we could not consider it as a discrete commodity to be auctioned.

In this market model, resource owners declare a minimum price that the users have to bid more, to participate in an auction. We modified our simulation architecture that each resource owner has an individual auctioneer that performs double auctions on behalf of its owner. When the resource has any free processing elements, the auctioneer accepts sealed-bids from users during a predetermined time interval. Each job that belongs to an application is represented by a single bid that includes the information of the length of the job (in MI), the bid price-per-MI (*i.e.* the offered price), and the multiplication of both, that is the total amount the user is willing to pay for the job. We assume that each processing element of the resource submits identical bids that offer the same price (*i.e.* the minimum price that the owner has specified) to qualify the model as double auction.

When auctioning multiple commodities, all winning bidders should pay for the items at the same price [76]; therefore, at the end of a bidding interval auctioneers determine a market-clearing price for the winners. To illustrate this phase, let's consider an example in which an auctioneer receives the bids illustrated in Fig. 4.1, and assume that there are three free processing elements at that time. At first, the bids are sorted in the decreasing order of payment amount. Here the winners are the first three bids since there are three free processing elements (bids of User-1 and User-2). Then, the market clears at the minimum price-per-MI (PPMI), through the winner biddings. In this case, the market-clearing price is 0.2 G$. This means that both User-1 and User-2 are going to pay 0.2 G$ per MI for the respective jobs. Resources sort the bids according to their payment amount rather than the PPMI, since we assume that they consider short-term profits.

USER−1	**USER−2**	**USER−1**	**USER−3**
Job Length: 500 MI PPMI: 0.25 G$ Payment: 125 G$	Job Length: 400 MI PPMI: 0.20 G$ Payment: 80 G$	Job Length: 200 MI PPMI: 0.25 G$ Payment: 50 G$	Job Length: 100 MI PPMI: 0.25 G$ Payment: 25 G$

Figure 4.1: Bids submitted to an auctioneer

In this auction model, users have to bid their true price since the bids are sealed. They know that they can only win if they bid higher than that of everyone else. Therefore, the model is incentive compatible. Essentially, we assume that they do not cheat on the length of their jobs to bid a lower PPMI with the same payment amount, or we assume that the resources verify the lengths.

From the users' view, initially the budget is shared among the jobs of the application in proportion to the lengths. Actually, the ratio of total (remaining) budget to the total (remaining) job length is the bid price-per-MI value of the user at a particular moment. Next, the resources are sorted in the decreasing order of processing speeds. Since they reveal their true valuations for the resources, they naturally prefer faster resources to execute their jobs. A user submits bids only if there are free processing elements and she meets the minimum price condition of the corresponding resource. If there are no free processing elements, or the minimum price condition could not be met for the corresponding resource, the user considers the next resource in the list. As an important fact, if the minimum price condition could not be met for none of the resources, then the user may start to remove some of the jobs from the job-list until the bid PPMI becomes sufficient to be participated in one of the auctions. If the user has managed to submit bids to any auctioneer, she waits until the auction session ends, and receives a reply from the auctioneer that informs about the jobs that has been accepted or denied.

4.3 Experimental Results

In this section, we present the results of the experiments that compare the pricing policies of the commodity market model, and the experiments that compare the auction model with the commodity market model.

Some of the experimental properties and parameters are same for all of the experiments. The attributes of the simulated resources are given in Table 4.1. As an exception, in the pricing comparison experiments resources are simulated as time-shared and in the market model comparison experiments resources are simulated as space-shared machines. Users submit only parameter sweep type of applications within a 10000 time interval, based on a Poisson arrival process. The experiments were repeated 10 times for each average arrival rate value, λ, that were varied from 5.10^{-4} to 5.10^{-3} with a 5.10^{-4} step size. The variation of the parameters related to Gridlets (jobs) are given in Table 4.2.

Table 4.1: Attributes of the simulated resources

Resource Name	Number of PEs	MIPS Rating of each PE	Min. Price per PE unit time in G$	Min. PPMI in G$
R1	32	10	3.5	0.35
R2	36	12	4.7	0.391
R3	48	13	5.2	0.4
R4	24	15	6.3	0.42
R5	36	20	9	0.45
R6	16	12	4.6	0.383
R7	24	11	4.1	0.372
R8	40	10	3.6	0.36
R9	24	10	3.4	0.34

Table 4.2: Variation intervals for job parameters

Number of Jobs in an Application	Size of a Job	Input File Size	Output File Size
50-150	1000-10000 MI	250-750 B	2.5-7.5 KB

4.3.1 Comparison of the Pricing Policies

For deadline and budget of each user, a relaxed and a tight (limited) value are computed considering the size of the application, and the minimum prices of the resources, in the way we did in section 3.5. Subsequently, from the tight-relaxed distributions a random budget and a random duration are assigned to the user. Resource owners enter the market with the minimum prices. The sale period of the stochastic approximation based pricing policy, T, and the price update interval of the dynamic pricing policy is set to 5 time units, and the desired utilization rate parameter, μ, is set to 0.5.

Fig. 4.2 demonstrates the average income distribution of resource owners. The average total gain under the fixed, dynamic and stochastic approximation (SA) based pricing policies are 2,654,830 G$, 4,489,723 G$, and 2,677,156 G$ respectively. Under the fixed pricing policy R5 holds the 74% of the total market gain, leaving only small shares to the other resources. This is becasue R5 is the most suitable (preferable) resource in terms of time (it is the fastest resource) and time-cost (it has the highest MIPS/PPMI ratio) optimization. Since prices are constant in the fixed pricing policy, resources are not able to adjust their

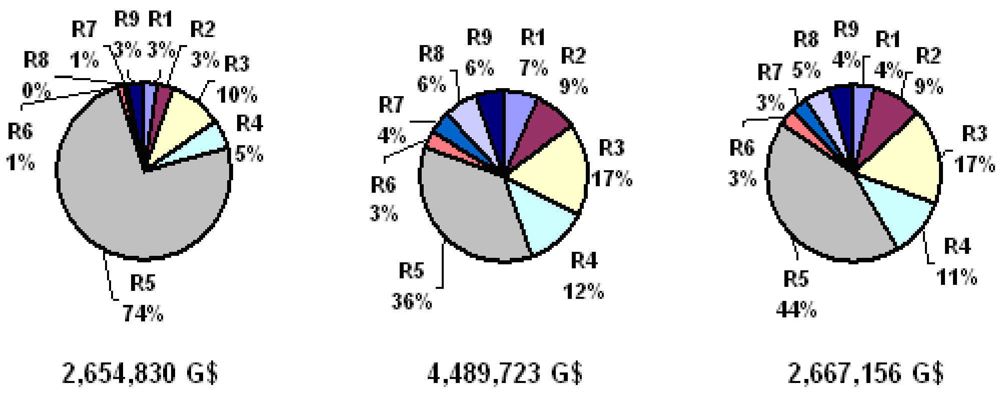

Figure 4.2: Average income distribution under the fixed (the chart on the left), the dynamic (the chart on the middle), and stochastic approximation based (the chart on the right) pricing policies

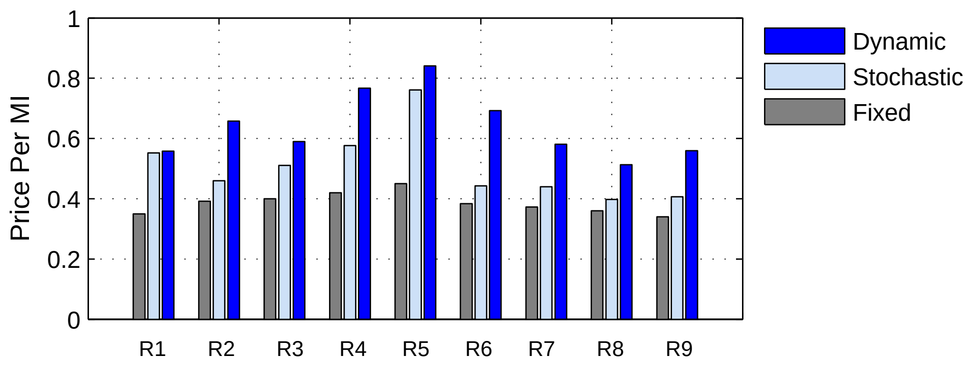

Figure 4.3: Average price-per MI of the resources under the pricing policies

prices to attract the users. However, all of the resources have been able to earn money under the dynamic and SA based policies, breaking the monopoly of R5.

Resources (commodities) are not identical, and tastes and preferences differs among users (buyers) in this market; they have different requirements such as optimizing time, cost or both of them. In such a market, attaining a fair income share is a though issue, since resources have different values to the users. As another consequence, there cannot be an equilibrium point for the prices in this market. As Fig. 4.3 illustrates price-per MI values varies among the resources. However, identical or comparable resources in terms of speed, have similar prices under the dynamic and SA based policies. Therefore, we can claim that

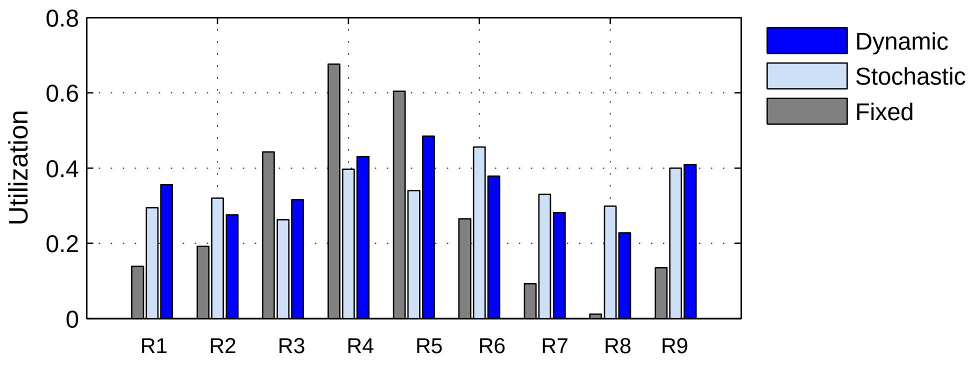

Figure 4.4: Average utilization of the resources under the pricing policies

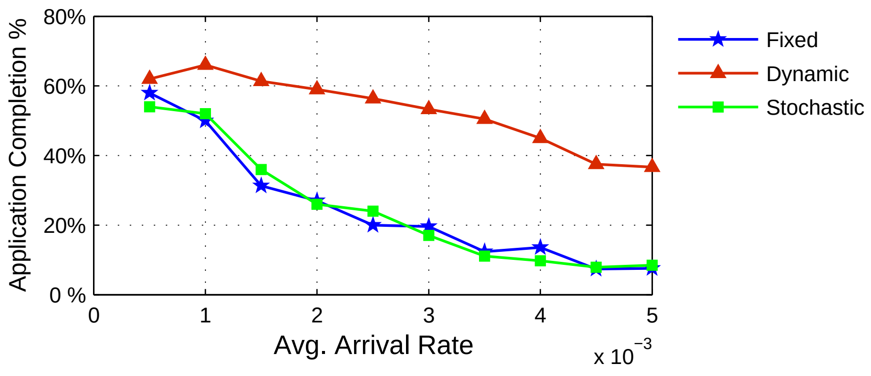

Figure 4.5: Average application completion percentages under the pricing policies

these policies achieve a partial equilibrium in the market.

The SA based policy, in fact, quickly converges to a local optimum price; however, after a certain amount of time (as A/Δ ratio increases) it behaves like the fixed price policy. Therefore, it is not versatile as the dynamic policy in adapting to the demand variations. This is clearly the reason for the income difference.

Average utilization of a system gives clear indications of overall system performance. The average utilization of all resources under the fixed, dynamic, and SA based policies are 28%, 35%, and 34% respectively. As Fig. 4.4 shows, the average utilization of the resources are more balanced under the SA based and dynamic policies than that of the resources under the fixed policy.

Fig. 4.5 shows the average percentage of the applications completed without a budget

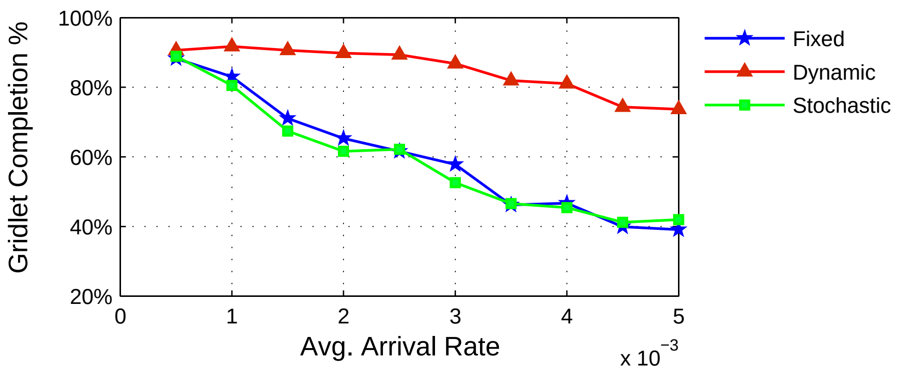

Figure 4.6: Average Gridlet completion percentages under the pricing policies

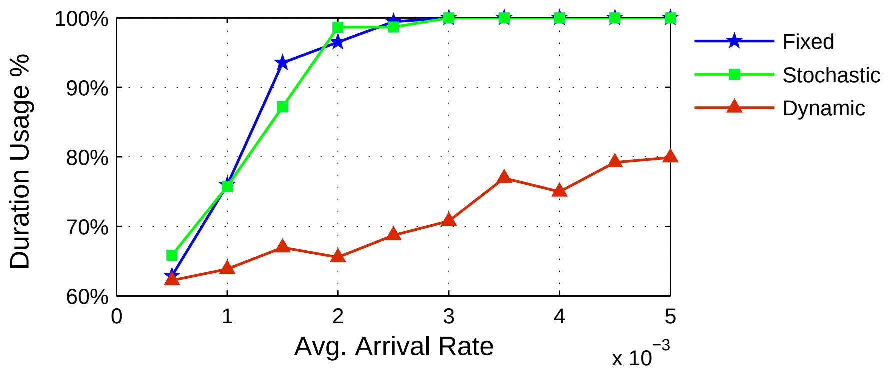

Figure 4.7: Average duration usage percentages under the pricing policies

shortage or a deadline violation, and Fig. 4.6 shows the average Gridlet completion percentages under the pricing policies. It can be seen that more Gridlets and therefore more applications are completed under the proposed dynamic pricing policy. This is because dynamic policy makes users, rapidly, tend to less loaded and cheaper resources when the formerly suitable resources became loaded and consequently their prices are increased. As Fig. 4.7 demonstrates, under the fixed and SA based policies, users' deadlines expire as the demand on the resources increases; hence, Gridlet and application completion percentages decreases.

To sum up, considering the experimental results we can assert that the dynamic pricing in which prices are adjusted with respect to the real-world market dynamics yields better social outcomes than the other pricing policies.

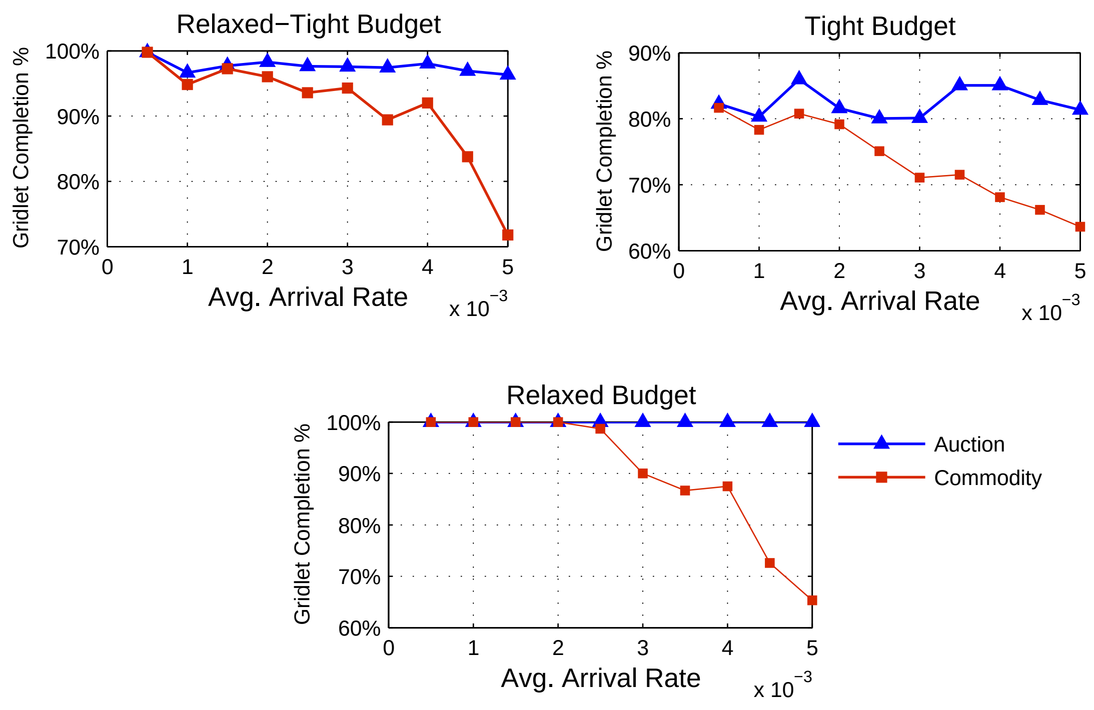

Figure 4.8: Average Gridlet completion percentages under the market models

4.3.2 Comparison of the Market Models

In the auction model, users do not have the chance to trade off between cost and time optimization strategies explicitly, since the prices are undefined. However, the optimization strategies are realized implicitly. First, the users start to bid on the resources in the decreasing order of their processing speeds, which means that they start with time optimization strategy. However, if their budgets are not enough for bidding on the resources, they tend to less expensive and probably less capable resources, which can be considered as time-cost optimization strategy. Finally, if their budgets became so insufficient that they had to remove some of their jobs, they stuck on the cheapest resources, which is certainly the cost optimization strategy.

We performed three sets of experiments to observe the outcomes of the market models. First, users are randomly assigned budgets from the tight-relaxed normal distribution, second, users are assigned only tight budgets, and last, they are all assigned relaxed budgets. In the experiments, we remove the deadline constraints to observe the true time performances of the market models.

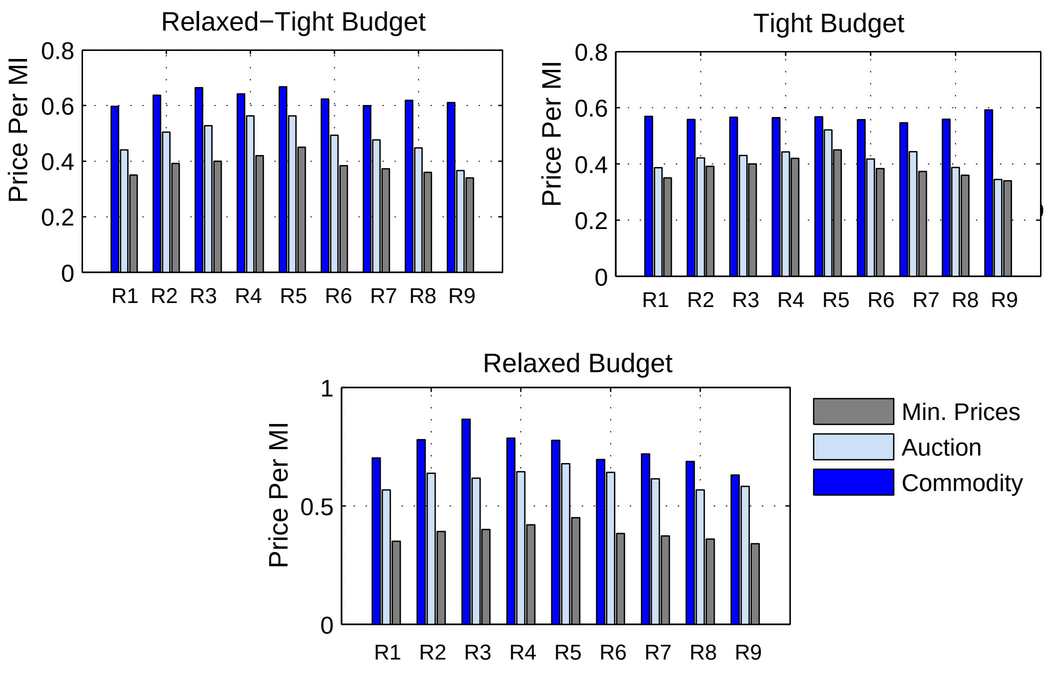

Figure 4.9: Average price-per MI of the resources under market models, when there is high demand in the market (*i.e.* $\lambda > 35.10^{-4}$)

For the commodity market model, resource owners use the dynamic pricing policy with $\mu = 0.5$. In addition, a guaranteed approach is adopted such that the heuristic waits instead of submitting jobs to a fully loaded resource; which is needed for a fair comparison with the auction model. Since we remove the deadline constraints, we have to modify our heuristic such that an optimization strategy is chosen by comparing the randomly alloted budget with the mean of random distribution.

The parameter T that represents the price update time interval for the dynamic pricing policy and bid acceptance time interval for the double auction model is set to 5 time units. Actually, we performed experiments with reasonable T values such as $T = 5, 10, 20, 40$, and received almost the same results for each case. This indicates that any small T value, in comparison to the average execution time of a job, can easily be chosen.

Actually, the main difference between the market models is the way the prices are determined. In the auction model prices are determined by the users; however, in the commodity market model prices are determined by the resource owners. Fig. 4.8 illustrates the average percentage of the completed Gridlets under the market models. Up to a specific arrival rate, *i.e* 2.10^{-3}, similar Gridlet completion percentages are revealed for all type of budget assignments. But after this point Gridlet completion percentages fall down under the commodity market model. This is due to the high price increase on the resources as the consequence of the demand increase in the market. Figure 4.9 illustrates this situation. As it can be seen the average price-per MI values revealed higher under the commodity market model when there is a high demand in the market. As Fig. 4.10 demonstrates, less money is spent in the commodity market model compared to the auction model. Since users bid their true values in the auction model, high budget consumption is in fact an expected result. Consequently, we can claim that commodity market model is more advantageous when the demand is low or normal in the market, and vice versa when the demand is high.

As Fig. 4.11 illustrates, considerable amount of time elapsed to complete the execution of jobs under the auction model with tight budget assignment. All the users with limited budgets, under the auction model, only stuck on the R9 (see Fig. 4.13) waiting for the completion of the jobs to get free processing elements.

Fig. 4.12 demonstrates the average income distribution and average total incomes of resource owners. All of the resources are able to earn money under the market models; however, resource owners are managed to earn more money under the auction model. And, Fig. 4.13 shows that no resource has left idle, and average utilization of the resources are balanced in general, except R9.

To sum up, the superiorities of the market models are relative. Here, we do not consider the deadline constraints; in such a case, the users may suffer under the auction model since their applications may not be completed. Regarding the performed set of experiments, we can conclude that resource owners are happier under the auction model. But if we consider deadlines again, this situation may change as well.

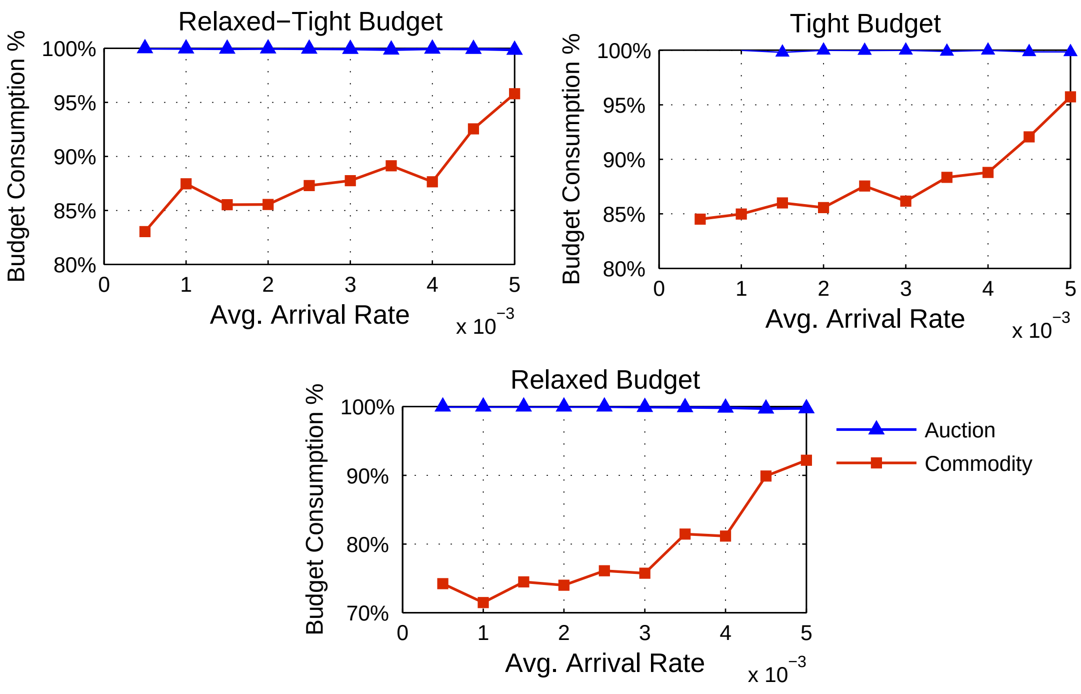

Figure 4.10: Average budget consumption percentages under market models

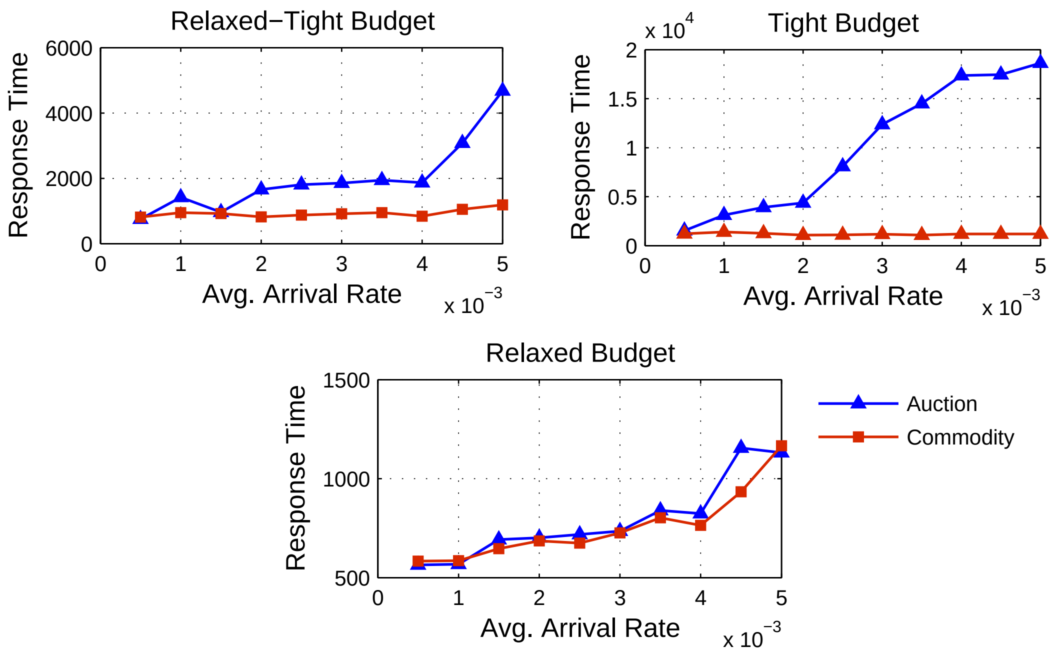

Figure 4.11: Average response time

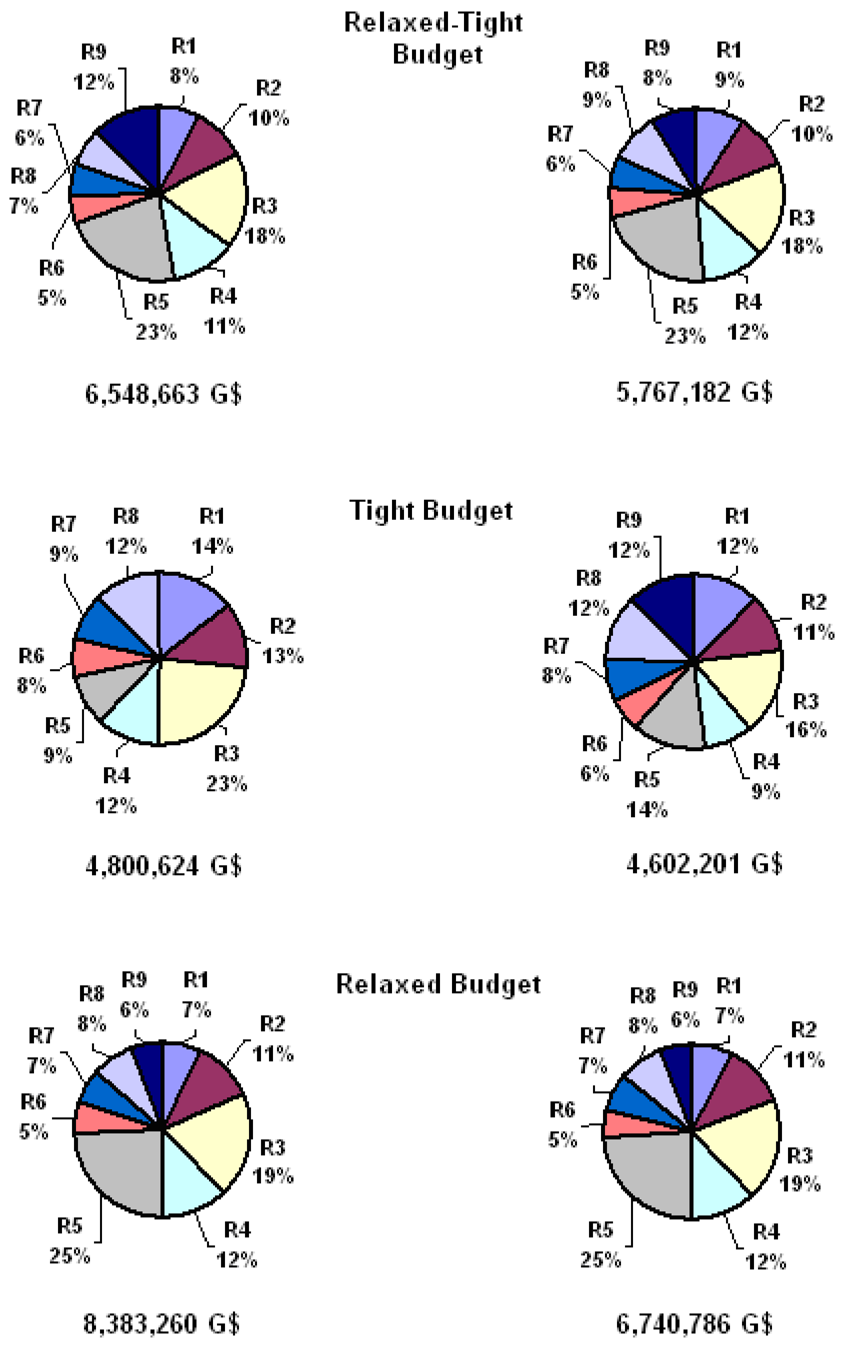

Figure 4.12: Average income distribution under the auction (the left column), and the commodity market (the right column) models

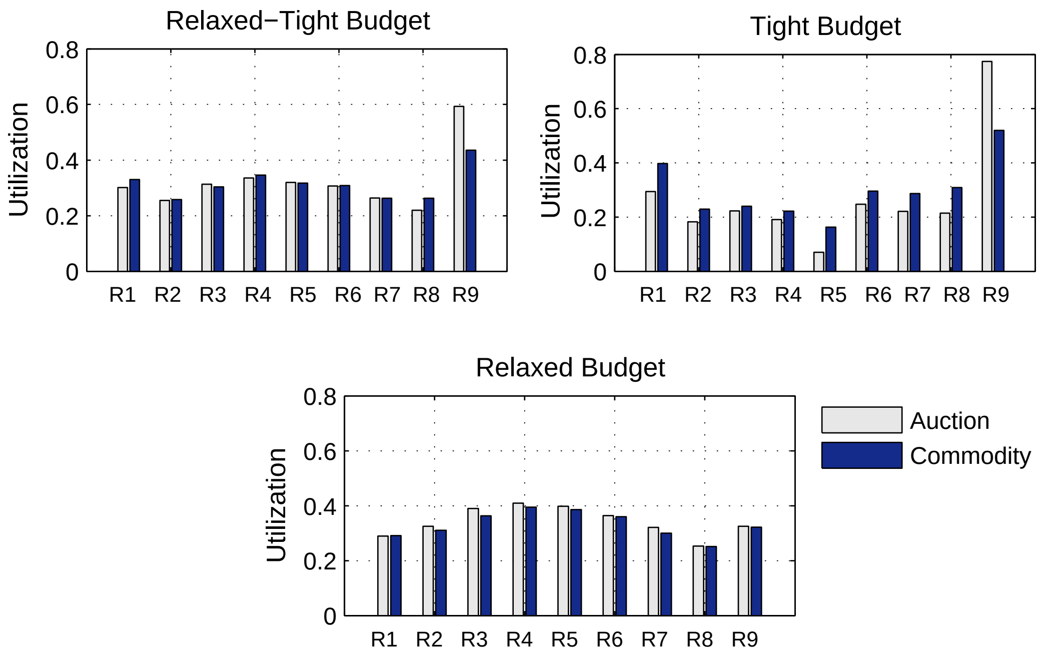

Figure 4.13: Average utilization of the resources under the market models

Chapter 5

CONCLUSIONS

5.1 State of the Art

Over the last decade, there has been much interest in computational grids which provide transparent access to large-scale distributed computational resources [58]. The main issue in such environments is the efficient allocation of these resources. Several scheduling and resource allocation systems have been developed to solve this problem. It has been also considered that constructing computational grid economy (market) systems may bring out several advantages, compared to these studies. Such a system would motivate people and companies to share their processing and data resources in order to gain money, which would consequently yield a great computing power for the scientific or private-sector community. It would also motivate scientific associations to share their resources, since they would be able to recover their costs. In addition, pricing the resources would restrict excessive resource usage (*i.e.* "Tragedy of the Commons" phenomenon [77]) since pricing is considered as a natural way of setting priorities to the users. Finally, scheduling would be a natural outcome in such a market where both resource owners and users can make their own decisions to maximize their utility and profit.

Nevertheless, the idea of constructing computational grid markets is rather new. There are no such real systems currently. However, some companies such as IBM [78], Parabon [79] and United Devices [80] give significant attention to this topic, and there are ongoing projects in the academic world [6, 60, 64]. We believe that grid economy systems will gain more attention as these studies progress and they will be the main computing infrastructures of the future.

5.2 Summary

The work presented in this thesis has an interdisciplinary nature; we have considered both economical and computational aspects of the grid systems in all phases of the study. This helped us to develop models and strategies that could satisfy both resource owners and users of a computational grid economy system in many respects.

First, we have investigated the problem of job scheduling in a computational grid economy system with the goal of minimizing both the makespan and the cost which is spent for the use of processing elements. Towards this goal, we have devised the SFTCO heuristic that is designed to complete all the jobs of a user within the budget and deadline constraints. The heuristic is capable of dealing both the sequential workflow and parameter sweep type of applications. We performed simulation studies to test the performance of our heuristic. In terms of job completion percentages, our heuristic outperformed the heuristics of Nimrod/G system for parameter sweep type of applications, and outperformed a random scheduling strategy for sequential workflow type of applications.

We have also presented two market models; commodity market and combinatorial double auction, for computational economy grids in which both users and resource owners can ensure their self-interests. In the commodity market model, the user preferences are represented by the optimization strategies of the SFTCO heuristic. And for resource owners we have considered a fixed, a dynamic and a stochastic approximation based [15] policy. The comparison experiments revealed that our dynamic pricing policy is more successful as a means for achieving social welfare that is the utility of all market agents considered in aggregate. In other words, users have managed to complete more jobs, and resource owners have managed to earn more money. In the combinatorial double auction model, resource owners can auction their own processing elements, and users can get multiple commodities concurrently, which differentiates the model from the typical auction studies in the grid computing field. Finally, we have compared the commodity market model in which resource owners use the dynamic pricing policy, with the double auction model. The experimental results showed that the models have both advantages and disadvantages in terms of achieving social welfare.

Considering the distributed nature of the resources, the decision making is decentralized in the models. There do not exist any central mechanism that adjusts the prices; each agent tries to maximize its own utility and profit. Moreover, models require relatively small communication overhead compared to negotiation based models such as tatonnement process, bargaining and open-bid auctions. In the commodity market model, agents interact with each other indirectly through the price mechanism; hence no communication takes place to determine the price. And the auction model is a type of sealed-bid that a single round of communication takes place between a user and the auctioneer in an auction session.

5.3 Future Directions

In SFTCO heuristic, when deciding whether to perform time optimization, we compare the duration alloted to the stage with the worst-case estimation of completion time. In fact, several studies in the literature such as [81] and [82] propose better duration estimation techniques for a set of independent parallel tasks. These techniques can be easily applied to the SFTCO heuristic.

The presented commodity market model is a spot market in which the resources are traded for immediate delivery and payment. As a future work, an advance reservation system (*i.e.* futures market [77]) can be applied to the commodity market model to reduce price volatility and ensure reliance among the market agents. Moreover, in the auction model, the minimum bidding prices that the resource owners determine, can be adaptive according to the demand (*i.e.* utilization) on the resource. It would be an interesting contribution since it associates the commodity market model with the auction model. Also, in this work, we only consider the processing elements as commodities; a computational grid economy system, in fact, should address pricing of many other requirements such as memory, data or bandwidth.

Appendix A

THE GRIDSIM TOOLKIT

GridSim [2, 14] is a Java based, open source, and discrete event based simulation toolkit developed for modeling and simulation of application scheduling on a range of classes of parallel and distributed computing systems such as clusters, grids, and P2P networks. The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution.

It is possible to simulate both conventional and economic-based distributed systems with this toolkit. Besides, developers can implement their own schedulers, resource management policies, pricing policies and different application models such as parameter sweep, process parallelism, workflow, and divide&conquer.

Some of the important features of the toolkit are as follows:

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space-shared or time-shared policies.
- Resource capability can be defined in the form of MIPS (Million Instructions Per Second) in accordance with SPEC (Standard Performance Evaluation Corporation) benchmark.
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on resources local time to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared.

- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

In the GridSim toolkit, Processing Elements (PEs) can be created with identical or different speeds. Besides, one or more PEs can be put together to create a machine. Similarly, one or more machines can be put together to create a grid resource. Thus, the resulting grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These grid resources can simulate time-shared or space-shared scheduling depending on the allocation policy.

A.1 Simulation of Scheduling in Time-Shared Resources

When jobs arrive, time-shared resources start their execution immediately and share resources among all jobs [14]. Whenever a new job arrives, the processing time of existing jobs are updated and then this newly arrived job is added to the execution set. Depending on the number of jobs in execution and the number of PEs in a resource, GridSim allocates appropriate amount of PE share to all jobs using the Round Robin algorithm. It should be noted that jobs that are sharing the same PE would get an equal amount of PE share. At the completion of a job, it is sent back to its originator and removed from the execution set.

A.2 Simulation of Scheduling in Space-Shared Resources

When jobs arrive, space-shared resources start jobs' execution immediately if there is a free PE available, otherwise, it is queued [14]. During the job assignment, job-processing time is determined and event is scheduled for delivery at the end of execution time. Whenever the job finishes and the internal event is delivered to signify the completion of scheduled job, the resource simulator frees the PE alloted to it and then checks whether there are any other jobs waiting in the queue. If there are, then it selects a suitable job depending on the policy (*i.e.* FCFS) and assigns to the PE, which is free. The completed jobs are sent back to its originator and removed from the execution set.

Appendix B

HEURISTICS OF THE NIMROD/G SYSTEM

B.1 Deadline and budget constrained scheduling with cost optimization

DBC_Scheduling_with_Cost_Optimization()

1. Identify cost of each resource in terms of CPU cost per second and capability to be delivered per cost-unit.

2. **SORT** resources by increasing order of cost.

3. **SCHEDULING:** Repeat while there exist unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event **AND** the time and process expenses are within deadline and budget limits:

   (a) For each resource perform load profiling to establish the job consumption rate or the available resource share through measure and extrapolation.
   (b) For each resource based on its job consumption rate or available resource share, predict and establish the number of jobs a resource can process by the deadline.
   (c) For each resource in order:
      i. If the number of jobs currently assigned to a resource is less than the predicted number of jobs that a resource can consume, assign more jobs from unassigned job queue or from the most expensive machines based on job state and feasibility. Assign job to a resource only when there is enough budget available.
      ii. Alternatively, if a resource has more jobs than it can complete by the deadline, move those extra jobs to unassigned job queue.

4. Repeat the following steps for each resource if it has jobs to be dispatched:

   (a) Identify the number of jobs that can be submitted without overloading the resource. The default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

B.2 Deadline and budget constrained scheduling with time optimization

DBC_Scheduling_with_Time_Optimization()

1. Identify cost of each resource in terms of CPU cost per second and capability to be delivered per cost-unit.

2. **SCHEDULING:** Repeat while there exist unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event **AND** the time and process expenses are within deadline and budget limits:

   (a) For each resource, predict and establish the job consumption rate or the available resource share through the measure and extrapolation strategy taking into account the time taken to process previous jobs.

   (b) If any of the resource has jobs assigned to it in the previous scheduling event, but not dispatched to the resource for execution and there is variation in resource availability, then move appropriate number of jobs to the Unassigned-Jobs-List. This helps in updating the whole schedule based on the latest resource availability information.

   (c) Repeat the following steps for each job in the Unassigned-Jobs-List:

      i. Select a job from the Unassigned-Jobs-List.

      ii. Create a *resource group* containing affordable resources (*i.e.*, whose processing price is less than or equal to the remaining budget per job).

      iii. For each resource in the resource group, calculate/predict the job completion time taking into account previously assigned jobs and the job completion rate and resource share availability.

      iv. Sort resources in the resource group by the increasing order of job completion time.

      v. Assign the job to the first resource in the resource group and remove it from the Unassigned-Jobs-List if the predicted job completion time is less than the deadline.

3. Repeat the following steps for each resource if it has jobs to be dispatched:

   (a) Identify the number of jobs that can be submitted without overloading the resource. The default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

B.3 Deadline and budget constrained scheduling with cost-time optimization

DBC_Scheduling_with_Cost-Time_Optimization()

1. Identify cost of each resource in terms of CPU cost per second and capability to be delivered per cost-unit.

2. **SCHEDULING:** Repeat while there exist unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event **AND** the time and process expenses are within deadline and budget limits:

   (a) For each resource, predict and establish the job consumption rate or the available resource share through the measure and extrapolation strategy taking into account the time taken to process previous jobs.

   (b) **SORT** the resources by increasing order of cost. If two or more resources have the same cost, order them such that powerful ones are preferred first.

   (c) Create *resource groups* containing resources with the same cost.

   (d) **SORT** the *resource groups* with the increasing order of cost.

   (e) If any of the resource has jobs assigned to it in the previous scheduling event, but not dispatched to the resource for execution and there is variation in resource availability, then move appropriate number of jobs to the Unassigned-Jobs-List. This helps in updating the whole schedule based on the latest resource availability information.

   (f) Repeat the following steps for each *resource group* as long as there exists unassigned jobs: (Use time optimization strategy)

      i. Select a job from the Unassigned-Jobs-List.

      ii. Create a *resource group* containing affordable resources (*i.e.*, whose processing price is less than or equal to the remaining budget per job).

      iii. For each resource in the resource group, calculate/predict the job completion time taking into account previously assigned jobs and the job completion rate and resource share availability.

      iv. Sort resources in the resource group by the increasing order of job completion time.

      v. Assign the job to the first resource in the resource group and remove it from the Unassigned-Jobs-List if the predicted job completion time is less than the deadline.

3. Repeat the following steps for each resource if it has jobs to be dispatched:

   (a) Identify the number of jobs that can be submitted without overloading the resource. The default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

BIBLIOGRAPHY

[1] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann, 1999.

[2] R. Buyya. *Economic-based distributed resource management and scheduling for grid computing.* PhD thesis, Monash University, 2002.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications*, 11:115–128, 1997. (http://globus.org).

[4] I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC'98*, pages 1–11, San Jose, California, USA, 1998.

[5] G. Allen, T. Dramlitsch, and I. Foster. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, page 52, Denver, Colorado, USA, 2001.

[6] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid. In *Proceedings of the High-Performance Computing*, pages 283–289, 2000.

[7] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Journal of Cluster Computing*, 5:237–246, 2002.

[8] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on project serendip data and 100,000 personal computers. In *Proceedings of the 5th International Conference on Bioastronomy*, 1997. (http://setiathome.ssl.berkeley.edu/).

[9] S.Shetty, P. Padala, and M. P. Frank. A survey of market-based approaches to distributed computing. Technical Report TR03-013, University of Florida, 2003.

[10] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[11] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Journal of Concurrency and Computation: Practice and Experience*, pages 1507–1542, 2002.

[12] H. P. Bivens. Grid workflow, 2004. http://zuni.cs.vt.edu/grid-computing/papers/draft-bivens-grid-workflow.pdf.

[13] R. Buyya, J. Giddy, and D. Abramson. An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications. In *Proceedings of the 2nd International Workshop on Active Middleware*, Pittsburgh, Pennsylvania, USA, 2000.

[14] M. Murshed, R. Buyya, and D. Abramson. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Journal of Concurrency and Computation: Practice and Experience*, pages 1–32, 2002. (http://www.buyya.com/gridsim/).

[15] N. Abe and T. Kamba. A Web marketing system with automatic pricing. *The International Journal of Computer and Telecommunications Networking*, 33:775–788, 2000.

[16] B. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report CSD-00-1092, University of California at Berkeley, 2000.

[17] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engineering*, 18(2):103–117, 1992.

[18] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the POPCORN project. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 592, Washington, DC, USA, 1998.

[19] J. Bredin, D. Kotz, and D. Rus. Market-based resource control for mobile agents. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 197–204, New York, NY, USA, 1998.

[20] Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour. Economic scheduling in grid computing. In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 128–152, London, UK, 2002.

[21] Yan Liu. Grid scheduling, 2002. http://www.cs.uiowa.edu/ yanliu/QE/QEreview.pdf.

[22] Chaitanya Kandagatla University. Survey and taxonomy of grid resource management systems. http://citeseer.ist.psu.edu/647028.html.

[23] K. Ranganathan and I. T. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[24] J. M. Schopf. A general architecture for scheduling on the grid, 2002. Special issue of JPDC on Grid Computing.

[25] Ramin Yahyapour. Grid scheduling architecture, 2002. http://www-ds.e-technik.uni-dortmund.de/.

[26] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, 32(2):135–164, 2002.

[27] D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Transactions on Software Engineering*, 15(11):1427–1436, 1989.

[28] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, 1977.

[29] X. He, X. Sun, and G. von Laszewski. Qos guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*, 18(4):442–451, 2003.

[30] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*, page 30, Washington, DC, USA, 1999. IEEE Computer Society.

[31] M. Maheswaran, T. Braun, and H. Siegel. Heterogeneous distributed computing, 1999. (citeseer.ist.psu.edu/maheswaran99heterogeneous.html).

[32] T. D. Braun et. al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel Distributed Computing*, 61(6):810–837, 2001.

[33] B. A. Shirazi, K. M. Kavi, and A. R. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[34] R. F. Freund and H. J. Siegel. Guest editor's introduction: Heterogeneous processing. *Computer*, 26(6):13–17, 1993.

[35] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 349, Washington, DC, USA, 2000.

[36] E. Srisan and P. Uthayopas. Heuristic scheduling with partial knowledge under gird environment. Presented at the Second International Symposium on Communications and Information Technology, 2002.

[37] T. G. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[38] D. Yu and T. G. Robertazzi. Divisible load scheduling for grid computing. Presented at the Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems Technology, 2003.

[39] M. Moges, D. Yu, and T. G. Robertazzi. Grid scheduling divisible loads from multiple sources via linear programming. In *16th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 423–428, 2004.

[40] J T. Hung and T. G. Robertazzi. Scalable scheduling for clusters and grids using cut through switching. *International Journal of Computers and their Applications*, 26(3):147–156, 2004.

[41] W. H. Min, B. Veeravalli, D. Yu, and T. G. Robertazzi. Data intensive grid scheduling: Multiple sources with capacity constraints. Presented at 15th Int. Conf. Parallel and Distributed Computing and Systems, 2001.

[42] H. Zhao and R. Sakellariou. A low-cost rescheduling policy for dependent tasks on grid computing systems. In *Proceedings of the 2nd AcrossGrids*, 2004.

[43] H. Chen and M. Maheswaran. Distributed dynamic scheduling of composite tasks on grid computing systems. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 119, Washington, DC, USA, 2002.

[44] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: user-level middleware for the grid. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, pages 75–76, Washington, DC, USA, 2000.

[45] The data grid project, 2005. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[46] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the International Conference of Distributed Computing Systems*, pages 104–111, San Jose, Ca., 1988.

[47] Network weather servise, 2005. http://nws.cs.ucsb.edu/.

[48] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw. The Legion resource management system. In *JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 162–178, London, UK, 1999.

[49] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. Technical Report UT-CS-95-313, 1995.

[50] Condor project homepage, 2005. www.cs.wisc.edu/condor/.

[51] J. Nakai. Reading between the lines and beyond. Technical Report NAS-01-010, NASA Ames Research Center, 2002.

[52] Glossary of terms, 2005. www.investopedia.com/terms/.

[53] T. Malone, R. Fikes, K. Grant, and M. Howard. *Enterprise: A market-like task scheduler for distributed computing environments*, pages 177–205. North-Holland, 1988.

[54] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[55] S. Smale. Convergent process of price adjustment and global newton methods. *Journal of Mathematical Economics*, 3:107–120, 1976.

[56] M. Wellman, W. Walsh, P. Wurman, and J. Mackie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[57] C.Chien, P. Chang, and V. Soo. Market-oriented multiple resource scheduling in grid computing environments. In *AINA*, pages 867–872, 2005.

[58] J. Gomoluch and M. Schroeder. Performance evaluation of market-based resource allocation for grid computing. *Concurrency - Practice and Experience*, 16(5):469–475, 2004.

[59] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium*, page 10046.2, Washington, DC, USA, 2001.

[60] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service oriented grid computing. Presented at 10th Heteregeneous Computing Workshop, 2001.

[61] L. He and T. R. Ioerger. Task-oriented computational economic-based distributed resource allocation mechanisms for computational grids. The 2004 International MultiConference in Computer Science, 2004.

[62] B. Stiller, J. Gerke, P. Flury, P. Reichl, and Hasan. Charging distributed services of a computational grid architecture. In *CCGRID*, pages 596–601, 2001.

[63] R. Buyya and D. Abramson ans S. Venugopal. The grid economy. In *Special Issue of the Proceedings of the IEEE on Grid Computing*, 2004.

[64] R. Buyya and S. Vazhkudai. Compute power market: Towards a market-oriented grid. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 574, Washington, DC, USA, 2001.

[65] O. Sonmez and A. Gursoy. A novel economic-based scheduling heuristic for computational grids. Presented at the 2nd International Conference on Computational Science and Engineering, 2005.

[66] T. Cheatham, A. Fahmy, D. C. Stefanescu, and L. G. Valiant. Bulk synchronous parallel computing-a paradigm for transportable software. In *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences*, page 268, Washington, DC, USA, 1995. IEEE Computer Society.

[67] D. B. Skillicorn, J. M. Hill, and W. F. McColl. Questions and answers about BSP. *Scientific Programming*, 6(3):249–274, 1997.

[68] S. Aytuna. Automated prediction of protein-protein interactions. Master's thesis, Koc University, 2005.

[69] P. V. Jithesh, N. Kelly, P. Donachy, J. Harmer, R. Perrott, M. McCurley, M. Townsley, J. Johnston, and S. McKee. Genegrid: Grid based solution for bioinformatics application integration and experiment execution. In *CBMS*, pages 523–528, 2005.

[70] The Standard Performance Evaluation Corporation, 2005. http://www.spec.org/.

[71] A. Y. Zomaya, M. Clements, and S. Olariu. A framework for reinforcement-based scheduling in parallel processor systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):249–260, 1998.

[72] L. He and T. R. Ioerger. Forming resource-sharing coalitions: a distributed resource allocation mechanism for self-interested agents in computational grids. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 84–91, New York, NY, USA, 2005.

[73] T. Mullen and J. Breese. Experiments in designing computational economies for mobile users. In *ICE '98: Proceedings of the first international conference on Information and computation economies*, pages 19–27, New York, NY, USA, 1998.

[74] D. W. Walker. Free-market computing and the integration of computers into the global economic infrastructure. *International Journal of Computers and their Applications*, 1(2):60–62, 1993.

[75] M. T. Wasan. *Stochastic Approximation.* Cambridge University Press, 1969.

[76] Inc. Agorics. Auctions, 2005. http://www.agorics.com/Library/auctions.html.

[77] G. Cheliotis, C. Kenyon, and R. Buyya. Grid economics: 10 lessons from finance. Technical Report GRIDS-TR-2003-3, IBM Research Zurich and Grid Computing and Distributed Systems Laboratory, University of Melbourne, 2003.

[78] Kathleen McGraw. IBM launches economic development grid initiative; greater Cleveland first region to benefit, 2005. http://www-1.ibm.com/grid/grid-press/pr-051805.shtml.

[79] Parabon, 2005. www.parabon.com.

[80] United Devices, 2005. http://www.ud.com/home.htm.

[81] Keqin Li. Analysis of an approximation algorithm for scheduling independent parallel tasks. *Discrete Mathematics and Theoretical Computer Science*, 3(4):155–166, 1999.

[82] Keqin Li. Average-case performance analysis and validation of online scheduling of independent parallel tasks. In *IPDPS*, 2004.

VITA

Ö. Ozan Sönmez was born in Istanbul, Turkey on September 06, 1980. He received his B.Sc. degree in Computer Engineering from Istanbul Technical University, Istanbul, in 2003. From September 2003 to August 2005, he worked as a teaching and research assistant in Koç University, Istanbul, Turkey and studied to develop “Scheduling in a Computational Grid Economy System” project. He has attended ICCSE2005 (Istanbul, Turkey) conference where he presented a paper about the project. At the time of press, he had an Ph.D. admission from the Technical University of Delft, Delft, Netherlands.