Economic scheduling in Grid computing using Tender model

by

Mohammad Bsoul

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy

of

Loughborough University

May 2007

This thesis is dedicated to

Abedel Rahman and Najieh, my parents.

Acknowledgements

First of all, I must thank my PhD supervisor, Dr. Iain W Phillips, who made this thesis possible. I would also like to thank my director of research, Chris Hinde, for his helpful advice. I am also grateful to Department of Computer Science at Loughborough University for accepting me into the PhD program.

I also have to thank my friends and colleagues: Nacho, Javier, Jose, John, Lezan, Mark, Matthew, Fadi, Mutaz, Ashraf, Tareq, Ahmad and Hesham, possibly many more.

Lastly, and most importantly, I would like to thank my parents, Abedel Rahman and Najieh. They raised me, supported me, taught me and loved me. For these reasons, this thesis is dedicated to them, with all my love and respect.

Abstract

Economic scheduling needs to be considered for Grid computing environment, because it gives an incentive for resource providers to supply their resources. Moreover, it enforces efficient use of resources, because the users have to pay for their use. Tendering is a suitable model for Grid scheduling because users start the negotiations for finding suitable resources for executing their jobs. Furthermore, the users specify their job requirements with their requests and therefore the resources reply with bids that are based on the cost of taking on the job and the availability of their processors. In this thesis, a framework for economic Grid scheduling using tendering is proposed. The framework entities such as users, brokers and resources employ tender/contract-net model to negotiate the prices and deadlines. The brokers' role is acting on behalf of users. During the negotiations, the entities aim to maximise their performance which is measured by a number of metrics.

In order to evaluate the entities' performance under different scenarios, a Java-based simulator, called MICOSim, supporting event-driven simulation of economic Grid scheduling is presented. MICOSim can perform a simulation of more than one hundred entities faster than real time. It is concluded from the evaluation that users who are interested in increasing the job success rate and paying less for executing their jobs have to consider received prices to select the most appropriate

bids, while users who are interested in improving the job average satisfaction rate have to consider either received completion time or both price and completion time to select the most suitable bids when the submission of jobs is static. The best broker strategy is the one that doesn't take into account meeting the job deadlines in the bids it sends to job owners. Finally, the resource strategy that considers the price to determine if to reply to a request or not is superior to other resource strategies. The only exception is employing this strategy with price that is too low. However, there is a tiny difference between the performances of different user strategies in dynamic submission. It is also concluded from the evaluation that broker strategies have the best performance when the revenue they target from the users is reasonable. Thus, the broker's aim has to be receiving reasonable revenue (neither too low nor too high) from acting on behalf of users. It is observed from the results that the strategy performance is influenced by the behaviour of other entities such as the submission time of user jobs. Finally, it is observed that the characteristics of entities have an effect on the performance of strategies. For example, the two user strategies that consider the received completion time and both price and completion time to determine if to accept a broker bid have similar performance, because of the existence of resources with various prices from cheap to expensive and existence of resources which don't care about the price paid for the execution. So, the price threshold doesn't have a large effect on the performance.

Keywords: Grid computing, Economic scheduling, Tender/Contract-net model, Strategy, Framework, Entity, User, Broker, Resource, Simulation.

Contents

List of Figures

List of Tables

List of Pseudocodes

Chapter 1

Introduction

1.1 An overview of Grid computing

Grid computing is the sharing, selection, and aggregation of a group of resources such as supercomputers, mainframes, storage systems, data sources, distributed applications and management systems that behaves as a network of computation [83, 19, 18, 63, 4].

The aim of Grid computing is bringing computer power to the users [16]. The enhancement in the speed and reliability of networks, in addition to the use of distribution protocols are important factors that have led to the use of Grids, to enable users to utilise resources owned by different providers remotely. A task which was impossible before [74]. Over the last three decades, the speeds of computers were doubled every year and a half. Moreover, for the last five years, the speeds of networks have doubled every eight months. In addition, the cost of manufacturing hardware is continuously decreasing [57].

Peer2Peer (P2P) computing is the sharing of PCs, which are connected to the Internet. The objective of this sharing is to provide computing power or sharing computer data between the connected PCs (Kazaa, Imesh, etc...). On the other hand, Grid computing is the aggregation of clusters of computers that

are geographically distributed using suitable protocols and standards [83]. P2P is lots of independent nodes interconnected to each other in no particular fashion, and providing a loosely connected, non-centralised system. In contrast to this, Grid is a well designed system, with system-wide standardied specification and adherence to policies which P2P lacks. P2P is not targetted towards any specific application, rather towards the aim of sharing files in a non-centralised manner. Whereas, Grid was developed for specific purpose such as scientific research.

In Grid computing, the users (customers) can rent the machines residing on the Grid for executing their computationally intensive applications instead of purchasing their own special expensive resources. This can be sometimes shown as dividing the cost of the resources participating in the Grid between the users who use the Grid.

The definition of Grid is taken from the electrical power Grid that provides pervasive access to power without caring where and how the power was produced. In the same way, the computational Grid should provide pervasive access to high end computing resources, without consideration for where and how the jobs were executed [19, 69, 57].

Grids can be classified into *computational* Grids which concern computing power, *scavenged* Grids which concern utilising unused resources and *data* Grids which concern providing access to data [83].

Amongst the main goals for employing Grid computing are:

- Providing a framework for utilising unused computational power as well as unused storage capacity.
- Simplifying collaboration between different organisations by providing direct access to computers, software and storages.
- Providing an access to resources which can't be accessed locally (remote resources).
- Executing the jobs more quickly due the fact that the job can be executed in

parallel on multiple resources (Multi-site computing/Co-allocation). However, in such cases the job needs to support parallelism.

Other distributed computing technologies can benefit from Grid technologies to solve some of their existing problems. For example, enterprise distributed computing systems can employ Grid technologies to get over the problems of sharing resources across institutional boundaries [38].

The most complicated task in Grid is the allocation process; mapping jobs to various resources. This is a known NP-complete (non-deterministic polynomial time) problem [62]. For example, the mapping of 50 jobs onto 10 resources produces 10^{50} possible mappings. This is because every job can be mapped to any of the resources. This task becomes more complicated in the case of co-allocation [23] which means that the job is executed on a number of resources instead of on a single resource [34]. Another complexity of resource allocation is the lack of accurate information about status of resources [43].

Much work was done on finding an optimal allocation. Some of the projects used conventional strategies that usually concern the overall system performance, but don't consider economics (prices) for allocating jobs to resources. On the other hand, A large number of projects use economics as strategies for allocating jobs. Economic strategies take into consideration the price of the resources when it needs to allocate jobs to resources and that price usually reflects the value of the resource to the user.

This thesis introduces a novel framework for economic scheduling in Grid computing using tender model.

1.2 Establishing our scenario

In our scenario, there are 3 classes of entities: Users, Brokers and Resources. There are n users U_1, U_2, $\cdots$, U_n, m brokers B_1, B_2, $\cdots$, B_m and w resources R_1, R_2, $\cdots$, R_w. Each user i has x_i jobs J_{i1}, J_{i2}, $\cdots$, J_{ix_i} and each resource j has y_j processors P_{j1}, P_{j2}, $\cdots$, P_{jy_j}.

The users have jobs that need to be executed. The users send their jobs to the brokers which in turn negotiate the job requirements with resources. The brokers negotiate in a way that maximises their utilities, in addition to satisfying the user requirements. The resources represent the computing power of the Grid and are where the users' jobs get executed. Resource providers make a profit by selling computing power.

Users are the entities that can only send jobs. Brokers are the entities that can send and receive jobs. Finally, Resources are the entities that just receive jobs. Each of these entities competes with other entities from the same class. Additionally, every entity employs a strategy in order to maximise its utilities (e.g. reduce the cost, maximise profit). The performance of each entity is measured by a number of metrics.

1.3 This work

In this thesis, we have designed, implemented and evaluated economy based entity scheduling strategies for Grid computing. This was achieved by:

- Discovering the related literature to have a good understanding of the topic.
- Developing a new framework for economic scheduling in Grid computing.
- Building a simulator for modelling the framework.
- Using the simulator for evaluating the performance of various entity strategies.

The aim of this thesis is to propose a new framework for decentralised economic scheduling in Grid computing using tendering. In this framework, all participating entities such as users, brokers and resources have utilities that need to be maximised by their own strategies. Within the framework, the performance of each strategy is compared to the performance of other strategies employed by entities of the same class.

1.4 Thesis structure

The present dissertation is divided into six chapters:

- The present chapter, **Chapter 1** gives an introduction to Grid computing, and describes our considered scenario.
- **Chapter 2** reviews the topics relate to scheduling in Grid computing.
- A framework for economic scheduling in Grid computing using tendering is introduced in **Chapter 3**.
- **Chapter 4** describes the MICOSim simulator for modelling economic scheduling. The simulator is used to compare the performance of the entity strategies.
- The results of the comparison is discussed in **Chapter 5**.
- Finally, **Chapter 6** concludes the thesis and defines the future lines of research to continue this work.

Chapter 2

On scheduling in Grid computing

This chapter is a review of the topics related to scheduling in Grid computing. This review covers the following topics:

1. The general components of the Grid.
2. The common application requirements.
3. Resource scheduling tasks and challenges.
4. Scheduling schemes.
5. Scheduling performance metrics.
6. Scheduling categories such as conventional and economic scheduling.
7. Market models.

2.1 Grid components

Grids consist of many components. The general components of the Grids are:

- Applications: Applications are the jobs needed to be executed on the resources of the Grid. The applications can be scientific or commercial applications. In some cases, jobs consist of subtasks that their execution results

are gathered and combined to represent the overall job result [39, 6, 27]. The split of a task into subtasks is called task profiling [62].

- Resources: The resources represent the working power of the Grid, and they are where the jobs get executed. The resources could be computational, storage or communication resources [16]. The Grid resources usually differ in speed, capacity, architecture, and operating system (heterogeneous). In economic scheduling, the resources have costs that are paid by users running their jobs on these resources. Usually, the demand and supply affect the prices. So, if the demand is higher than the supply, the prices go up, and it is the opposite if the supply is higher than the demand. The resources can be combined together to form a service which can be bought by the user. This is referred to by service aggregation [69].

- Scheduler: The task of scheduler is executing the jobs on the available resources. The scheduler is also responsible for discovery, allocation, and aggregation of resources. It should take into account the user requirements for the jobs which can also be called Quality of Service (QoS) [18, 98].

- Broker: Its role is searching the Grid to find suitable resources to execute the jobs submitted to it. It is sometimes responsible for negotiating the prices between various users and resource providers. The common situation is having many brokers act on behalf of users to find suitable resources to execute their jobs. The brokers sometimes also act on behalf of resources. The brokers can also be considered schedulers because they schedule jobs on resources. However, schedulers can't be considered brokers because they don't have the ability to negotiate.

2.2 Application requirements

As mentioned before, applications are the jobs needed to be executed on resources and sometimes they consist of smaller parts called subtasks. However, those ap-

plications have their requirements that need to be considered by the scheduler. Thus, the scheduler has to select the resources that are capable to meet those requirements.

Some of the application requirements which need to be handled by the scheduler are:

- Deadline [27]: It is the time specified by the user to finish executing its job on a resource or a collection of resources.
- Budget: It is the amount of money a user can afford for executing its job on a resource or a group of resources.
- CPU speed: This is an important requirement in computational Grids. It is the microprocessor speed required for running the application. Speed is defined as the number of instructions the processor can execute per time unit. The common used measurements are Million Instruction Per Second (MIPS) and FLoating-point OPerations per Second (FLOPS).
- Physical memory: The amount of ram needed for running the application (e.g. 256 or 512 MB).
- File size: The required storage space for running the application. This storage space is important for storing the program files needed for running it on a particular resource.
- Bandwidth: This requirement arises, because of the need to move the program to the location where it is going to be executed, and transferring the results of the execution to the user. This movement of a program code and results requires a network bandwidth, and the user might specify the amount of bandwidth needed.
- Security: It represents the minimum security level the resource must have in order to be accepted for executing the job of the user. The security is needed for a safe execution environment.

2.3 Resource scheduling tasks and challenges

The following is a list of tasks the scheduler is responsible for. These tasks are important for efficient execution of applications on resources.

- Resource discovery [52, 23, 63, 29]: The rule of the discovery is providing a list of resources that are available and authorised for a specific user. This list is usually obtained by searching a database which contains information about the available resources. An example of this database is Monitoring and Discovery Service(MDS) in Globus [32]. This step is also important for handling the situation of resources entering and leaving the Grid.

  In general, resource discovery mechanisms use a database (centralised approach), or a set of databases (Distributed approach) reside at different places to obtain information about available resources such as their speeds, current loads, architecture , and operating systems in order to find the resources that can meet the requirements of users. The use of a distributed discovery approach has the advantage that it is scalable, but suffers from communication cost when the databases need to be updated.

  Three schemes exist to spread the information about the current situation of resources to other entities. Table 2.1 describes the three schemes.

Universal awareness	Spreads the information to all other entities in the network.
Neighbourhood awareness	Spreads the information to all entities within a specific distance.
Distinctive awareness	The range of the information spread out is determined by the importance of the entity.

Table 2.1: Data spreading schemes in resource discovery [63]

- Resource selection: The scheduler selects a number of resources from the list provided in resource discovery step that meet the user requirements such as deadline, cost or both.

- Job mapping: The mapping function is moving the submitted jobs to the

appropriate resources to start their execution. The mapping is done based on the user requirements like time and cost.

- Job monitoring: The jobs may face unpredictable situations which need to be addressed. For that reason, job monitoring has the rule of checking how the execution of jobs is going and detecting if there is a failure or unexpected situation that needs to be addressed by the scheduler. Network Weather Service (NWS) [49] is an example of job monitoring.

However, due to the nature of Grid environment, the scheduling faces a number of challenges which make it a complicated issue. The challenges are:

- The resources are geographically distributed across the world. This raises the need of discovering the available resources on the Grid.

- The resources on the Grid are heterogeneous. They may have different architectures, speeds and operating systems. Therefore, this needs to be taken into consideration and the job has to be sent to the resource which is suitable for it.

- There are various administrative domains on the Grid, each with its local access and security policies. For that, different access policies must be supported, and the security model used in the Grid has to be mapped to the security models used in those domains [9]. Grid security is considered in [22, 80, 8].

- Due the fact that there are many entities (eg. users, resources) participating in the Grid, the security becomes an important thing to ensure the identity of the entity attempting to use a service (authentication), and if this entity has the privilege to use the service (authorisation). Securing the Grid is important to encourage both the resource providers and users to participate in the Grid, because they will know that their resources and jobs are in a safe environment, and are protected against attacks. There are two security hazards in the Grids: programs which include code that infects the resources

that run it and infected resources that crush the programs run on them [51]. Globus GSI (Grid Security Infrastructure) is a project that handles security issues, and uses public-key for authentication [37]. Resource allocation based on trust is described in [80].

- The users who submit their jobs for execution don't have control over their jobs. As a result, the completion time of their jobs can't be predicted accurately all the time, and some jobs might miss their deadlines [74].

- Grid resources are dynamic in nature. Thus, the information about resources should be updated regularly to reflect any change in their status.

2.4 Scheduling schemes

The following are the kinds of scheduling schemes used in Grids:

- Centralised schemes [61, 39, 64, 6]: In such schemes, a centralised scheduler has information about all the domains and the resources of these domains. All jobs are submitted to that centralised scheduler which in turn submits the jobs to the resources that are suitable for them based on the information it holds about the domains and their resources. None of the domains has its own scheduler, and the domains just send information regarding their resources such as resources' speeds and their availability to the scheduler to help it in making scheduling decisions. Such schemes are not scalable due to the large amount of information the centralised scheduler retains. Furthermore, the jobs that are submitted to the scheduler sometimes suffer from a long access delay when they access the scheduler, because all jobs were submitted to the same scheduler [34, 67]. It also suffers from single point of failure, because there is only one scheduler. Thus, if the scheduler fails, the interaction between the users and resource providers stops [94]. On the positive side, such schemes are easy to implement.

- Hierarchical schemes [84, 100]: Here the jobs are submitted to a central

scheduler. Then, the central scheduler forwards the jobs to the domains which meet the jobs requirements. At this point, the central scheduler has no control on those jobs. The advantage of such schemes is that each domain can employ its own scheduling policy and which can be different from other domains scheduling policies. However, the job cannot be reallocated to another resource at another domain even in the case a better resource is found.

- Distributed schemes [35, 89, 22]: There is no central scheduler in distributed schemes. Instead, each domain has its own scheduler. Each domain queries other domains either periodically or in the case an event happened to obtain information about the resources' states reside in other domains. When a job needs to be executed, it is submitted to the local scheduler which resides in the same domain. The job is then submitted to a suitable local resource or to a resource in another domain, if it is found more suitable for executing it. This scheme has many advantages like scalability [82], reliability, easily implementable, and no single point of failure. But, this scheme leads to allocation instability in some situations [67].

  It is important to have communications between the domains reside on the Grid. A heuristic scheme to allow communications between the Grid networks with bursty background traffic is proposed in [56].

  A number of distributed scheduling algorithms that are adaptable to changes in resource usage are proposed and compared to other algorithms that rely on centralised, hierarchical, and distributed schemes in [82].

2.5 Scheduling performance metrics

Every scheduling system aims to improve its scheduling performance. For evaluation purposes, each scheduling system uses a metric or a number of metrics for measuring the quality of scheduling, and which might be different from other metrics used by other scheduling systems.

The common metrics (or combination of them) used in measuring the performance are:

- Completion time [43]: It equals to the execution time plus the time needed to begin the execution of the job. Sometimes, it includes the time needed to return the results of execution to the user.

- Load balancing [25]: The scheduling quality is specified by the fairness of division of work load between various resources. Each resource gets a piece of work load depending on its specifications.

- Speedup: This is the percentage of improvement achieved when using more than one resource (processor) to execute the job, in contrast to using one resource only for executing the same job [49, 99].

- Utilisation: It can be the percentage of resources utilised or resource utilisation percentage. Percentage of resources utilised is the number of utilised resources divided by the number of free and utilised resources (all resources) at a certain time. On the other hand, resource utilisation percentage is the resource utilisation rate divided by the resource maximum allowed utilisation at a certain time.

- Broker profit [17]: This metric measures the performance of each broker based on the profit the broker obtained from acting on behalf of users.

- Resource profit [17]: The performance is measured based on the profit the resource received from executing user jobs.

- Users consumption [89, 17]: It is about decreasing the budgets paid by the users (jobs owners).

- Failure rate: It is the percentage of jobs couldn't be executed. Sometimes, it is defined as the percentage of jobs that missed their deadlines [89, 27].

- Reliability [17]: Reliability is the percentage of jobs executed successfully. It can be considered as the opposite of failure rate discussed above.

- Makespan (schedule length): The makespan is defined as the total time needed to complete the execution of all the jobs in the meta-task (set of tasks) [48, 61, 42]. This metric is used in [39, 15, 26, 45, 92, 6] to measure the scheduling performance.

- Average satisfaction rate per job [17]: This metric shows to what extent the initial deadlines of executed jobs belonging to a user were met. It is equal to the summation of differences between initial deadlines of those jobs and their completion times divided by their number.

2.6 Scheduling categories

Scheduling is divided into two main categories: conventional and economic scheduling. These scheduling categories are discussed below.

2.6.1 Conventional scheduling

Conventional (traditional) scheduling considers the overall performance of a system as a metric for determining the scheduling quality. For example, the time it takes to schedule all jobs (makespan). Additionally, it doesn't take the cost as a factor for scheduling jobs on resources.

Next, a number of projects that are based on conventional scheduling are described.

SmartNet SmartNet is a framework responsible for managing resources and jobs. It executes the user's jobs on a group of machines as if they are a single machine only. It considers many constraints when attempting to map a job to a machine like resource availability, and network speed between the job and the machine. It supports both tasks with dependency and independency. It concerns enhancing the performance of users' jobs more than enhancing the performance of machines.

SmartNet concerns two utilities in deciding the mapping performance: maximising throughput (minimising makespan) and minimising the average expected run time of each job. SmartNet uses a collection of heuristics to find the best allocation in the possible maps.

Both machine availability and heterogeneity affect scheduling decisions. Two types of heterogeneity are considered which are tasks and resources heterogeneity

This work uses a centralised scheduler which has a poor performance if there is a large number of machines, and which lacks scalability [40, 39].

The AppLeS project This project is based on agents that use application level scheduling paradigm which means that the system evaluation is based on its influence on the application.

In AppLeS, each application has an AppLeS agent that makes use of both static and dynamic information to schedule its application on a suitable resource.

The interaction within the system works as follows: First, the user sends information to the AppLeS agent. Then, the Coordinator filters possible schedules according to that information. Next, the Resource Selector determines the collection of resources that the Coordinator must consider. This is followed by computing a promising schedule for each possible resource structure by the Planner. Then, Performance Estimator is employed by the Coordinator to assess each schedule performance based on the performance objectives of the user. Finally, the Actuator selects the schedule that improves the user's performance objective.

AppLeS project employs a system called Network Weather Service for observing the performance of different resources, predicting the performance of resources in the future and sending the results of prediction to all AppLeS agents concerned in them [14].

Fully decentralised discovery in Grid environments This system concerns resource discovery in a large and dynamic collection of resources using a fully decentralised architecture. In this framework, the users submit their requests to a node which is often a local node. If this node has the resources that meet the

user needs, then it responds to the user with the matching resource information. Otherwise, it forwards the requests to another node and this node submits the requests to another node and so on. This process continues until time to live field of the request expires or suitable resources are found. In the case a node has information matching the forwarded request, it sends the information directly to the node that initiated the forwarding which in turn sends it to its user.

Four request forwarding algorithms are used: Random, Experience-based + Random, Best neighbour and Experience based + Best neighbour. Random algorithm chooses the node to which a request is forwarded at random. Experience based + Random algorithm records the requests served by other nodes, so the request is forwarded to the node that served similar requests previously. If there is no such node, then the request is forwarded to a random node. Best neighbour algorithm forwards a request to the node that served the largest number of requests neglecting the type of requests served. Finally, Experience based + Best neighbour behaves as Experienced based + Random algorithm mentioned before except in the situation where no node that served similar requests before exists. In that situation, the request is forwarded to the node that served the largest number of requests [52].

Performance prediction technology for agent-based resource management in Grid environments This work provides a dynamic resource publication and discovery using agents. In this architecture, an agent represents the local Grid resource and behaves as a service provider and service requester. The architecture uses a hierarchy organisation of agents to enable them to provide service advertisement and discovery. Agents also may work together to discover available resources. It supports quality of service (QoS) using Performance Analysis and Characterisation environment (PACE).

The task of PACE is providing quantitative predictive data to evaluate the performance of complex applications which are allocated to a local Grid resource. The agent manages its local resource using a combination of scheduling algorithm

and PACE in order to provide predictive capabilities concerning the local Grid resources and their services.

The system consists of Grid users, Grid resources, agents, and a Performance Monitor and Advisor (PMA). The Grid's users are divided into Grid service, tool developers, applications developers and Grid end users. The system concerns scientific super-computing applications. The Grid's resources might be Massive Parallel Processors (MPP), a cluster of workstations or PCs. The system uses PACE to create a hardware characterisation template in order to provide a model of each hardware resource. Agent Capability Tables (ACTs) are used by each agent to record service information belonging to other agents. They are tuples comprising an agent ID and corresponding service information. The system uses data pull (agent requesting other agents service information) and data push (submitting an agent service information to other agents) for maintaining ACT coherency.

In this system, the architecture of agent comprises three component layers. The communication layer is the first layer, and its task is performing communication functions and acting as an interface to external environment. The coordination layer is the second layer and consists of ACT manager, PACE evaluation engine, scheduler, and a matchmaker. ACT manager regulates ACT database access from agents. PACE evaluation engine is responsible for producing performance prediction information. The scheduler searches for the application that has the shortest execution time on the resource specified by an ACT item. The role of matchmaker is comparing the requirements of the request with the scheduling results which in turn affects the agent behaviour. Those components determine the behaviour of an agent upon receipt of messages from the communication layer. Finally, local management is responsible for application management, resource allocation, and resource monitoring.

The scheduling algorithm selects a group of processors to execute the application. The selected processors must lead to a minimum completion time for the application. The minimum completion time equals to the earliest start time in addition to the expected time to finish executing the application. The earliest start

time is the time when all the required processors for execution are free. However, the selected processors should meet the application's requirement which is the deadline in this case. Overall system efficiency, load balancing, service discovery speed and discovery success rate are the metrics considered to determine the system's performance [23].

Condor-G Condor-G uses mechanisms to provide security, resource discovery, and access to resources across different domains. It also provides management for computation and use of resources within a single administrative domain as well. It uses both the inter domain resource management protocols of the Globus Toolkit and the intra domain resource management functions of condor to permit the user to use multi-domain resources, so they behave as if they all belong to a single domain. In this architecture, the user defines the task it wants to execute.

This structure has the role of discovering and acquiring suitable resources, without any regard to their location. A part of its role is initiating, monitoring, managing the applications execution on the specified resources, detecting failures and taking actions against them and informing the user of application termination.

The Grid Resource Allocation and Management (GRAM) protocol is used to remotely submit the computational request to a remote computational resource and to monitor and control the computation results.

Condor-G agent uses the Grid protocols to interact with machines in order to execute user jobs on these machines. It is also responsible for the resubmission of failed jobs and recording of computation on storage to be able to restart the job if job failure happens. Its objective is to maximise the resource utilisation.

When a user requests to submit a job, a Grid-manager daemon is created by the scheduler. Grid-manager is responsible for handling all jobs that belong to a single user and terminates upon completion of those jobs. Globus Job-manager daemon resulting from Grid-manager job submission request connects to the Grid-manager using Globus Toolkit's Global Access to Secondary Storage (GASS) to transfer the job's executable and standard input files to provide real time streaming

of standard output and error. As a next step, the job manager sends the jobs to the local scheduler of the site which was chosen to execute it.

For resource discovery and scheduling purposes, Condor-G uses a resource broker that is considered as a part of the Condor-G agent to collect information about user authorisation, application requirements, and resource status in order to provide a list of suitable resources [41].

Grid Computing Pool Grid Computing Pool (GCP) concerns large scale computing devices and computing tools that are connected by high speed networks. The system's users use the Internet from anywhere to access the computing services. GCP makes use of priorities to execute some applications more quickly. The system may allocate a single computer or cluster to an application, or many supercomputers or clusters may work together to execute the application if it is computationally intensive.

The system employs the Client-Server model. The users use a client to submit their requests to the Grid. That client connects to an active node via GCP server in order to get access to computing services.

Each user must register first to be able to use Grid computing services. By doing that, the user can sign in to check the status of its submitted tasks. The system divides the applications into three layers according to their computational intensity. The server receives requests from clients and tries to allocate suitable resources to those applications. The task of server are receiving requests, sending responses to clients, sending tasks to compute engines, receiving outputs from compute engines, scheduling tasks, keeping log of status of operations and broadcasting scheduling information to other servers in the Grid.

This system has a number of servers that all have the same functions. However, only one of those servers can function at a time. The reason of having many servers is if failure occurs at a server, another server can be used. The system employs Hyper Text Transfer Protocol (HTTP) for sending and receiving messages between the client and server.

Each server includes a task scheduler that ensures mapping applications to the resources that meet their requirements. Compute engines provide the computing resources for executing different applications. Servers use Grid Computing Engine Interface to access compute engines and heterogeneous mathematical computing tools.

GCP uses Grid Software to provide interfaces for communication, resource scheduling, resource location, authentication and data access. Grid software is also responsible for building different high layer Grid computing services [96].

Advantages of Grid computing for parallel job scheduling This system is implemented to demonstrate the advantages of sharing resources belong to various autonomous domains. The system divides the Grid into autonomous sites, and each site has its resources and users. Furthermore, each site consists of a single parallel machine which is made up of nodes. Each node has its specifications such as processor speed and memory size, and is connected to other nodes via a fast interconnection network. The machines located at different sites have similar nodes, but different number of them. This system uses space-sharing allocation and doesn't take into consideration that an application can surpass its permitted time. The system has a centralised scheduler for scheduling various applications.

The system uses backfill scheduling algorithm for scheduling applications on the parallel machine. The idea of backfilling is if the available resources are not enough to execute the next application in the list, then it tries to find another application from the list that can be executed on the available resources without delaying the execution of the next application.

The system addresses three scenarios that are local job processing, job sharing and multi-site computing. In local job processing scenario, the resources belonging to one site can only be utilised by the users of the same site, so sharing between sites is prohibited. Job sharing scenario allows resources belonging to different sites to be used by the users of other sites, but doesn't allow the application to be executed on resources of multiple sites concurrently. The scheduling in job sharing

consists of two steps machine selection (selecting machines using Best-fit strategy on which the selected machine for the job must leave the least number of free resources if started), and scheduling algorithm (Backfilling). Finally, the multi-site computing scenario allows the resources of different sites to be used by the users of other sites as well as allowing the application to be executed on multiple sites concurrently. In multi-site computing scenario, the scheduler searches for a site that has enough available resources for executing the application. If no such site exists, the sites are sorted in non-increasing order according to the number of free resources they have to decrease the number of sites that their free resources are combined by the scheduler to execute the application. Then, the scheduler executes the application on a combination of free resources belonging to multiple sites. In multi-site scenario, the overhead results from bandwidth and delay is addressed by increasing the required execution time.

Five configurations were simulated. Each of those configurations consists of 512 resources. Those configurations comprise two configurations that have sites with the same number of resources, two configurations that have sites with different number of resources and one configuration that has a single site containing all resources. Four types of workloads were used each having 10000 jobs to ensure that the results are consistent.

Average Weighted Response Time (AWRT) is used to determine the scheduling quality and equals to the sum of weighted response time divided by the sum of resources utilised. A smaller AWRT means better scheduling quality.

The simulation results show that the use of job sharing and multi-site computing scenarios enhanced the performance of scheduling. Furthermore, the results show that the performance of configurations that have sites with the same number of resources is better than the performance of configurations which have sites with different number of resources [34].

Towards trust-aware resource management in Grid computing systems
In this model, the Grid is divided into Grid Domains (GDs) to facilitate the

handling of scalability, site autonomy and heterogeneity. The GDs are considered administrative entities, and each GD has a collection of resources and users which is managed by a single administrative authority. Moreover, each GD is associated with two virtual domains: resource and client domains. Resource domain (RD) represents the resources of GD, while Client Domain (CD) represents the users of the GD. GDs use trust level tables to show the trust levels (5 trust levels) between various users and resources. Those trust levels are asymmetric which means if a resource is trustable for a user, then this doesn't necessarily mean that the user is trusted by that resource.

Each CD or RD has an agent for monitoring the transactions and updating trust tables. Those agents also determine the trust between various resources and users by checking direct (via direct transactions between the user and the resource) and recommender (resource or user reputation obtained from other entities already had a transaction with it) channels.

The requests are collected first for a period of time that is set before (batch mode), and this collection of requests is called meta-request. Then, the scheduler is called to map meta-requests using a heuristic function called trust aware min-min. The trust aware min-min maps a job to the machine that provides the earliest expected execution cost considering security overhead in mapping and calculation of time needed to complete the execution of that job on that machine.

The trust model was simulated on two classes. The first class represents the simulation of homogeneous machines, while the other class represents the simulation of heterogeneous machines. Because of the use of a trust aware allocator, performance of homogeneous machines was improved by 20%, and the performance of heterogeneous machines was improved by 13%. It is supposed that the trust between two entities decreases with time if no direct transaction happened recently between them [9].

NetSolve This model is based on a client-server model, and its goal is to provide a remote access to computational resources. The resources either reside on local

or geographic network.

The interaction between entities is as follows: a client sends a request to another agent that might be in the same or different domain asking for a service that can be running a job on a resource. As a result, the agent chooses a resource that satisfies the request needs such as the size and the type of the job. NetSolve considers the load on different resources and works on balancing the load on them. Furthermore, it checks the estimated execution time of the job on every machine to decide which machine is to execute the job. The participants can access NetSolve through various interfaces such as MatLab, C, Fortran and Shell scripts [25].

Adaptive scheduling for Master-Worker applications on the computational Grid It provides an adaptive scheduling strategy that employs the master-worker approach as a middle-ware. The adaptive strategy attempts to allocate the minimum number of processors, while providing a good speedup. A good speedup is achieved by trying to keep the processors busy, and preventing them from staying idle until the work is done.

The runtime information received from the application along with the information contained in an empirical table generated by simulation are used to determine the number of processors to be allocated. Then, the scheduling algorithm adjusts the number of allocated processors, if it sees that the efficiency of the master-worker application can be enhanced without important loss in performance (speedup). Efficiency indicates how good is the utilisation of the n allocated processors [49].

A QoS guided scheduling algorithm It is an algorithm that builds on the conventional min-min algorithm [61], and it concerns both QoS and non-dedicated computing; where resources have their own tasks too that need to be executed on them. So, they are not fully dedicated to the Grid. The QoS are embedded into the algorithm to enhance the utilisation and efficiency of the Grid.

The tasks consist of tasks with or without QoS requirement. Additionally, the resources are divided into low and high QoS resources. The tasks with no

QoS requirements can run on both kinds of resources, but the tasks with QoS requirements can run only on high QoS resources. The Grid includes non-dedicated hosts, where each host has a number of computational resources that might be homogeneous or heterogeneous.

The QoS guided min-min heuristic calculates the completion time of all the tasks. Then, it maps the tasks with QoS requirements by finding the task that has the minimum completion time and maps it to the resource that has the earliest execution time for it. This continues until all the tasks with QoS requirements are mapped. Finally, the tasks without QoS requirements are mapped in the same way the tasks with QoS requirements were mapped.

It is shown that their QoS guided min-min heuristic surpassed the conventional min-min heuristic in most cases, and tolerates imprecise measures of execution times. The utility used to determine the performance of various heuristics is minimising makespan (maximising throughput) [48].

Reinforcement learning (RL) algorithm This work employs multi-agent learning techniques to solve the resource allocation dilemma. It uses RL to overcome that dilemma without having global knowledge on the status of resources in the system such as the load levels, and without any interaction between agents.

The agents that represent users select the resources with minimum queue length and that can minimise the completion time of the jobs. In the RL algorithm, each agent has a value named Q-value for each resource that reflects the resource efficiency up to now for this agent. Upon completion of each job, the Q-value is updated . For new jobs, agents select resources according to the $\in$ greedy rule. So, it selects the resource with the highest Q-value, while with considering probability $\in$, the agent chooses among the other resources randomly.

RL algorithm was compared with two other algorithms. The random selection is the first algorithm RL algorithm was compared to. In this algorithm, agents select resources randomly using uniform probability. The second one is least load algorithm, where agents select the resource, which has minimum load. It is shown

that the RL algorithm outperforms the two algorithms in most cases. However, random selection surpasses the RL algorithm in the case the job arrival rate is low.

The agents that enter the Grid must start the learning process, which takes some time before they can behave sufficiently [43].

Scheduling Co-Reservations with Priorities in Grid Computing Systems Three algorithms that support co-reservations (incorporate many resources) are proposed for scheduling jobs on CPUs. The first algorithm is the Co-reservation scheduler with Priorities and Benefit functions (Co-RSPB) which takes into account priorities of different requests when scheduling jobs on the available CPUs, and where each request has a benefit function that specifies the utility obtained by the user. In Co-RSPB, the requests in meta-request are sorted in non-increasing due to their priorities. However, if a request consists of sub-request, they are sorted in non-increasing order according to the minimum CPU requirement. The Co-reservation scheduler with Best Fit Scheme (Co-RSBF) is the second algorithm, and differs from Co-RSPB in that it sorts requests according to the sum of CPU requirements of their sub-requests. The last algorithm is Co-reservation scheduler with Best Fit and Refining that employs Co-RSBF, but differs in that it attempts to improve the reservation by supplying more benefit to the accepted requests for reservation.

Two utilities are used to describe the performance of the algorithms: system benefit and number of requests rejected. The simulation results show that Co-RSPB surpasses the other algorithms in term of maximising the system benefit, because it considers requests priorities. On the other hand, Co-RSBF outperforms the others in term of minimising the requests rejected [65].

Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems In this work, a group of resources is connected by a LAN, and the LANs are connected by a WAN. The tasks considered here are tasks that have subtasks which communicate with each other. The subtasks of a task can be ex-

ecuted within a single LAN only. However, the allocated resources must satisfy the requirements of tasks which are here not exceeding their deadlines.

There are two levels of scheduling. The first level occurs at the WAN that uses a distributed scheduler (external) to choose a LAN which part or all of its resources will execute the subtasks of a task. The second level occurs at the LAN and uses a centralised scheduler (internal) to choose a number of resources that reside on the LAN to execute the task's subtasks. The external scheduler is responsible for receiving transmitted tasks, sending bid requests (sealed-bid auctioning) to internal schedulers, choosing an internal scheduler according to the received bids, coordinating the execution of subtasks and collecting results. The external scheduler selects the internal scheduler that submitted a bid that offers an estimated task response time (task completion time minus task transmission time) less than the estimated task response time in the bids of the other internal schedulers. On the other hand, the internal scheduler receives tasks from the external scheduler in order to execute their subtasks on a number of available resources, then sends the results of subtasks to the external scheduler. The internal scheduler selects the machines that provide the minimum expected subtask response time (time of returning the subtask's results minus the time when the subtask was initiated).

The Grid's performance in the simulation is measured by two metrics: percentage of tasks that satisfied their deadlines and average task response time [27].

RR scheduling algorithm RR [42] is a scheduling algorithm that doesn't require any predicted information about the characteristics of available resources like their speeds. RR allocates one of the free processors to one of the available tasks until all processors are allocated or there are no tasks to get executed. At this point, if there is still a number of non-mapped tasks, it waits until one of the processors becomes free, then it allocates that processor to one of the remaining tasks and continues this process until all the tasks are mapped. When all tasks are mapped and a number of processors become free, replicas of one or more tasks still in execution can be assigned to the processors that became free, because the

execution of the instance may finish before the execution of the original mapped task. However, when the execution of an instance of the task gets finished, the algorithm's role is to delete the other instances of the task.

This work considers that some tasks might have instances that run on a number of processors at the same time. This is inefficient and considered a waste of resources. This becomes more inefficient in the case of using an economic approach, because the task's owner must pay for executing each of the instances.

K-windows scheduling algorithm A scheduling algorithm which uses heuristics is proposed in [81]. It is called K-windows and aims to search for the best cluster to execute a job. K is a random value which is selected from a range of values and it takes a value between 1 and n. n is the number of machines in the cluster which contains the smallest number of machines in the Grid.

In order to find the most appropriate cluster to execute a job, the algorithm first gets information about average CPU load, memory usage and task remaining. Then, it selects a suitable cluster based on the information above. In the process of selecting a cluster to execute a job, CPU load comes first as a constraint for this selection, then memory usage, and finally task remaining.

The algorithm was compared with two other simple algorithms (Random and Round-Robin algorithms). The comparison was done on ThaiGrid system which is a Grid infrastructure residing in Thailand. In this comparison, there are ten tasks, each task is a two arrays multiplication of sizes 100*100 and 1000*1000. The K-windows algorithm surpassed both Random and Round-Robin algorithms.

2.6.2 Economic scheduling

The study of using economics to manage the computing resources returns to 60s and 70s [87]. In economic scheduling, cost is considered an essential factor for scheduling jobs. For a scheduling to take place, the parties must positively value this scheduling [67].

It is believed that the use of commercial private Grid will be first employed

between companies before it starts to be used by the public [78]. The economic problem results from having different ways for using the available resource, so how to decide what is the best way to use them [67].

It is mentioned in [47] that the participation in the Grid must be cheaper for the users than purchasing their own resources, and must satisfy their requirements. On the other hand, resource providers must know if it is worth to provide their resources for usage by users.

It is mentioned in [46] that pricing of resources should not be per time unit or slot (e.g. cost per minute). This is due the big differences in speeds of processors. So, the price per time unit of a processor might be cheap, but the user might pay large amount of money, because it is slow. Furthermore, the users have to know how many time units they need to finish their jobs. Thus, the cost must be determined based on the tasks the resources perform.

Some of the advantages of using economics (prices) to decide what resource to select are described in table 2.2.

Flexibility	Resources can be used by the users just when they need them because this use is not for free, and the users are charged for this use.
Efficiency	Resource cost expresses how much a resource is important to a user, or in other words: it shows the resource value.
Scalability	Economics facilitate handling the situation of users and resources entering and leaving the Grid.
Feedback	The costs charged for the use of resources might be used as a history to enhance the allocation policies in the future

Table 2.2: Advantages of considering economics in selecting resources [55]

The next sections describe the projects that are based on economic scheduling.

Deadline scheduling algorithm In [89], a deadline scheduling algorithm that supports load correction and fallback mechanisms to improve the algorithm's performance in an environment that is based on client-server model is proposed. It submits each job to a resource that can finish it in time less than or equal than the time specified for completing the job. However, if no such resource is found,

it selects the resource which can finish the job in a time very close to the deadline of that job but still beyond the deadline. Fallback is a notification from a server to a client to inform the client to resubmit its job again to the system, because the server found that it can't meet the deadline of the client's job. There is an upper limit to the number of attempts to submit the job again to the system. On the other hand, load correction is used to overcome the problem of sending all the jobs to the fastest resource by considering the number of jobs that were submitted to resources for execution and have not completed yet.

The project's objective is to decrease the number of jobs that don't meet their deadlines. The resources are priced according to their performance. It uses bricks simulator which is written in Java after extending it to assess the performance of the implemented deadline algorithm.

The deadline algorithm is compared to a greedy algorithm, and it is stated that the greedy algorithm results in higher cost than the deadline algorithm.

They show that fallback mechanism has led to a large improvement in reducing the failure rate, while load correction mechanism has just led to a slight improvement in reducing the same rate.

Nimrod/G Nimrod/G [19, 18] uses a component based architecture. The system architecture consists of five components that are client or user station, parametric engine, scheduler, dispatcher and job wrapper.

Client or user station is a user-interface for controlling the application of the user. Using this interface, the user can change the application requirements such as execution time, cost and deadline. This change leads to allocating the resources that fit the user requirements. The second task is monitoring the status of all the jobs which are under the control of this user. Finally, it is responsible for running multiple instances of the client on different machines.

Parametric engine is a central manager agent that is responsible for managing and maintaining the whole application in addition to the creation of jobs, job maintenance, interacting with users, advising the schedule, parameterisation of

the experiment and dispatching.

Resource discovery, resource selection and job assignment are the roles of the scheduler. In resource discovery stage, the resource discovery algorithm contacts with the Grid information service directory to identify a list of authorised machines and tracking resource status information. The resource selection algorithm has the task of selecting the resources that can meet the job's deadline and at the same time reduce the cost of the computation.

The dispatcher is responsible for starting the execution of an application on a suitable resource. It also periodically interacts with the parametric engine to update the status of application execution.

The job wrapper starts the application execution on the resource, then submits the results back to the parametric engine via dispatcher. It also acts as a mediator between the parametric engine and the resource on which the application is running.

The most important user requirements parameters in this architecture are the price the user wants to pay and deadline.

This architecture uses Globus toolkit services. Additionally, it can be extended to work with any emerging Grid middle-ware service. Nimrod/G employs the history of execution times and resource costs to schedule the submitted tasks, while meeting the time and cost requirements of the user.

Nimrod/G includes four scheduling algorithms which are cost, time, conservative time and cost-time. Cost scheduling algorithm tries to decrease the amount of money paid for executing the jobs as much as possible while meeting their deadlines. Time scheduling algorithm attempts to minimise the time required to complete the jobs while meeting their budget. The conservative time scheduling algorithm aims to execute the jobs while meeting their budget and deadlines. Additionally, it minimises the time needed to execute them, when higher budget is available. Finally, cost-time scheduling algorithm works as cost scheduling algorithm except that when there are two or more resources with the same price, it employs time scheduling algorithm while scheduling jobs on them.

Market-based Resource Allocation for Grid Computing This model [44] acts as a marketplace where computational resources are traded. It consists of clients (users), servers (resource providers) and electronic marketplace.

This model supports time and space shared allocations. Furthermore, it supports co-allocation. It is supposed that resources have background load which changes with time and that has the highest priority for execution, so they are not fully dedicated to the Grid.

This model uses two kinds of scheduling policies. In the first scheduling policy, all the resource units of the server are allocated to the arriving task and the number of allocated resource units varies according to the change in the background load. In the second scheduling policy, the arrived task shares the resource with the other tasks already executing on the resource (proportional allocation), and the resource percentage utilised by the arrived task changes according to the number of other tasks using the same resource.

The model compares three allocation algorithms: Round-Robin (conventional strategy), continuous double auctions (economic strategy) and proportional share (economic strategy). Furthermore, the model considers two categories of tasks: tasks with the same priority and tasks with different priorities. For tasks with the same priority, completion time of all tasks (makespan) is the metric considered to measure the scheduling performance of the three allocation algorithms. On the other side, the Weighted Completion Time (WCT) is the metric for measuring the performance of allocation algorithms in the scenario of tasks with different priorities. WCT is equal to the mean of task weights multiplied by their completion times.

According to the simulation results, the three allocation protocols have the same performance when the load on resources is low. But, the gap between the performance of the three allocation algorithms becomes bigger when the load increases. In this situation, the continuous double auction algorithm has the best performance followed by proportional share. However, when the resource heterogeneity is increased, proportional share will have the best performance, where

Round-Robin will be the worst one.

Computational Communities In this work, a Grid middleware that eases the mapping of jobs to the available resources, taking into account the requirements of users that should be satisfied is defined. Resource providers and users use a computational currency.

The described model consists of organisations which supply their resources to be used by users to gain money from that use, users who utilise the resources, mappers that suggest a set of execution plans that suit job's requirements and brokers that are responsible for negotiations between resource providers and users. The above components communicate with each other via a public resource marketplace.

The resources are maintained through local administrative domain and made available to users through the domain manager which is responsible for publishing the information about resources in one or more public marketplaces and maintaining the local access control policy.

In this model, the same resource can be advertised in different marketplaces, and this is true for users who can send their requests to various marketplaces. The project considers three types of resources: computational, storage and software. The resources are advertised as Java objects.

In order to allocate a resource to a job, the application mapper first provides a collection of suggested execution plans. Secondly, the broker contacts the resource providers to negotiate the price of various suggested execution plans. Finally, the brokers send the prices to the user who in turn chooses what satisfies him [68].

Market-based proportional resource sharing for clusters The architecture provides a decentralised economic resource management on clusters. It employs proportional share schedulers. So, each job gets a percentage of the resource depending on the number of jobs already executing on the same resource.

In this architecture, the cluster nodes act as sellers and the users act as buyers. It consists of five layers that are resources, resource managers, economic front end, access modules and end users. Resources are the computational resources that are

allocated to applications of users. Resource managers are considered the operating system entities that are responsible for resource allocations. The economic front end maps value to resource allocations. Access modules are used by users to access resources to run their applications. End users send their applications to run on cluster resources [28].

Parallel Virtual Machine (PVM) PVM enables the computational resources to be used as if they are a single high performance machine. It involves system level daemons that afford services to local and remote jobs running on resources.

PVM includes a library of interface routines that allow access to the system. PVM supports both execution on single and multiple resources by splitting the task into subtasks.

A cost matrix which is specified by a user determines the cost of running each subtask on each resource. It includes parameters such as expected execution time and utilisation cost in the currency used in the system. The cost matrix is used later to select the machines for executing the submitted subtasks. The subtasks interact by sending parameter values. For every subtask, its parameter values are received from predecessor subtasks if there are such predecessors [11, 86].

Scheduling under uncertainty This work concerns executing jobs that consist of multiple services, while satisfying the requirements of them that comprise time, cost and minimum percentage of surety. The surety means the maximum risk the user permits in meeting its budget. The jobs are monitored, and surety is calculated regularly to check that it is still greater than or equal to the minimum percentage of surety specified by user. If the surety percentage is less than the percentage specified by user, a recovery action is taken by the scheduler, and surety is increased to a level that is greater than or equal to the user's minimum level. For example, this can be done by replacing services offering the execution of the critical path (the path with the longest execution time) with faster services.

The scheduler can select slower services and which are cheaper to replace some of the services not along the critical path to save part of the budget. This saving

can be used later for recovery, if something unexpected happened.

The service providers submit their bids to a user who uses Pareto optimality to decide which bids to accept. Pareto optimality accepts a bid, when no other bids with absolute advantage in terms of time, price and surety are found [74].

Economic scheduling in Grid computing This system comprises multiple domains (Metadomains) and employs the auction model. Each domain has a set of local resources and local management instances (Metamanagers). Metamanagers control those metadomains, manage the local resources and respond to requests from users. Each metamanager includes a local scheduler that behaves as a broker to other metadomains.

For discovery purposes, the metamanager contacts the directory services (contains information about resources), in addition to exploring the neighbourhood. Each metamanager retains a list of links to other metamanagers.

It is stated that this structure increases reliability as well as preventing single point of failure, because a failure at a domain has an effect on that domain only.

The utility function used by a user can be minimising the start time of the job, minimising the computation cost or both. The metamanager chooses the offer with the highest utility value. The system supports both automatic selection and manual selection by the user. The user account is checked to ensure that there is enough credit to pay for the job to be run.

If no enough local resources are found to run a job, multi-site scheduling is selected in order to obtain the number of resources required. As a part of multi-site scheduling, the scheduler attempts to find all free time intervals of the suitable resources within the requested time and combine them in different ways. This usually produces a large number of combinations. Thus, the combination with the highest utility value is selected.

Average Weighted Response Time (AWRT) is used to determine the scheduling quality and equals to the sum of weighted response time divided by the sum of resources utilised. A smaller AWRT means better scheduling quality. The simula-

tion results show that the economic scheduling surpasses the conventional (FCFS, backfilling) scheduling for all workloads and resource configurations. It also shows that the utility function that considers only minimising the job start time has achieved the best performance in contrast to the other used utility functions [35].

Compute Power Market Compute Power Market is an architecture responsible for managing Grid resources, and mapping jobs to suitable resources according to the utility functions used. It comprises buyers, sellers and markets. The interaction happens between Market Resource Agent and Market Resource Broker that acts on behalf of its user. The Market Resource Agent which is downloaded from Market Resource Agent Download unit provides the market with current status of the resource, and it is also responsible for accepting and rejecting jobs requesting execution on its resource. The Market Resource Broker which is downloaded from the Market Resource Broker Download Unit explores the market in order to find a suitable resource to execute the job submitted to it.

This market supports commodity, auction and contract-net models. The buyers and sellers use utility functions to describe their interests. For instance, buyers might want to minimise the cost of executing their jobs, while the sellers want to maximise their revenue. The market has a database called market entry index to keep information regarding various users and providers in the market, and it also contains references to other markets. The information about providers is refreshed by and update unit, so the information about the providers remains up-to-date. However, the user can obtain information about the available resources on the Grid by contacting market information services.

In this architecture, the markets can get a percentage of the resource providers (sellers) revenue in return for the services they provide [22].

GridWay In GridWay [66], the user submits its jobs to the Grid through a command-line interface. Each user has a broker called Submission agent that is responsible for discovering available resources, monitoring user's jobs and submission. The Submission agent aims to maximise the optimisation criterion specified

by the user it acts on behalf. The optimisation criterias are Performance, 1/CPU-Price-per-second and Performance/CPU-Price-per-second that aim to minimise the execution time, total cost and the performance to cost ratio of the submitted job, respectively.

The GridWay uses two object classes (MdsEconomicInfo and MdsCpuPrice) to deal with economic information. For example, the CPU price information generated by the resource provider measured by Grid currency unit is contained in an attribute called Mds-Cpu-Price-Per-Second.

An evaluation of communication demand of auction protocols in Grid environments In [7], a framework that is based on reverse auction (tendering) is presented. It comprises users (buyers), brokers (auctioneers) and resources (sellers). In this framework, the user first submits jobs to its broker that is responsible for submitting and monitoring them. Then, the broker starts an auction and sets its parameters (e.g. auctions' number of rounds and reserve price). Then, the broker submits a request for all resources. Next, the resources decide to bid or not based on the initial price of the job. If there is a bidder, the broker clears the auction and sends the result of the auction to the participants. Otherwise, it increases the price and sends new requests to the resources. This continues until there is a bidder.

Amazon Elastic Compute Cloud Amazon Elastic Compute Cloud (Amazon EC2) [3] allows users to run their jobs/applications on computing resources. The users use either a web interface or a command line to control their computing resources (instances). Each instance supplies what equals to 1.7 GHz Xeon CPU, 1.75 GB of memory, 160 GB of storage and 250 Mb/s of bandwidth. In order to use Amazon EC2, the user has first to create an Amazon Machine Image (AMI) which includes applications, libraries, data and configuration settings. However, the user can also select from a number of AMIs provided by Amazon EC2, if it doesn't want to setup its AMI from scratch. Then, the user uploads its image into Amazon Simple Storage Service (Amazon S3) which is a repository for storing

images. Next, the user has to setup network access and security. Finally, the user uses Amazon EC2 web service to manage instances of its AMI. The user is charged for the instances (per hour) it uses, and for the bandwidth (per GB) it consumes. But, there is no charge for data transferred within Amazon EC2, and between Amazon EC2 and Amazon S3.

Market economy based resource allocation in Grids In this master thesis [73], six auction based user scheduling policies (strategies) for selecting resources were proposed: TimeOptimized, BudgetOptimized, NewTimeOptimized, NewBudgetOptimized, Combined and History based. TimeOptimized and BudgetOptimized improve average turnaround time and average budget per job, respectively. NewTimeOptimized and NewBudgetOptimized are enhancements for TimeOptimized and BudgetOptimized, respectively and they consider the success rate as well. The success rate is the number of jobs that met their deadlines. NewTimeOptimized tradeoffs between average turnaround time and and success rate, whilst NewBudgetOptimized tradeoffs between average budget per job and success rate. They both use a parameter named k that takes value between 1 and n. If k=1, NewTimeOptimized and NewBudgetOptimized are going to be the same as TimeOptimized and BudgetOptimized, respectively. As k increases, both average turnaround time and success rate increase for users employ NewTimeOptimized, while both average budget per job and success rate increase for users employ NewBudgetOptimized. Combined policy decreases the average turnaround time and average budget spent per job based on user preference. Finally, History based policy uses previous success rate for selecting between NewTimeOptimized and NewBudgetOptimized.

GridIS In [97], a P2P decentralised framework for economic scheduling using tender model was presented. Additionally, a scheduling algorithm for resource providers was proposed. In order to reduce the number of jobs that don't meet their deadlines, conservative degree (CD) that takes a decimal value between 0 and 1 is used by the algorithm. When CD equals to 0, the algorithm is going to

be aggressive and accepts all jobs and therefore jobs might miss their deadlines. As CD increases, the algorithm is going to be more conservative which means it is going to keep the unconfirmed jobs in its job queue for consideration for some time. The algorithm adds unconfirmed jobs to its job queue at the probability of its CD. In this work, CD was varied to see its effect on both failure rate and deadline miss rate. When both system load and CD are low, none of the jobs failed to execute or to meet their deadlines. However, when the system load is increased, deadline miss rate is increased too, but failure rate hasn't been affected. When CD is high, deadline miss rate is zero. But, failure rate increases when system load increases.

2.6.3 Differences between conventional and economic strategies

The major differences between conventional and economic scheduling can be summarised in two points:

- Conventional scheduling usually considers the overall system performance to determine the scheduling quality such as makespan, reliability and utilisation rate, while economic scheduling concerns improving the value gained by each entity separately, and which is usually represented by utility functions.

- Conventional scheduling doesn't take into account the price of resources when allocating resources to the submitted jobs, whilst cost is considered important in economic scheduling, and affects resource allocation decisions. Due to the reason that conventional scheduling doesn't concern access cost to resources, each user attempts to submit its job to the best resource on the Grid leading to overloading the high performance resource, while leaving the other resources underutilised (load balancing problem). On the other hand, economic scheduling concerns the costs of resources, so each user submits its job to the resource that just satisfies his requirements, and not necessary to be the best resource on the Grid, because the submission of the job to a

high performance resource means higher cost to the user.

2.7 Market models

A Market is defined in [91] as a context where the selling and buying of goods and services occur. The markets were began to be used in Grids in the 1980s due to the enhancement and popularity of computer networks [67]. The employment of markets in Grids leads to some advantages such as simplicity, flexibility, efficiency, scalability and dynamic adjustability [67]. Employing market models gives an incentive for resource providers to supply their resources, because they want to get some profit from sharing their resources [93].

In the computational Grid that employs economic strategies, the resource owners provide their resources to be utilised by users to gain profit from this utilisation. However, the job owners and resource providers need to deal with the way the resource owners specify the price of their resources, and how they charge for the resource consumption. This is determined by the market model that is used in the Grid.

In the next sections, six market models are discussed.

Commodity market model In commodity model [18, 54, 20, 95, 94, 22, 84, 4], the resource owners determine a price for the usage of their resources, and charge according to the percentage and duration of usage. If the pricing policy used is fixed, then the resource price will be stable all the time and will not be affected by supply and demand. Otherwise, the resource price will fluctuate through time (eg. off-peak and peak times) and will be affected by supply and demand.

In this model, the scheduler/broker usually contacts a price database which contains the prices of resources to get a list of prices. Sometimes, the resource owners charge different prices for different users. The commodity (the use of a resource for a specific time) can be sold for future use [53]. The dilemma in the commodity model is in giving resources prices that reflect their values accurately.

Bargaining model In this model [4], the users and resource owners negotiate the price of executing the users jobs on resources. The negotiation process can lead either to an acceptable price for both owner and user, or to a rejection from the owner, in the case it hasn't accepted the final price and the price could not be negotiated anymore.

Tender/Contract-net model In tender model [22, 17], the scheduler/broker issues a request for job execution with specific requirements. Then, the various resource owners send their bids to the scheduler which selects the most appropriate bid and contacts that resource owner in order to start the execution.

Auction model It can be said that auction model [18, 20, 95, 94, 90, 22, 70] is the opposite of tender model. In tender model, the resource owners send their bids for executing a job, while in auction model the job owners send their bids to the resource owner. Thus, in tendering, the scheduler/broker selects a bid from a number of resources' bids, but in auction model each user attempts to put a higher bid in order to gain that resource, so it can execute its job.

In some projects, the user can bid on a service that consists from a single resource only, but some projects discuss bidding on a service that consists of more than one resource and that can be a set of storage, CPU and network resources [75, 71, 31, 90, 76]. The resource owner can define some requirements for the auction such as the starting price, expiry time of auction and reserve price [18].

The drawback of this model is that the users spend a lot of time bidding on services, especially in the case when each service consists of a bundle of resources, and which is considered a NP (non-deterministic polynomial time) problem in [77]. Another drawback is the overhead resulted from negotiations between the parties involved in the auction. However, auction model is easy to be understood and implemented and has well known economic characteristics [94, 30].

There are five types of auctions:

- English (ascending-price) auction: Seller declares low opening bid. Then, bidding increases continuously until no bidder wants to increase the price,

and the winning bidder pays the highest bid value. English auction is considered an open auction in which each bidder knows others bids.

- First price sealed bid auction [27]: Each user sends one bid without any knowledge about others bids. The auction is awarded to the user with the highest bid.

- Vickery/Second price sealed bid Auction: Similar to the previous type of auction, bid is awarded to the user with the highest bid, but it pays an amount of money equals to the second highest bid.

- Dutch (descending-price) auction: The auction starts at a high price and drops continuously until it reaches a price that is accepted by one of the bidders. The bidders know the bids of the others as in English auction [5].

- Double Auction [44]: It is a system where buyers enter bids, and sellers enter offers simultaneously. The task of the auctioneer here is matching between bids and offers that are suitable for each other. This model is already deployed in NYSE (New York Stock Exchange) [1] and AMEX (American Stock Exchange) [2] markets.

A comparison between commodity market and auction models was made in [95, 94]. Two types of resource providers were considered: CPUs and disk storage providers.

Efficiency is used to determine which of the two markets behaves better. The resource efficiency is measured by calculating the average percentage of time each resource stayed utilised. The user efficiency is measured by computing the average number of jobs completed for each user. It is stated that in general commodity markets outperform auction markets.

However, some of the assumptions about the behaviour of buyers and sellers in this work don't reflect the behaviour of real buyers and sellers (e.g. having cooperation between service providers).

Proportional resource sharing model In this model [44, 79], the users have credits that are used to access resources. The percentage of a resource the user can utilise is equivalent to the number of credits it has in contrast to the number of credits the others have. For example, If a resource is represented by N credits, the user who has n credits can utilise n/N of the resource.

Cooperative bartering model In this model, each entity participates in a sharing environment, so it provides resources to other entities and uses the resources of other entities on the Grid. Thus, each entity behaves as a buyer and a seller at the same time. Each entity gets a number of credits proportional to the number and quality of resources it provides. Those credits are used to access others' resources [18].

2.8 Critique

Section 2.6 has described conventional and economic scheduling including the projects that are based on. However, those projects have some limitations that need to be considered. In conventional scheduling, there is no motivation for resource providers to share their resources, because they won't get profit in return. Furthermore, employing conventional scheduling results in having utilisation or load balancing problem, because all users will execute their jobs on the resources with high performance. As a result, those resources will be highly utilised all the time, while the other resources will be left free or underutilised. For those reasons, conventional scheduling is not suitable for Grid scheduling. On the other hand, economic scheduling gives an incentive for resource suppliers to provide their resources, because they will obtain profit from the users' usage. Moreover, the job owners use the resources only when they need them, since they have to pay for the utilisation.

However, the projects that employ economic scheduling have some gaps. Firstly, some of them employ centralised scheduler [19] which is known to have serious drawbacks (see section 2.4). Secondly, they usually consider proposing economic

scheduling strategies for users [19] or resources [97], but not both. Thirdly, the brokers don't receive profit from acting on behalf of users, which is unrealistic. So, there is no incentive for brokers to participate. Finally and which is the most important, only few projects [22, 7, 97] consider economic scheduling within an environment that uses tender model, in spite of its suitability for Grid computing. Tendering is supported in [22], but it is not described how the entities interact using it. In [7], the brokers don't receive profit in return of acting on behalf of users which is unrealistic as mentioned above. Additionally, deadline satisfaction is not considered in evaluating the scheduling performance. In [97], the brokers don't receive profit from acting on behalf of users too, and cost is not considered in evaluating the users' performance.

The aim of this thesis is to fill those gaps by proposing a new framework for decentralised economic scheduling in Grid computing using tendering. In this framework, all participating entities such as users, brokers and resources have utilities that need to be maximised by entities' strategies. Within the framework, the performance of each strategy is compared to the performance of other strategies employed by entities of the same class.

For metrics to measure the scheduling performance described in section 2.5, load balancing is not important in economic scheduling, because each entity behaves in a selfish manner and doesn't care about the others. Since executing a job on more than one processor in not in the scope of this project, speedup cannot be used. Utilisation doesn't reflect the real performance in economic scheduling, because a resource can have a high utilisation rate, but generate a small profit due to the execution of cheap jobs. However, another resource might be less utilised, but generates more profit due to the execution of expensive jobs. Reliability is the same as job success rate, and failure rate is just an opposite to reliability. Makespan is appropriate for centralised scheduling. However, the jobs of different users might arrive at different times in decentralised scheduling, so makespan can't even be used to measure the performance of different users. Thus, average satisfaction rate per job has to be used instead and it determines to what degree

the user (initial) deadlines were satisfied.

Eventually, the application (job) requirements (see section 2.2) that the user needs to specify in its request are budget (price), deadline and job class. However, the user doesn't have to know the resource specifications required for running its job such as CPU speed, physical memory and file size, because this is the role of the resource. After the resource receives a request for executing a job, it calculates the expected completion time based on its specifications and status, and then decides if it can fulfil the user's requirements specified in the request. Finally, bandwidth and security can be part of the job class, because the job class determines the resources the job requires when running and which can be physical or virtual resources.

2.9 Chapter summary

In this chapter, an overview of scheduling in Grid computing has been given. The overview begins with describing the Grid components. Then, common application requirements have been mentioned. Next, resource scheduling tasks and challenges have been explained. This is followed by describing the scheduling schemes. Then, scheduling performance metrics have been mentioned. Next, scheduling categories like conventional and economic scheduling have been described including the related projects. This overview ends by describing the market models. After the overview, a critique has been presented.

The survey undertaken in this chapter shows that the current projects which consider scheduling in Grid computing have some limitations. In the next chapter, a new framework that overcomes the limitations of these projects is introduced.

Chapter 3

An economic scheduling framework using tendering

3.1 Introduction

In this chapter, a new framework that supports economic scheduling in Grid computing is proposed. The users, brokers and resources employ tender model to negotiate the prices and deadlines. Through negotiations, users, brokers and resources aim to maximise their utilities.

In Grid computing, resource providers supply their processing resource to the Grid, and jobs are allocated to these resources for their execution. The aim of resource providers is to make profit, and the aim of users is to reduce cost. This situation creates an economic market.

Much work [89, 19, 18, 44, 66, 73, 97] was done on finding an optimal allocation of resources to users' jobs. Some of the projects use conventional scheduling that usually concerns the system performance in general and don't consider the economics (prices) for allocating jobs to resources. On the other hand, a large number of projects consider economic scheduling for allocating jobs. Economic scheduling takes into consideration the price of the resources when it needs to allocate jobs to resources and that price usually reflects the value of the resource to the

user. Furthermore, economic scheduling gives an incentive for resource providers to supply their resources, and enforce the job owners to make cost-effective use of the resources.

The competing aims of maximising profit and reducing cost lead to negotiations of price. This is determined by the economic model in use. Some of the models that are applicable to Grid environment are auction [18, 20, 95, 94, 90, 22, 70], commodity [18, 54, 20, 95, 94, 22, 84, 4] and tender/contract-net [22, 17] models.

The tender model is discussed in this chapter's proposed framework. In this model, the user issues a request for job execution with specific requirements and which is submitted to a group of brokers. Then, the various resources respond with bids to brokers. The user then selects the most appropriate bid based on the user's strategy, and contacts that broker. Finally, that broker contacts the selected resource in order to start the execution. The Tender model is an appropriate model to employ in the Grid, because users begin the negotiation. Additionally, the users specify their job requirements with their requests and the resources reply with bids that are based the the cost of taking on the job and the availability of their processors.

The rest of this chapter is organised as follows: Section 3.2 introduces the framework's entities. The strategies that are employed by those entities are presented in section 3.3. Finally, Section 3.4 summarises the chapter.

3.2 Framework entities

The framework consists of a number of entities that use tender model. The entities are classified into resources, brokers and users. A resource sends bids and receives jobs, a broker sends jobs and bids and receives jobs and bids and a user sends jobs and receives bids. Figure 3.1 shows the negotiations that take place between the entities, while Table 3.1 shows the parameters which are sent through the negotiations that occur between them. For each negotiation, the job parameters such as job characteristics and requirements are sent from the user to brokers. Then,

each of these brokers can either negotiate with the user or negotiate a group of resources. After the broker sends offers to resources, the interested resources send their bids to this broker. Next, the brokers keep negotiating with the resources or with the user who in turn accepts one of the sent bids from brokers or keeps negotiating.

The job parameters are:

- User number: The job owner's number.
- Job ID: The job's number.
- Sender number: The number of the entity that submitted the job parameters for the job whose number is Job ID.
- Receiver number: The number of the entity that receives the job parameters.
- Negotiation number: This shows the negotiation's number for the submitted job. A negotiation is a direct communication between two entities (e.g. communication between a user and a broker).
- Price, job length, deadline and job class that are described in section 3.2.2.

On the other hand, the bid parameters are:

- User number: The job owner's number.
- Job ID: The job's number.
- Sender number is the number of the entity that submitted the parameters.
- Receiver number is the number of the entity that receives the bid parameters for the job whose number is Job ID.
- Negotiation number: It shows for what negotiation's number this reply (bid) is for.
- Price: The revenue that is asked for executing the job.

- Expected completion time: The date by which the job is expected to complete.
- Expiry date: The date when the bid will expire.

Grid accounting is not in the scope of this project. Grid accounting is considered in [10, 30].

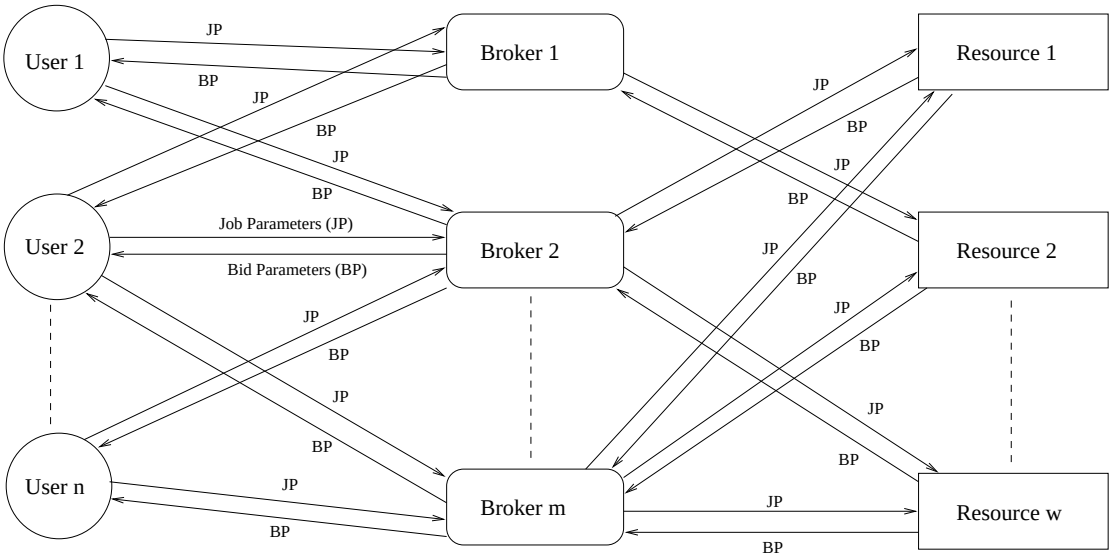

Figure 3.1: Interaction between users, brokers and resources.

Table 3.1: Negotiation parameters.

Job parameters	Bid parameters
User number	User number
Job ID	Job ID
Sender number	Sender number
Receiver number	Receiver number
Job length	Negotiation number
Job class	Price
Negotiation number	Expected completion time
Price	Expiry date
Deadline	

3.2.1 Resources

The resources represent the working power of the grid, and they are where the jobs are executed. Each resource comprises a number of processors that have speeds

measured in MIPS (Million Instructions Per Second). MIPS is used because no better metric was found. In the future, a more appropriate metric to measure the speed of processors is going to be used. In this work, all the resources (processors) use space shared scheduling policy, i.e. a processor can execute only one job at a time.

In the proposed framework, the resource is shown as an entity that sends bids and receives jobs as appears in figure 3.2. The resource receives jobs (requests) from brokers. After processing the requests, it sends bids based on the cost of taking on the jobs and the availability of its processors.

This framework supports the situation where resource prices fluctuate based on the demand and supply. Thus, when the demand becomes higher, the resource becomes more expensive and the chance of the broker request to be accepted becomes smaller. Otherwise, the resource becomes cheaper and the chance becomes greater. On the other hand, when the supply (free processors) is higher, the resource price decreases and the possibility of the resource to accept a job request becomes higher. Contrarily, the price increases and the possibility becomes lower, if the supply is lower. Different models can be implemented to determine how the resource price is varied and how the demand is measured.

Two kinds of costs are paid by resources: Cost per million instructions (MI) and cost per time unit. Cost per MI is only paid when a resource is executing a job, because this results in consuming more power in addition to other expenses relate to the job executing. On the other hand, cost per time unit is paid all the time whether the resource is idle or busy and results from the expenses arisen from maintaining the resource (e.g. security, software).

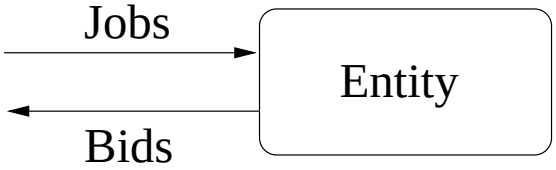

Figure 3.2: Resource.

3.2.2 Users

The users have jobs that need to be executed on the resources reside on the Grid. Each job has a price (currency), length (MI), deadline (date) and a class. Price is the amount of money that will be paid for executing the job. The price that is submitted to the brokers might be increased during the negotiations with the brokers until the job owner and a broker agree on a price for executing the job. The length indicates the number of instructions the job is expected to take. The model of a job in use is such that the job is progressed through a fixed number of instructions and then saved. So, if the job took more instructions than is determined in the request, the resource executes the number of instructions the user (job owner) paid for and returns the results to the user. Then, the user can ask for executing the rest of instructions in a new job request. The deadline is the date by which the job must be completed. The class is a descriptive category of a job that is based on the computer resources it requires when running.

For every job, there is the submission time which is the time when the negotiation starts for the job (the job is submitted to a group of brokers). If a resource accepts to execute the job, the job is allocated (submitted) to the resource at time called the allocation time. Then, the job is assigned to one of the available processors belong to the resource at time called the start time. Finally, the job finishes its execution and the results are returned to the job owner at time called finish time.

The user in this framework is an entity that sends job parameters and receives bid parameters as appears in figure 3.3. The user sends jobs to brokers, so they can find suitable resources for executing the user's jobs. On the other hand, the brokers send their bids to the user after contacting the resources, so the user can select one the bids or keeps negotiating with brokers.

Every user maintains historical performance for brokers which is used in selecting the brokers the user wants to contact. So in the next negotiations, the users just send requests to brokers with good historical performance to reduce the number of negotiations that happen between them and brokers. The historical

performance is updated based on the responses from various brokers. Furthermore, every user sends requests to the brokers with poor historical performance after a while to see if their performances (e.g. their responses to its requests) improve.

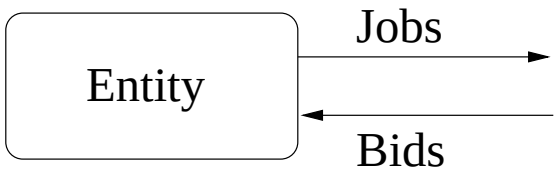

Figure 3.3: User.

3.2.3 Brokers

The broker aims to find suitable resources for executing the jobs owned by users that the broker acts on behalf of them. The existence of brokers is important for users because they have knowledge about resources reside in the Grid. The brokers employ various strategies and aim to maximise their own profit. Different to users and resources, brokers interact with two classes of entities that are users and resources, while users only interact with brokers and resources only interact with brokers.

In this framework, the broker is represented as an entity that sends job and bid parameters and receives job and bid parameters as shown in figure 3.4. It sends bid and job parameters to users and resources, respectively, and receives bid and job parameters from resources and users, respectively. Each broker just sends one bid to user at a time.

Similar to users, brokers sustain historical performance, but for resources. The brokers employ historical performance to reduce the number of communications that occur between them and resources. As a result, the brokers only contact a group of resources.

As resources, brokers pay two kinds of costs: cost per MI and cost per time unit. However, the costs paid by brokers are usually less than the costs paid by resources. This is due to the fact that resources have processors that need to be

maintained.

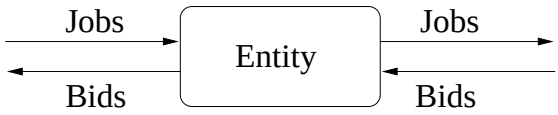

Figure 3.4: Broker.

3.3 Entity strategies

3.3.1 Resource strategies

This work considers three resource strategies: Price, Deadline and Price-Deadline. The need of three strategies arises from the fact the parties negotiate the price and the deadline of the job, so three strategies must exist that concern price, deadline and both. The strategies are described in the following three sections.

Price strategy

A resource that employes this strategy checks the broker's price. If the price is less than its minimum acceptable price (which can be fixed or dynamic) or the class of the job that was submitted by broker is not accepted by the resource strategy (equation 3.1), the resource ignores it. Else (equation 3.2), it sends a bid with an initial price which is greater than or equal to its minimum acceptable price.

$$Broker\ price < mnresacpr \text{ or } Job's\ class \notin accjobcl \tag{3.1}$$

$$Broker\ price \geq mnresacpr \text{ and } Job's\ class \in accjobcl \tag{3.2}$$

Where *mnresacpr* is the resource minimum acceptable price and *accjobcl* contains the job classes that are accepted by the resource strategy.

In further negotiations, the resource makes the same check in the previous paragraph. But, in the case that is represented by equation 3.2, the resource updates its minimum acceptable price. The current resource bid's price and completion time are also updated, and then a new bid is sent to the broker. However,

there is a limit on the number of negotiations/rounds to which the resource can respond.

Pseudocode 1 shows the pseudocode of this strategy. For definitions of variables used in this pseudocode, refer to table 3.2.

Pseudocode 1: Resource strategy 1 (Price strategy).

```
Initialise conneg to true;
Initialise mnresacpr, respr, resct;
Initialise initial price, initial completion time;
if conneg = true then
    if recbrpr ≥ mnresacpr and recjcl ∈ accjobcl then
        if round# = 1 then
            nofrounds = nofrounds − 1;
            Send a bid that includes the initial price and completion time in
            addition to other parameters to the broker;
        else
            if nofrounds > 0 then
                Update mnresacpr, respr and resct using a method (e.g. fixed
                decrements to their current values);
                nofrounds = nofrounds − 1;
                Send a bid that includes respr and resct in addition to other
                parameters to the broker;
            else
                conneg = false;
            end
        end
    end
end
```

Deadline strategy

In this strategy, the broker's deadline is checked. If this deadline is less than the resource minimum acceptable deadline (which can be fixed or dynamic) or the class of the job that the broker submitted is not accepted by the resource strategy (equation 3.3), the resource neglects the request. Otherwise (equation 3.4), it

sends a bid with an initial completion time that is greater than or equal to its minimum acceptable deadline.

$$Broker\ deadline < mnresacdl \text{ or } Job's\ class \notin accjobcl \tag{3.3}$$

$$Broker\ deadline \geq mnresacdl \text{ and } Job's\ class \in accjobcl \tag{3.4}$$

Where *mnresacdl* is the resource minimum acceptable deadline and *accjobcl* contains the job classes that are accepted by the resource.

In further negotiations, the resource makes the same check as previously except that in the case that is represented by equation 3.4, the resource updates its minimum acceptable deadline, price and completion time and a new bid is sent to the broker. The resource has to check if it reached its maximum allowed number of rounds to determine if it has to continue the negotiation for this job or not.

Pseudocode 2 shows the pseudocode of this strategy. For definitions of variables used in the pseudocode, refer to table 3.2.

Pseudocode 2: Resource strategy 2 (Deadline strategy).

```
Initialise conneg to true;
Initialise mnresacdl, respr, resct;
Initialise initial price, initial completion time;
if conneg = true then
    if recbrdl ≥ mnresacdl and recjcl ∈ accjobcl then
        if round# = 1 then
            nofrounds = nofrounds − 1;
            Send a bid that includes the initial price and completion time in
            addition to other parameters to the broker;
        else
            if nofrounds > 0 then
                Update mnresacdl, respr and resct using a method (e.g. fixed
                decrements to their current values);
                nofrounds = nofrounds − 1;
                Send a bid that includes respr and resct in addition to other
                parameters to the broker;
            else
                conneg = false;
            end
        end
    end
end
```

Price-Deadline strategy

Both broker's price and deadline are checked. If the price or deadline is less than the resource minimum acceptable price or deadline, respectively or the class of the job that was submitted is not accepted by the resource strategy as in equation 3.5, the resource neglects the broker's offer. Else (equation 3.6), it sends a bid with an initial price and completion time that are greater than or equal to its minimum acceptable price and deadline, respectively.

$$\begin{aligned} \textit{Broker price} &< \textit{mnresacpr} \text{ or} \\ \textit{Broker deadline} &< \textit{mnresacdl} \text{ or} \\ \textit{Job's class} &\notin \textit{accjobcl} \end{aligned} \tag{3.5}$$

$$\begin{aligned} \textit{Broker price} &\geq \textit{mnresacpr} \text{ and} \\ \textit{Broker deadline} &\geq \textit{mnresacdl} \text{ and} \\ \textit{Job's class} &\in \textit{accjobcl} \end{aligned} \tag{3.6}$$

Where *mnresacpr* is the resource minimum acceptable price, *mnresacdl* is the resource minimum acceptable deadline and *accjobcl* contains the job classes that are accepted by the resource.

In further negotiations, the resource makes the same checks, however, in the case that is represented by equation 3.6, the resource updates its minimum acceptable price and deadline. Moreover, the current resource bid's price and completion time are also updated and the resource sends a new bid to the broker. Again as in previous strategies, the resource has to check if it reached its maximum allowed number of rounds.

Pseudocode 3 shows the pseudocode of the strategy. For definitions of variables used in the pseudocode, refer to table 3.2.

Pseudocode 3: Resource strategy 3 (Price-Deadline strategy).

```
Initialise conneg to true;
Initialise mnresacpr, mnresacdl, respr, resct;
Initialise initial price, initial completion time;
if conneg = true then
    if recbrdl ≥ mnresacdl and recbrpr ≥ mnresacpr and recjcl ∈ accjobcl then
        if round# = 1 then
            nofrounds = nofrounds − 1;
            Send a bid that includes the initial price and completion time in
            addition to other parameters to the broker;
        else
            if nofrounds > 0 then
                Update mnresacpr, mnresacdl, respr and resct using a method
                (e.g. fixed decrements to their current values);
                nofrounds = nofrounds − 1;
                Send a bid that includes the respr and resct in addition to other
                parameters to the broker;
            else
                conneg = false;
            end
        end
    end
end
```

3.3.2 User strategies

The users employ three strategies for selecting the best bids for executing their jobs. As resources, the users require three strategies. Each strategy regards one the two negotiated requirements (price or completion time) or both. The strategies are Price, Completion_time and Price-Completion_time and are described below.

Table 3.2: Definitions of pseudocodes' variables of resource strategies.

Variable	Definition
accjobcl	The list that contains the job classes that are accepted by the resource strategy.
respr	The current resource price. $respr > 0$
resct	The current resource completion time. $resct > Current\ time$
recbrpr	The received broker price. $recbrpr > 0$
recbrdl	The received broker deadline. $recbrdl > Current\ time$
recjcl	The received job's class. $recjcl \in N$
mnresacpr	The current resource minimum acceptable price. $mnresacpr > 0$
mnresacdl	The current resource minimum acceptable deadline. $mnresacdl > Current\ time$
nofrounds	The maximum number of rounds to which the resource can respond. $nofrounds \geq 1$
conneg	The variable that determines if the negotiation has to continue or not. $conneg = false \mid true$
round#	The current round number. $round\# \geq 1$

Price strategy

In this strategy, the user starts the negotiation with an initial price that is less than or equal to its maximum acceptable price (which can be fixed or dynamic) and that is sent with other job parameters to brokers.

Then, a group of brokers replies with bids and some brokers might not reply at all. For all brokers replied, the user checks each bid price. If the bid price is greater than the user maximum acceptable price or the bid is expired (equation 3.7), the user just neglects it. Otherwise (equation 3.8), the user adds the bid to the other bids it will consider. If all the bid prices are greater than the user maximum acceptable price, the user updates its maximum acceptable price. The user also updates its current price and deadline, and then sends new offers to the brokers.

$$Broker\ price > mxusacpr \text{ or } expdate < Current\ time \tag{3.7}$$

$$Broker\ price \leq mxusacpr \text{ and } expdate \geq Current\ time \tag{3.8}$$

Where *mxusacpr* is the user maximum acceptable price and *expdate* is the date

when the broker bid will expire.

The negotiation continues until the user receives a valid bid with price that is less than or equal to its maximum acceptable price (negotiation is successful) or until the user determines it doesn't want to negotiate anymore (negotiation is unsuccessful). If two or more bid (broker) prices satisfy equation 3.8, the user selects the bid with the lowest price. If two or more bids have the same lowest price, the user selects the one that has minimum completion time. If they also have the same minimum completion time, the user just selects one of them randomly.

Pseudocode 4 shows the pseudocode of this strategy. For definitions of variables used in the pseudocode, refer to table 3.3.

Pseudocode 4: User strategy 1 (Price strategy).

```
Initialise conneg to true;
Initialise mxusacpr, uspr, usdl;
Initialise initial price, initial deadline;
if conneg = true then
    if round# = 1 then
        nofrounds = nofrounds − 1;
        for i = 1 to Whtosels.size do
            Send an offer that includes the initial price and deadline in addition to other job
            parameters to the broker whose number is in Whtosels.(i);
        end
    else
        if bprls.size > 0 or expdate.size > 0 then
            for j = 1 to bprls.size or expdate.size do
                if bprls.(j) ≤ mxusacpr and expdate.(j) ≥ Current time then
                    conlspr.add(bprls.(j));
                end
            end
            if conlspr.size > 0 then
                Select the broker that sent the bid price ∈ min{conlspr};
            else
                if nofrounds > 0 then
                    Update mxusacpr, uspr and usdl using a method (e.g. fixed increments to their
                    current values);
                    nofrounds = nofrounds − 1;
                    for x = 1 to Whtosels.size do
                        Send an offer to the broker whose number is in Whtosels.(x);
                    end
                else
                    conneg = false;
                end
            end
        else
            if nofrounds > 0 then
                Update mxusacpr, uspr and usdl using a method (e.g. fixed increments to their
                current values);
                nofrounds = nofrounds − 1;
                for x = 1 to Whtosels.size do
                    Send an offer to the broker whose number is in Whtosels.(x);
                end
            else
                conneg = false;
            end
        end
    end
end
```

Completion_time strategy

At the beginning of the negotiation, the user sends an initial deadline that is less than or equal to its maximum acceptable completion time (which can be fixed or dynamic) in addition to other job parameters to brokers.

The actions that are taken by user based on broker replies are:

- No action is taken by user if no bid is received from the broker.
- The user neglects the bid that was received from the broker if the bid's completion time is greater than its maximum acceptable completion time or the bid is expired (equation 3.9).
- The user adds the bid to the other bids it will consider if the bid's completion time is less than or equal to its maximum acceptable completion time and the bid is still valid (equation 3.10).

$$Broker\ completion\ time > mxusacct \text{ or } expdate < Current\ time \tag{3.9}$$

$$Broker\ completion\ time \leq mxusacct \text{ and } expdate \geq Current\ time \tag{3.10}$$

Where *mxusacct* is the user maximum acceptable completion time and *expdate* is the date when the broker bid will expire.

If none of the bids fulfil the conditions in equation 3.10, the user updates its maximum acceptable completion time, price and deadline. Then, new offers are sent to brokers.

The negotiation lasts until the user receives a valid bid with completion time that is less than or equal to its maximum acceptable completion time (negotiation is successful) or until it wants to discontinue the negotiation (negotiation is unsuccessful). If two or more bid completion times are less than or equal to the maximum, the user selects the bid with the minimum completion time. If two or more bids have the same minimum completion time, the user selects the one with

the lowest price. If they also have the same lowest price, the user selects one of them randomly.

Pseudocode 5 shows the pseudocode of this strategy. For definitions of variables used in the pseudocode, refer to table 3.3.

Pseudocode 5: User strategy 2 (Completion_time strategy).

```
Initialise conneg to true;
Initialise mxusacct, uspr, usdl;
Initialise initial price, initial deadline;
if conneg = true then
    if round# = 1 then
        nofrounds = nofrounds − 1;
        for i = 1 to Whtosels.size do
            Send an offer that includes the initial price and deadline in addition to other job
            parameters to the broker whose number is in Whtosels.(i);
        end
    else
        if bctls.size > 0 or expdate.size > 0 then
            for j = 1 to bctls.size or expdate.size do
                if bctls.(j) ≤ mxusacct and expdate.(j) ≥ Current time then
                    conlsct.add(bctls.(j));
                end
            end
            if conlsct.size > 0 then
                Select the broker that sent
                the bid completion time ∈ min{conlsct};
            else
                if nofrounds > 0 then
                    Update mxusacct, uspr and usdl using a method (e.g. fixed increments to their
                    current values);
                    nofrounds = nofrounds − 1;
                    for x = 1 to Whtosels.size do
                        Send an offer to the broker whose number is in Whtosels.(x);
                    end
                else
                    conneg = false;
                end
            end
        else
            if nofrounds > 0 then
                Update mxusacct, uspr and usdl using a method (e.g. fixed increments to their
                current values);
                nofrounds = nofrounds − 1;
                for x = 1 to Whtosels.size do
                    Send an offer to the broker whose number is in Whtosels.(x);
                end
            else
                conneg = false;
            end
        end
    end
end
```

Price-Completion_time strategy

This strategy regards both price and completion time to decide which broker bid to select. At the beginning of the negotiation, the user sends an initial price and deadline that are less than or equal to its maximum acceptable price and completion time, respectively, with other job parameters to brokers.

The actions that are taken by user based on broker replies are:

- No action is taken by user if no bid is received from the broker.
- The user neglects the bid that is received from the broker if the bid's completion time or price is greater than its maximum acceptable completion time or price, respectively or the bid is expired as in equation 3.11.
- The user adds the bid to the other bids it will consider if the bid's completion time and price are less than or equal to its maximum acceptable completion time and price, respectively, and the bid is still valid as in equation 3.12.

$$\begin{aligned} \textit{Broker completion time} &> \textit{mxusacct} \text{ or} \\ \textit{Broker price} &> \textit{mxusacpr} \text{ or} \\ \textit{expdate} &< \textit{Current time} \end{aligned} \tag{3.11}$$

$$\begin{aligned} \textit{Broker completion time} &\leq \textit{mxusacct} \text{ and} \\ \textit{Broker price} &\leq \textit{mxusacpr} \text{ and} \\ \textit{expdate} &\geq \textit{Current time} \end{aligned} \tag{3.12}$$

Where *mxusacct* is the user maximum acceptable completion time, *mxusacpr* is the user maximum acceptable price and *expdate* is the date when the broker bid will expire.

If none of the bids meet the conditions in equation 3.12, the user updates its

maximum acceptable completion time and price, deadline and price. The user then sends new offers to brokers.

The user that employs this strategy continues the negotiation until it receives a valid bid with completion time and price that are less than or equal to its maximum acceptable completion time and price, respectively (negotiation is successful), or until its maximum allowed number of rounds is reached (negotiation is unsuccessful). If the user received two or more bids that are acceptable to it, the user selects the bid with minimum value of $((\textit{Completion time} - \textit{Current time}) \times \textit{Price})$. If the minimum is achieved by several bids, the user selects one of them randomly.

Pseudocode 6 shows the pseudocode of the strategy. For definitions of variables used in the pseudocode, refer to table 3.3.

Pseudocode 6: User strategy 3 (Price-Completion_time strategy).

```
Initialise conneg to true;
Initialise mxusacpr, mxusacct, uspr, usdl;
Initialise initial price, initial deadline;
if conneg = true then
    if round# = 1 then
        nofrounds = nofrounds − 1;
        for i = 1 to Whtosels.size do
            Send an offer that includes the initial price and deadline in addition to other job
            parameters to the broker whose number is in Whtosels.(i);
        end
    else
        if bctls.size > 0 or bprls.size > 0 or expdate.size > 0 then
            for j = 1 to bctls.size or bprls.size or expdate.size do
                if bctls.(j) ≤ mxusacct and bprls.(j) ≤ mxusacpr and expdate.(j) ≥ Current time
                then
                    conlsct.add(bctls.(j)) and conlspr.add(bprls.(j));
                end
            end
            if conlspr.size > 0 or conlsct.size > 0 then
                Select the bid with minimum value of ((Completion time − Current time) × Price);
            else
                if nofrounds > 0 then
                    Update mxusacpr, mxusacct, uspr and usdl using a method (e.g. fixed
                    increments to their current values);
                    nofrounds = nofrounds − 1;
                    for x = 1 to Whtosels.size do
                        Send an offer to the broker whose number is in Whtosels.(x);
                    end
                else
                    conneg = false;
                end
            end
        else
            if nofrounds > 0 then
                Update mxusacpr, mxusacct, uspr and usdl using a method (e.g. fixed increments to
                their current values);
                nofrounds = nofrounds − 1;
                for x = 1 to Whtosels.size do
                    Send an offer to the broker whose number is in Whtosels.(x);
                end
            else
                conneg = false;
            end
        end
    end
end
```

Table 3.3: Definitions of pseudocodes' variables of user strategies.

Variable	Definition
Whtosels	The list that contains the broker numbers to which the user wants to send its job request.
expdate	The list that contains the expiry dates of the broker bids.
bprls	The list that contains the bid prices that were received from brokers. *Bid price* > 0
bctls	The list that contains the bid completion times that were received from brokers. *Bid completion time* $>$ *Current time*
conlspr	The list that contains the bid prices that the user wants to consider.
conlsct	The list that contains the bid completion times that the user wants to consider.
round#	The current round number. *round#* ≥ 1
uspr	The current user price. *uspr* > 0
usdl	The current user deadline. *usdl* $>$ *Current time*
mxusacpr	The user maximum acceptable price. *mxusacpr* > 0
mxusacct	The user maximum acceptable completion time. *mxusacct* $>$ *Current time*
conneg	The variable that determines if the negotiation has to continue or not. *conneg* $=$ *true* $\mid$ *false*
nofrounds	The maximum number of rounds the user wants to initiate. *nofrounds* ≥ 1

3.3.3 Broker strategies

Like users and resources, brokers have their strategies too and they aim to maximise their own utilities. The priority of brokers is to maximise the profit they generate from negotiating on behalf of users and this what User-Resource_price_difference strategy is about. The revenue generated equals to the difference between the user and resource prices, if the broker bid was successful. Another group of brokers (which employ User-Resource_price_difference-Deadline_met) don't consider bids from resources, before they ensure the bids satisfy the deadlines of their users. The details of broker strategies are mentioned in the rest two sections.

User-Resource_price_difference strategy

This strategy accepts every job execution request from users. For each request, it sends a bid with price that is less than the sent user price as in equation 3.13 in addition to other parameters to resources.

$$User\ price > brsentpr \tag{3.13}$$

Where *brsentpr* is the price sent from broker to resources.

The actions that are taken by broker based on resources replies are:

- No action is taken by broker if no bid is received from the resource.
- The broker neglects the bid that was received from the resource, if the bid's price is greater than *brsentpr* or the bid is expired (equation 3.14). The reason is that the user price must be greater than the received resource price in order for the broker to receive some profit (the difference between user and resource prices).
- The broker adds the bid to the other bids it will consider if the bid's price is less than or equal to *brsentpr* and the bid is still valid (equation 3.15).

$$Resource\ price > brsentpr \text{ or } expdate < Current\ time \tag{3.14}$$

$$Resource\ price \leq brsentpr \text{ and } expdate \geq Current\ time \tag{3.15}$$

Where *brsentpr* is the price sent from broker to resources and *expdate* is the date when the resource bid will expire.

If none of the resource bids satisfy equation 3.15, the broker doesn't send a bid to the user that sent the request. However, if two or more bids satisfy the equation, the broker selects the bid with the lowest price. If two or more bids have the same lowest price, the broker selects the one that has minimum completion time. If they have also the same minimum completion time, the broker selects one of them randomly.

Pseudocode 7 shows what the strategy does when it receives a request from a user, while pseudocode 8 shows what it does when it is the time to make a decision. Table 3.4 shows the definitions of variables used in pseudocodes 7 and 8.

Pseudocode 7: What broker strategy 1 (User-Resource_price_difference strategy) does when a user request is received.

```
for i = 1 to Whtosels.size do
    Send an offer that includes the price and deadline in addition to other
    job parameters to the resource whose number is in Whtosels.(i);
```

Pseudocode 8: What broker strategy 1 (User-Resource_price_difference strategy) does when it is the time to make a decision.

```
if rprls.size > 0 or expdate.size > 0 then
    for j = 1 to rprls.size or expdate.size do
        if rprls.(j) ≤ brsentpr and expdate.(j) ≥ Current time then
            conls.add(rprls.(j));
    if conls.size > 0 then
        Select the resource that sent the bid price ∈ min{conls};
        Send a bid with price equals the user request price to the user;
```

Table 3.4: Definitions of pseudocodes' variables of broker strategies.

Variable	Definition
Whtosels	The list that contains the resource numbers to which the broker wants to send its job request.
expdate	The list that contains the expiry dates of the resource bids.
rprls	The list that contains the bid prices that were received from resources. *Bid price* > 0
conls	The list that contains the bid prices that the broker wants to consider.
brsentpr	The price sent from broker to resources *brsentpr* > 0

User-Resource_price_difference-Deadline_met strategy

The only difference between this strategy and the previous strategy is that the sent bid from a resource must also have a completion time that is less than or equal to the deadline of the submitted job in order for the broker to consider it as appears in equation 3.16.

$$\textit{Completion time} \leq \textit{User deadline} \tag{3.16}$$

3.4 Chapter summary

In summary, this work has introduced a Grid computing framework that supports economic scheduling using tendering. In this framework, three classes of entities exist: users, brokers and resources.

Users are the buyers and they aim to find resources that meet their requirements, while maximising their utilities (e.g. reduce the cost). Users send their requests to brokers which in turn contact the resources to find suitable ones to execute the jobs.

Brokers are the auctioneers and they are mediators between buyers (users) and sellers (resources). Brokers act on behalf of users and their objective is to find suitable resources for executing user jobs, while maximising their own utilities (e.g. profit) as well.

The final class of entities is resources that are the sellers and which have processors that are utilised by user jobs. As other classes of entities, resources' goal is to maximise their utilities (e.g. profit) using the strategies they employ.

Three user strategies have been described: Price, Completion_time and Price-Completion_time strategies. The first strategy concerns price when considering a bid, while the second strategy concerns completion time for considering a bid. In the last strategy, both price and completion time are checked to determine if to consider a bid or not.

On the other hand, three resource strategies have been defined. The first strategy is Price strategy and it regards price in order to indicate if to send a bid or not. The second strategy is Deadline strategy and it regards deadline instead of price to determine if it is worth sending a bid or not. The third strategy is Price-Deadline strategy and regards both price and deadline to specify if to submit a bid or not.

Finally, two broker strategies have been presented: User-Resource_price_difference and User-Resource_price_difference-Deadline_met strategies. Both strategies behave the same except that the second strategy ensures the deadline of job owner was satisfied in the resource bid (by considering a resource bid only if its completion time is less than or equal to the deadline of the job owner).

Chapter 4

MICOSim: A simulator for modelling economic scheduling

4.1 Introduction

This chapter is concerned with the design and implementation of MICOSim, an event-driven simulator written in Java for evaluating the performance of Grid entities (users, brokers and resources) under different scenarios.

In the previous chapter, a framework for economic scheduling in Grid computing using tendering was introduced. However, the entities in the framework such as users, brokers and resources employ strategies that their performance need to be evaluated under different circumstances. Unfortunetly, it is nearly impossible to evaluate the performance of different entities in a repeatable and controllable manner for different scenarios such as changing the number of entities in real Grid environments. The reason is that the availability of resources and their load change with time and it is impossible for an individual user to control actions of other users on the Grid.

Thus, a simulator is needed for evaluating the performance of different entity strategies under different scenarios. Those scenarios comprise:

- Varying the numbers of users, resources and brokers.
- Varying the entities' specifications such as varying the numbers of jobs that are owned by different users and jobs' lengths and classes, and varying the numbers of processors the resources have and processors' speeds.
- Varying the strategies that are employed by various users, brokers and resources.
- Varying the strategies' parameters like the parameters they send to other entities such as prices, completion times and deadlines.

After this introduction, Section 4.2 discusses related work including some of its limitations. Section 4.3 and 4.4 introduce the implemented simulator's components and their interactions, respectively. Finally, section 4.5 concludes this chapter.

4.2 Related work

Simulation is defined as "*attempting to predict aspects of the behaviour of some system by creating an approximate model for it*" [72]. Building simulators has a number of advantages: there is no need for building a real system, conducting more easily controlled experiments and allowing to run a huge number of experiments. A number of simulators have been implemented such as Bricks, SimGrid, GangSim, OptorSim and GridSim.

Bricks [88] is a Java-based simulator developed at the Tokyo Institute of Technology in Japan and is used for comparing scheduling algorithms and frameworks in client-server like global computing systems under different circumstances such as varying the workload. However, it uses centralised scheduling which is known to have drawbacks such as inscalability and single point of failure.

SimGrid [24] is an event-driven simulator developed in the university of California and it deals with single-client multi-server scheduling. However, because it can only be used for simulating a single client, it is hard to simulate a group of competing users that each employs its own strategy.

SimGrid 2 [60] and 3 [59] are enhanced versions of SimGrid which have new features involving simulating distributed scheduling agents in dynamic distributed environments and supporting more network models.

GangSim [33] simulates environments where there is a large number of institutions and users that control a huge number of computers and storage systems. In GangSim, the allocation of resources is decided from the interaction between virtual organisations(VOs). In MICOSim, the allocation of resources is determined from the interaction between individual entities.

OptorSim [13] is a Data Grid simulator written in Java for evaluating replica optimiser algorithms in various grid configurations. Furthermore, it is used for evaluating an economic model using a Peer to Peer auction protocol [12]. The economic model is used to choose replicas for running jobs and to determine where to create replicas dynamically in Grid sites by employing a file revenue prediction function.

Finally, Gridsim [21] is a Java-based toolkit and is used for modelling and simulation of entities in Grid environments. It is built on top of SimJava which is a discrete event simulation engine that runs entities in separate threads. It is mainly concern is Grid economy where there are users (buyers), resources (sellers) and brokers for discovering the available resources and allocating them to user jobs. Threads have a drawback which is platform dependency, so the program runs differently under various operating systems platforms [50]. GridSim doesn't support that the brokers have their own strategies too in order to maximise their own utilities.

Therefore, a simulator that supports decentralised scheduling in Grid computing has to be implemented. In this simulator, all entities can have their own utilities and can interact with each other using tender model. In this chapter, an event-driven simulator, MICOSim, that was implemented in Java and overcomes the limitations of the above simulators is described. In this simulator, threads were not used because of their drawback which is platform dependency. Additionally, this simulator was built from scratch.

4.3 MICOSim components

MICOSim's basic components are TheSystem, Entity, Entitystrategy and Scenario. TheSystem is the class that is responsible for handling the interaction (communications) between different entities. Entity is the class that different entities are created from. Also, each entity has an Entitystrategy that is an abstract class that contains definitions of methods that their bodies are the same for all strategies. It also contains abstract methods that are missing their bodies and that are defined in the subclasses inherited from this class such as user, broker and resource strategies. The definitions of those abstract methods in the subclasses rely on the specification of the created strategy. Scenario is the class that indicates the specifications of the performed simulation.

4.3.1 TheSystem

TheSystem is responsible for the interactions occur between various entities. For example, it informs the entities' strategies to take actions according to their order in its event list. Additionally, it forwards the messages between different entities like informing the entity that won the award of executing the job. An event list is a Vector class which contains the events that should happen through the simulation ordered by the occurrence time. An event is represented by an object which comprise the event's occurrence time, the entity's class and the entity's number. At the begining of the simulation, the list contains events that their relevant actions are initiating negotiations for user jobs. When the simulation starts, the system removes the first event from the list and executes the relevant action. Any new events that occur as a result are inserted on the list at the appropriate point. This continues until the event list becomes empty. If two ore more events have the same occurrence time, then their relevant actions are executed sequentially.

4.3.2 Entity

Entity is an object that can send and receive both jobs and bids. However, the sub-classes that are created from Entity class have some of the capabilities the entity has. For example, users can only send jobs and receive bids, while resources can do the opposite. On the other hand, brokers can send and receive both jobs and bids, but can't execute jobs as resources. Three classes of entities are created from Entity: users, brokers and resources. All entities have common things such as name which is unique, number which is used in communications to specify the recipient of job or bid parameters and strategy which is the course of action to achieve entity's goal(s).

The next sections describe the parameters and methods that are specific to each category of entities.

User

User is the entity that sends jobs and receives bids (see 3.2.2). The parameters that are specific to a user are:

- Number of jobs: The number of jobs the user has.
- Job IDs: The IDs (numbers) of the jobs belong to the user.
- Job lengths: The length of jobs belong to the user in Million Instructions (MI).
- Jobs' classes: The class of each job belongs to the user. The class shows the computer resources it requires when running.
- Job numbers of negotiations: How many negotiations occurred between the user and the brokers for every job.
- The number of jobs that were executed.
- Job costs: The cost the user paid for executing each of its jobs.
- The number of brokers the user knows about.

- Historical performance of each broker: Each broker has an integer value between min (very poor) and max (very good). Initially, each broker is given a value between min and max. This value is used to determine with which brokers the user keeps negotiating. This value is decreased or increased by the strategy that is employed by the user based on the occurrence of specific conditions.

Each user has also methods for informing the broker that the user accepted its bid, updating the historical performance of the broker that submitted the job that has just finished its execution, increasing the number of jobs that were executed when a job of the user is completed, increasing the number of negotiations occurred between the user and a broker and creating a new event for a new job if the submission of jobs is dynamic (e.g. the submission time of each job is determined based on the completion of the previous job).

Broker

The broker sends and receives jobs and bids (see 3.2.3). Every broker has a number of parameters and methods that are used during the negotiations. The broker parameters are:

- The job parameters (jobs which are owned by users who the broker acts on behalf) mentioned above like IDs, lengths and classes in addition to the IDs of the users sent them. The broker needs to keep information about the jobs submitted to it, so it can pass them later to resources which the broker will interact with.

- Jobs' numbers of negotiations: How many negotiations occurred between the broker and users, and between the broker and resources for every job submitted to the broker.

- Cost per MI: The cost resulted from acting on behalf of a job owner measured in MI.

- Cost per time unit: A continuous cost that results from maintaining the broker's requirements such as maintaining the broker's software and keeping the security up to date, and it is measured per time unit.
- Revenue: The entire amount of income before any deductions are made.
- Profit: The excess of income over expenses. The expenses are represented by cost per MI and cost per time unit.
- The number of resources the broker knows about.
- Historical performance of each resource: The same as historical performance in users, except it is for a resource and not for a broker.

The broker has four methods. The first method increases the number of negotiations occurred between it and a resource. The second method informs the resource that the broker accepted its bid. The third method updates the historical performance of the resource that didn't meet its sent completion time for a job. Finally, the fourth method increases the broker's profit and revenue when its bid is accepted by a user.

Resource

The resource (see 3.2.1) is an entity that receives jobs in order to execute them. Moreover, it sends bids to brokers that submitted jobs to it . As users and brokers, each resource has its parameters and methods. The resource parameters are:

- Number of processors: The number of processors the resource has.
- Number of free processors: The number of processors that are unallocated to any job.
- Processors' speeds measured in MIPS (Million Instruction per second).
- Next availabilities: The next availability for a processor is the time when the processor will be available (free). In other words, it is the time when the job

currently executing on the processor in addition to the jobs that currently in the execution queue if there are any will be completed.

- Total number of jobs that were executed on the resource.
- Total number of jobs that are waiting in the execution queue or currently executing on the resource.
- Jobs' numbers of negotiations: How many negotiations took place between the resource and brokers for every job submitted to the resource.
- Cost per MI: The cost resulted from executing the job measured in MI.
- Cost per time unit: A continuous cost that results from maintaining the resource's requirements like sustaining its machines, software and security and it is measured per time unit.
- Revenue: The same as revenue in brokers.
- Profit: The same as profit in brokers.

Each resource comprises the following methods that are called when the resource:

- receives a job to update its profit, revenue, number of free processors, the availability of the processor allocated to the job and the number of jobs waiting in the execution queue or currently executing. Additionally, an object that contains information about the submission of the job is created. For example, it contains information about when the job was submitted and when it is going to be received by its owner.
- finishes executing a job for updating the number of executed jobs on the resource and the numbers of jobs that are waiting in the execution queue or currently executing.

4.3.3 Entitystrategy

As mentioned above, Entitystrategy contains definitions for the methods that their bodies are the same for all inherited strategies, and misses definitions for the

abstract methods that their contents rely on the characteristics of the inherited classes. Entitystrategy has two defined methods for copying:

- job parameters from the entity (user or broker) that employs it in order for TheSystem to send them to a group of entities (broker(s) or resource(s)).
- bid parameters from the entity (broker or resource) that employs it in order for TheSystem to send them to a group of entities (user(s) or broker(s)).

On the other hand, Entitystrategy has also four abstract methods. The first method returns an object containing the parameters of the strategy (subclass) for a particular job. The second method determines how the strategy behaves. The third method is used for updating the availability of the processor allocated to a job. Finally, the fourth method is used to calculate the expected time needed to finish executing the job. The last two methods are only used by resource strategies.

There are three kinds of strategies that are user, broker and resource strategies. The sections below describe the parameters of each kind of strategy.

User strategy parameters

Each strategy comprises a number of parameters such as current deadlines and prices of jobs belong to the user who employs the strategy, to when the user wants to wait for each job before making a decision and other parameters that are specific to the strategy.

Broker strategy parameters

Each strategy controls a number of parameters that are:

- The broker's (that employs the strategy) current prices, deadlines, and completion times for users' jobs that acts on behalf of them.
- Expiry dates of broker's sent bids.
- A parameter that specifies if the broker sent a new bid in the last negotiation or not.

- Waiting time which determines until when the broker wants to wait before making a decision.
- Other variables that are specific to the strategy.

Resource strategy parameters

The parameters that are included in each resource strategy are:

- The resource's (that employs the strategy) current prices and completion times for users' jobs submitted by brokers to the resource which employs it.
- The job classes that are acceptable.
- A parameter that specifies if the resource sent a new bid in the last negotiation or not.
- Expiry dates of resource's sent bids.
- Other variables that are specific to the strategy.

4.3.4 Scenario

This class indicates the characteristics of the performed experiment involving:

- The kind of submission (e.g. static, dynamic).
- Number of users and number and lengths of jobs owned by each of them.
- Number of brokers.
- Number of resources and number and speeds of processors belong to each of them.
- The strategies employed by different entities.
- Costs per MI and costs per time unit of brokers and resources.
- Strategies' related parameters such as increment in price, initial price and maximum acceptable price.

4.4 MICOSim components' interaction

Figure 4.1 shows the interaction occurs between MICOSim's components. First of all, Scenario's parameters are passed to TheSystem. Then, TheSystem checks the first event in its eventlist to know which entity should start the negotiation first. Next, that entity employs its strategy to know the course of action to be taken. Then, the strategy adds a new event to the TheSystem event list that specifies with what entities it wants to negotiate. Furthermore, the strategy copies the needed parameters to the appropriate object and that will be used by the entities that will respond to the negotiation. Then, TheSystem deletes the current event and checks the next event to see what should happen next. This continues until the eventlist is empty.

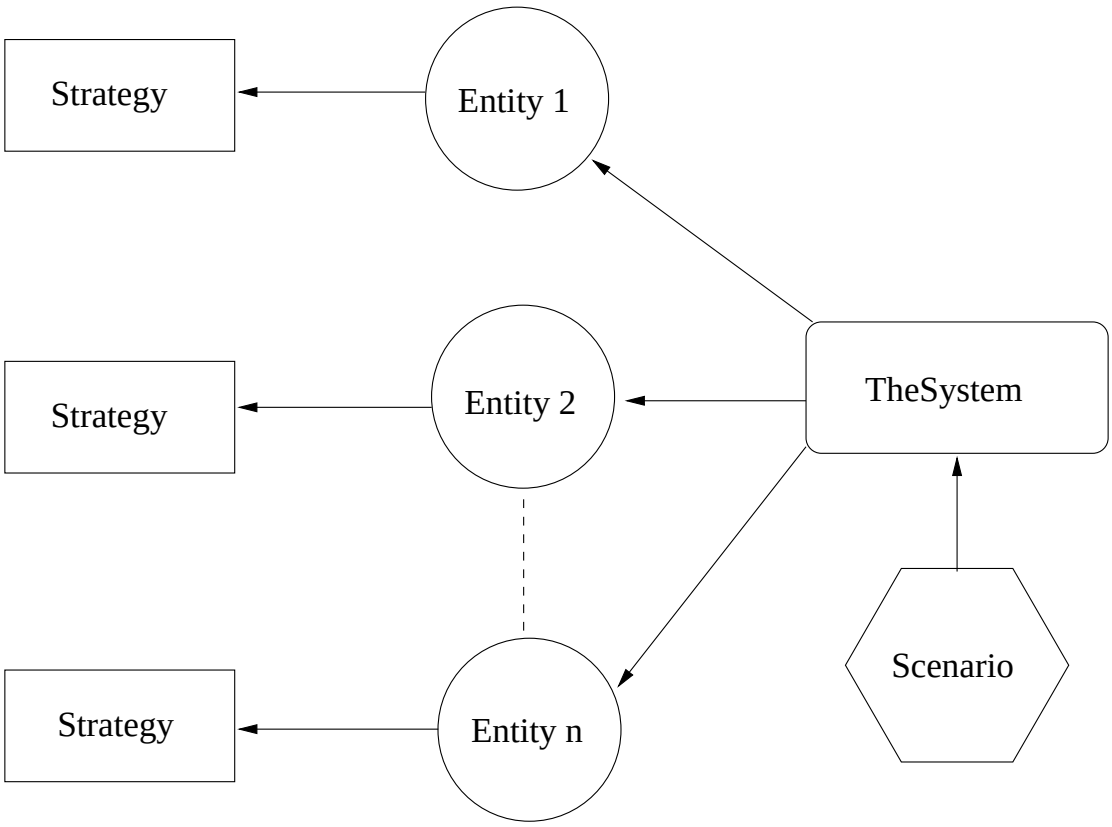

Figure 4.1: Interaction between MICOSim's components.

Figure 4.2 shows the negotiation occurs between a user and a broker. First, the user employs its strategy and then copies the required parameters for negotiation to a job description. Then, the job description is passed to TheSystem which in

turn forwards it to the appropriate broker when its time to take an action comes. When the broker employs its strategy, it copies the required parameters to a bid description which is passed to the TheSystem. Eventually, the TheSystem sends the bid description to the user when its time to take an action is reached.

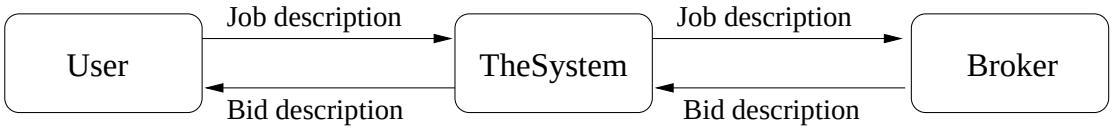

Figure 4.2: Negotiation between a user and a broker.

On the other hand, figure 4.3 shows the negotiation happens between a broker and a resource. What happens is similar to what mentioned above for figure 4.2 except that the broker passes a job description to the TheSystem, and the resource passes a bid description to the TheSystem.

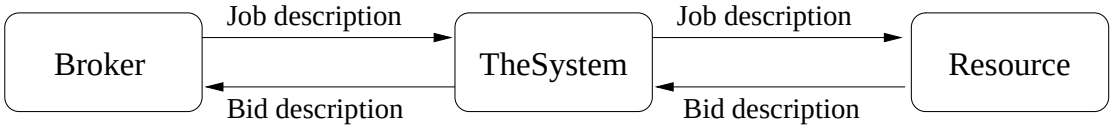

Figure 4.3: Negotiation between a broker and a resource.

4.5 Chapter summary

In this chapter, a simulator called MICOSim, for modelling economic scheduling in Grid computing using tender model has been described. It has the ability to simulate Grid entities under different scenarios. The users can have different number of jobs with different lengths measured by MI, while resources can have different number of processors with different speeds measured by MIPS. Additionally, the users can employ various strategies with different parameters such as prices and deadlines as well as resources that can employ various strategies with different parameters like prices and completion times. MICOSim also simulates brokers with different interests for acting on behalf of users.

This chapter has also presented the MICOSim's components including the required parameters and methods for each of them. Finally, the interaction occurs

between various components of MICOSim has been introduced.

This object-oriented simulator can be easily extended to support more models such as commodity and auction models. This can be achieved by modifying some of the classes which compose the simulator.

Chapter 5

Results and discussion

5.1 Introduction

In chapters 3 and 4, a Grid computing framework was presented and a simulator that models this framework described. In this chapter, the simulator is used to evaluate the performance of the entity strategies introduced in chapter 3 with static and dynamic submission of jobs.

This chapter is structured as follows: Section 5.2 describes the metrics for measuring the strategies' performance. Section 5.3 explains how the simulation is configured. Section 5.4 introduces a binary insert algorithm for improving the simulation performance. Section 5.5 demonstrates that our simulation is verified by showing a consistency between the simulation output and hand calculations. Section 5.6 discusses the simulation results. Finally, a summary is given in section 5.7.

5.2 Comparison metrics

In the current work, there are n users U_1, U_2, $\cdots$, U_n, m brokers B_1, B_2, $\cdots$, B_m and w resources R_1, R_2, $\cdots$, R_w. Each user i has x_i jobs J_{i1}, J_{i2}, $\cdots$, J_{ix_i} and

each resource j has y_j processors P_{j1}, P_{j2}, $\cdots$, P_{jy_j}.

In order to determine the best performing strategies a suitable set of metrics need to be defined. These metrics are influenced by the individual targets of each class of entity. For example, a user needs to have jobs executed within a particular time and for a particular cost; resources aim to make the most profit; etc.

Table 5.1 shows the metrics used to measure the performance of users, brokers and resources.

Table 5.1: Metrics of users, brokers and resources.

User metrics	
$M_1 = \frac{n}{tn} \times C1$	n = Number of jobs that were executed and owned by the user, tn = total number of jobs owned by the user and $C1$ is a constant.
$M_2 = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} L_i} \times C2$	n = number of jobs that were executed and owned by the user, C_i = Cost of executing job$_i$, L_i = Length of job$_i$ in MI and $C2$ is a constant.
$M_3 = \frac{\sum_{i=1}^{n} ID_i - CT_i}{n} \times C3$	n = Number of jobs that were executed and owned by the user, ID_i = Initial deadline of job$_i$, CT_i = Completion time of job$_i$ and $C3$ is a constant.
Broker metric	
$M_b = \sum_{i=1}^{m}(D_i - Ca_i) - Cmb$	m = number of jobs that were submitted by the broker and executed on a resource, D_i = Difference between the prices of job$_i$ owner and the resource that executed the job, Ca_i = Cost arisen from acting on behalf of the user who owns job$_i$ and Cmb is the cost of maintaining the broker entity during the time passed.
Resource metric	
$M_r = \sum_{i=1}^{m}(P_i - Ce_i) - Cmr$	m = number of jobs that were executed on the resource, P_i = Price of executing job$_i$ on the resource, Ce_i = Cost of executing job$_i$ on the resource and Cmr is the cost of maintaining the resource entity during the time passed.

The performance of three classes of entities is compared. For users, we use three metrics for measuring their performance: the job success rate (M_1), the average cost per MI (Million Instructions) (M_2) and average satisfaction rate per job (M_3). The first and the third metrics need to be maximised, while the second metric needs to be minimised. The first metric indicates the number of jobs

executed from the overall number of jobs. The second relates to the cost and therefore profit. The cost per MI is used rather than total cost so as the metric value is independent of job size. The final relates to whether jobs are executed within the deadline. Each metric is used separately to identify the best performing users and their strategies.

For brokers and resources, one metric is used for measuring their performance: the overall profit generated by a broker (M_b) or a resource (M_r). The need for this metric arises from the fact that this is an economic scheduling, and money is the main factor for evaluating the performance.

5.3 Simulation setup

The performance of the entity strategies is evaluated under different scenarios which result from varying parameter values. Four parameters are varied for user strategies, one for broker strategies and two for resource strategies.

The user strategy parameters that are varied:

- Initial price determinator (*IPD*): This determines the initial price, which depends on the maximum acceptable price. For example, if the parameter equals 0.6, then the initial price is equal to 0.6 or 60% of the maximum acceptable price of the same user.

$$IP = IPD \times MAP \tag{5.1}$$

$$MAP = MAPPMI \times JL \tag{5.2}$$

Where *IP* is the initial price,
IPD is the initial price determinator,
MAP is the maximum acceptable price,
MAPPMI is the maximum acceptable price per million instructions (MI) and it is equal to 0.01 and
JL is the job length in MI.

The *MAPPMI* is small (0.01), which restricts the size of *MAP*. The value of *MAPPMI* won't affect the evaluation of performance, because it is the same for all users in order to reduce the comparison complexity. However, the users within the simulation environment have different initial prices and increment of prices which are computed based on the initial price and increment in price determinators, respectively.

- Increment in price determinator (*IIPD*): This parameter determines the increment to the current price (which equals the initial price in the first round) in every round of negotiation depending on the maximum acceptable price. For example, if this parameter equals 0.1, the increment to the current price equals 0.1 or 10% of the maximum acceptable price of the same user.

$$IIP = IIPD \times MAP \tag{5.3}$$

Where *IIP* is the increment to the current price in every round of negotiation,
IIPD is the increment in price determinator and
MAP is the maximum acceptable price.

- Initial deadline determinator (*IDD*): This determines the initial deadline depending on the maximum acceptable completion time. For example, if this parameter equals 0.5, then the difference between the initial deadline and current time equals 0.5 or 50% of the difference between maximum acceptable completion time of the same user and current time.

$$ID - CT = IDD \times (MACT - CT) \tag{5.4}$$

$$MACT = CT + (MACTD \times \frac{JL}{RMIPS}) \tag{5.5}$$

Where *ID* is the initial deadline,
CT is the current time,

IDD is the initial deadline determinator,

MACT is the maximum acceptable completion time,

MACTD is the maximum acceptable completion time determinator and it equals 2,

JL is the job length and

RMIPS is the average speed of processors in MIPS (Million instructions per second) and it equals 300.

By setting *MACTD* to 2, the jobs will have reasonable *MACT* (neither tight nor relaxed).

- Increment in deadline determinator (*IIDD*): This parameter determines the increment to the current deadline (which equals the initial deadline in the first round) in every round of negotiation depending on the maximum acceptable completion time as in equation 5.6.

$$IID = IIDD \times (MACT - CT) + (CT - TPR) \tag{5.6}$$

Where *IID* is the increment in deadline,

CT is the current time,

TPR is the time when the *MACT* was computed in the previous round,

IIDD is the increment in deadline determinator and

MACT is the maximum acceptable completion time.

The *MACT* (which is computed when time equals *CT*) in round $n + 1$ is going to equal the *MACT* (which is computed when time equals *TPR*) in round n plus the difference between the two times, which is $(CT - TPR)$. Thus, the time difference $(CT - TPR)$ also needs to be considered when computing *IID*.

However, if the strategy doesn't consider the completion time as a threshold to determine if to accept a bid or not, the initial deadline and increment in deadline are computed based on a parameter called maximum deadline to send (*MDTS*).

Maximum deadline to send determinator ($MDTSD$) is used in computing $MDTS$ instead of maximum acceptable completion time determinator $MACTD$. $MDSTD$ and $MACTD$ are equivalent. In this case, initial deadline (ID), increment in deadline (IID) and maximum deadline to send ($MDTS$) are computed as in equations 5.7, 5.8 and 5.9.

$$ID - CT = IDD \times (MDTS - CT) \tag{5.7}$$

$$IID = IIDD \times (MDTS - CT) + (CT - TPR) \tag{5.8}$$

$$MDTS = CT + (MDTSD \times \frac{JL}{RMIPS}) \tag{5.9}$$

On the other hand, if the price is not considered by the strategy as a threshold to determine if to accept a bid or not, the initial price and increment in price are computed based on a parameter called maximum price to send ($MPTS$). Maximum price to send per MI ($MPTSPMI$) is used in computing $MPTS$ instead of $MAPPMI$. $MPTSPMI$ and $MAPPMI$ are equivalent. In this case, initial price (IP), increment in price (IIP) and maximum price to send ($MPTS$) are computed as in equations 5.10, 5.11 and 5.12.

$$IP = IPD \times MPTS \tag{5.10}$$

$$IIP = IIPD \times MPTS \tag{5.11}$$

$$MPTS = MPTSPMI \times JL \tag{5.12}$$

Revenue determinator (RD) is the parameter that is varied for broker strategies and it determines how much from the user price the broker wants to get as revenue. For example, if the parameter equals 0.2 and the user price is 10, the broker sends

to resources a request with price equals to 8 ((1 - 0.2) × 10). So, if one of the resources accepts the request, the broker keeps 2 (10 - 8) which is the difference between the user and resource prices.

$$BR = RD \times UP \tag{5.13}$$

$$PSTR = (1 - RD) \times UP \tag{5.14}$$

Where BR is the broker revenue,
RD is the revenue determinator,
UP is the user price and
$PSTR$ is the price sent from broker to resources.

Finally, the two parameters that are varied for resource strategies are:

- Resource minimum acceptable price per MI ($RMAPPMI$): This must be less than or equal to user maximum acceptable price per MI. Otherwise, the negotiations won't be successful.

$$RMAPPMI \leq UMAPPMI \tag{5.15}$$

$$RMAP = RMAPPMI \times JL \tag{5.16}$$

  Where $RMAPPMI$ is the resource minimum acceptable price per MI,
  $UMAPPMI$ is the user maximum acceptable price per MI,
  $RMAP$ is the the resource minimum acceptable price and
  JL is the job length in MI.

- Minimum acceptable deadline determinator ($MADD$): The higher this parameter's value is the higher minimum acceptable deadline. Minimum ac-

ceptable deadline is computed as shown in equation 5.17.

$$MAD = EPCT + MADD \times (EPCT - CT) \tag{5.17}$$

$$EPCT = TPF + \frac{JL}{MIPS} + TIO + TTRB \tag{5.18}$$

Where MAD is the minimum acceptable deadline,
$EPCT$ is the earliest possible completion time for the job,
$MADD$ is the minimum acceptable deadline determinator,
CT is the current time,
TPF is the time when the processor with the earliest availability is going to be free,
JL is the job length,
$MIPS$ is the processor's speed in MIPS,
TIO is the time needed for input and output and
$TTRB$ is the time needed for the resource bid to reach the broker.

For the strategy that is under consideration, one of the parameters is varied while the other parameters are kept fixed. Table 5.2 shows the fixed parameters and the values they take. There is no fixed parameters for broker strategies, because there is only one varied parameter. Furthermore, every strategy's performance is measured in a Grid environment where there is a large number of competing entities. In the simulated environment, there are 108 users, 10 brokers and 27 resources in addition to the entity running the strategy that is under consideration. In simulations, the number of jobs should be much larger than the number of available processors. Moreover, the number of brokers should be relatively small in order to reduce the number of communications (negotiations). Because of the existence of 3 user strategies, Every one third of users (36 users) employs the same strategy. Similiarily, every one third of resources (9 resources) has the same strategy, and every half of brokers (5 brokers) has the same strategy (3 resource

strategies and 2 broker strategies). However, entities which employ the same strategy have different sets of parameters.

Table 5.2: Fixed parameter values.

User strategies' parameters	
Initial price determinator	0.6
Increment in price determinator	0.1
Initial deadline determinator	0.6
Increment in deadline determinator	0.1
Resource strategies' parameters	
Minimum acceptable price per MI	0.007
Minimum acceptable deadline determinator	0.6

In every simulation, the parameters of other entity strategies that are within the simulated Grid environment are predetermined in such a way that covers the possible expectations. For example, low, medium and high (e.g. initial price) or low and high (e.g. increment in price) and which depends on the varied parameter and the range of values it takes. Table 5.3 shows the sets of values the paramters take. Each entity strategy takes one of the possible combinations.

Table 5.3: Sets of values the parameters take

User strategies' parameters	
Initial price determinator	{0.2, 0.5, 0.8}
Increment in price determinator	{0.1, 0.3}
Initial deadline determinator	{0.2, 0.5, 0.8}
Increment in deadline determinator	{0.1, 0.3}
Broker strategies' parameters	
Revenue determinator	{0.1, 0.3, 0.5, 0.7, 0.9}
Resource strategies' parameters	
Minimum acceptable price per MI	{0.002, 0.005, 0.008}
Minimum acceptable deadline determinator	{0.2, 0.5, 0.8}

In all scenarios, each user has 10 jobs. Two kinds of submission are considered in this work: *static* and *dynamic*. In static submission, the inter-arrival times of users' jobs follow negative exponential distribution with mean of 15. The differences between the inter-arrival times are small to represent the situation where the

system load is high. Negative exponential distribution is often used for modelling the inter-arrival times of events [58]. In dynamic submission, the submission time of each job is determined based on the completion of the previous job. Thus, the user submits its first job and waits until it receives the results of executing its job or until it fails to find a suitable resource for executing its job. Then, the next job is submitted. This continues until all of its jobs are submitted. The dynamic submission scenario represents the situation where the system load is low, because the user has to wait until its current job finishes its execution and then submits the next job. The submission time of the first job of every user is computed in the same way as in static submission, because no job is submitted before it. On the other hand, the jobs' lengths follow Pareto distribution with shape of 5 and scale of 100000. These values of shape and scale give a reasonable range of job lengths. The Pareto distribution is appropriate for modelling the lengths of supercomputer jobs (a few large ones, many small ones) [36]. The sort of jobs that are to be executed in Grid computing are similar to supercomputer jobs. The selection of values for shape and scale generates job lengths in range between 100000 MI and 600000 MI, and which seems to be reasonable range of values for job lengths.

Each user maintains a historical performance for each broker that takes an integer value between 1 and 10 (very poor to very good respectively) and that is:

- increased 1 (if it is less than 10) when a broker replies to the user request within the waiting time. Waiting time determines until when the user wants to wait before making a decision.

- increased 1 if 30 time units pass from the last change in historical performance of a broker and its historical performance is less than 4.

- decreased 1 (if it is greater than 1) when a broker doesn't respond to a job request 3 successive times or if it doesn't meet the job deadline.

If the historical performance goes below 4, the user won't send a request again to that broker until its historical performance reaches 4 again. 4 was selected because

it seems a reasonable value (around the middle of the range of values the historical performance can take).

During the negotiations, each user increments its price and deadline (after the first round of negotiation) based on the increment in price and deadline determinators, respectively. The user keeps negotiating until:

- finding a suitable resource for executing its job.
- reaching the maximum number of rounds it wants to initiate without finding a suitable resource for executing its job. Pseudocode 9 shows how the maximum number of rounds (*MNOR*) is computed for different user strategies. For Price and Completion_time strategies, the *MNOR* is indicated by the number of rounds needed for the *IPD* (see equation 5.1) and *IDD* (see equation 5.4), respectively to reach 1 which are computed by $\lceil \frac{1-IPD}{IIPD} \rceil$ and $\lceil \frac{1-IDD}{IIDD} \rceil$. One is added to the previous equations because initial price and deadline are going to be sent without being incremented in the first round of negotiation. Ceiling is used because in the last round the current price or deadline sometimes exceeds its limit after it is fully incremented based on the increment determinator. The limit is maximum acceptable price for current price and maximum acceptable completion time for current deadline. In such a case, the current price or completion time will be partially incremented, so it equals its limit instead of exceeding it. Because of this, the maximum number of rounds is going to be none integer number if ceiling is not used. For example, suppose the user employs Price strategy and initial price is 0.6, increment in price is 0.3 and maximum acceptable price is 10. In the first round, the price is going to be 6 (0.6×10). In the second round, the current price is going to be 9 ($(0.6 + 0.3) \times 10$). In the last round, the price is going to be 12 ($(0.9 + 0.3) \times 10$). However, it can be seen that the price exceeded the maximum acceptable price ($12 > 10$), so the price is going to be 10 instead of 12. The *MNOR* is computed as follows:

  $MNOR = \lceil \frac{1-IPD}{IIPD} \rceil + 1 = \lceil \frac{1-0.6}{0.3} \rceil + 1 = 3$

For Price-Completion_time strategy, the *MNOR* is determined by computing the number of rounds until both price and deadline reach maximum acceptable price and completion time, respectively. For instance, if the price needs 3 rounds to reach maximum acceptable price and deadline needs 4 rounds to reach maximum acceptable completion time, the *MNOR* is going to be 4 ($max(3, 4)$).

Pseudocode 9: How to calculate the maximum number of rounds the user wants to initiate.

if *the user employs Price strategy* **then**
  $MNOR = \lceil \frac{1-IPD}{IIPD} \rceil + 1$;
end

if *the user employs Completion_time strategy* **then**
  $MNOR = \lceil \frac{1-IDD}{IIDD} \rceil + 1$;
end

if *the user employs Price-Completion_time strategy* **then**
  $MNOR = max(\lceil \frac{1-IPD}{IIPD} \rceil + 1, \lceil \frac{1-IDD}{IIDD} \rceil + 1)$;
end

Where *MNOR* is the maximum number of rounds the user wants to initiate, *IPD* is the initial price determinator, *IIPD* is the increment in price determinator, *IDD* is the initial deadline determinator and *IIDD* is the increment in deadline determinator.

In the performed simulations, each processor's speed is 300 MIPS and each resource has 5 processors. Additionally, every resource only accepts the sent broker price and deadline, if it decides to reply. In other words, it doesn't send price and completion time different to sent broker's price and deadline, respectively. This represents the expected behaviour of resources in the real world. However, since the first resource strategy doesn't use the deadline threshold to decide if to reply or not, it does a check (see figure 5.1) to determine what its bid completion time is going to be, should it decides to reply.

Where `tempstpar.completion_time` is the earliest possible completion time for the job, `jd.deadline` is the deadline of the requester (broker), `TheSystem.simtime`

```
1    if(jd.deadline < (tempstpar.completion_time +
2       (0.3 * (tempstpar.completion_time - TheSystem.simtime)
           )))
3    {
4       tempstpar.negcomp = tempstpar.completion_time +
5       (0.3 * (tempstpar.completion_time - TheSystem.simtime)
           );
6    }//if end
7    else
8    {
9       tempstpar.negcomp = jd.deadline;
10   }//else end
```

Figure 5.1: The Java code for determining the sent completion time of resource price strategy

is the simulation time and `tempstpar.negcomp` is the completion time the resource is going to send to the requester.

The above check is important to overcome the problem of sending a completion time that is equivalent to the requester's deadline but can't be met by the resource. This is especially true when the resource is congested. So, if the requester deadline can be met, the resource sends a completion time equivalent to it. Otherwise, the resource sends a completion time which can be satisfied.

In the performed simulations, resources accept all job classes. The measurement of the performance of different strategies won't be accurate, if various resources only accept to execute some classes of jobs. The reason is that the performance might partially result from considering requests with specific job classes and not only because of the employed strategy. Because the resources won't negotiate forever for each job request, it is assumed that the maximum number of rounds to which the resource can respond for every request is five. Finally, the cost per MI and cost per time unit which are deducted from the resource revenue are equal to 0.002 and 1.0, respectively. The costs per MI and time unit are small, which restricts the total costs deducted from the overall revenue received from executing jobs especially that the jobs executed on the Grid resources are usually long. Long jobs need large number of MIs and take a long time (large number of time units) to execute.

When a broker receives a job request from a user, it negotiates as follows: First,

it sends offers to a group of resources and waits for replies from them. Then, it processes the replies (bids) and sends a bid to the user based on the received bids from resources. The employed strategy determines how the broker selects a resource bid. If there is no reply from any of the resources or none of the replies is interesting, it doesn't send a bid.

Since the main concern of brokers is to make profit, they forward the sent user deadline to resources as it is and just send a new price for executing the job and which should be less than the sent user price in order to keep some money for themselves. As resources, the cost per MI and cost per time unit are deducted from the broker revenue and they equal to 0.0005 and 0.25, respectively. As with resources, the costs per MI and time unit are small, which restricts the total costs deducted from the overall revenue received from acting on behalf of users. The costs for resources are higher than the costs for brokers, because the expenses that are paid for maintaining resources are higher than the expenses arisen from acting on behalf of users as in brokering.

As with users, brokers maintain historical performance, but for resources. The historical performance is updated in the same way as users' historical performance. The existence of historical performance arises from the need of reducing the number of negotiations occur between various entities. By reducing the number of negotiations, both overhead and bandwidth consumption are also reduced.

Each user or broker makes a decision when the number of received bids equals to the number of sent requests or when the waiting time is reached. The waiting time equals the current time plus 10 time units for every user, while it equals the current time plus 5 time units for every broker. The time needed for a request or bid to reach another entity is one time unit. The computation of the waiting time should take into account the time expected until the other entities reply to the request. It can be seen that more time units are added to the current time when computing the waiting time for users than the time units added when computing the waiting time for brokers. This is because when a user sends a request to a broker, the broker contacts a group of resources before replying to the user

request. On the other hand, when a broker sends a request to a resource, the resource can respond directly and therefore it is expected to take shorter time to reply. At the beginning of the performed simulation, users and brokers send their requests to all brokers and resources, respectively. Then, the users and brokers send their requests only to brokers and resources, respectively that have good historical performance.

If a job is accepted for execution on a resource, it is allocated to the processor with the earliest availability and the new availability is computed as follows:

$$NA = TPF + \frac{JL}{PS} + TIO \tag{5.19}$$

Where NA is the new availability, TPF is the time when the processor with the earliest availability is going to be free, JL is the job length, PS is the processor's speed measured in MIPS and TIO is the time needed for input and output.

5.4 Enhancing simulation speed

In the early version of the event-driven simulator, the events which are ordered by occurrence time were added to the event list (object of Java class Vector) using the Java interface Comparator which is:

"*A comparison function, which imposes a total ordering on some collection of objects*" [85].

However, the running time of each scenario was so long and it was taking at least hours to finish. By using the profile option, it was discovered that the most time consuming method was the method for adding events to the event list. In order to improve the running speed, a binary insert algorithm is employed to reduce the time needed to add an event. Binary insert is a modified version of well known binary search. After the improvement, the running time of each scenario was reduced to only couple of minutes. Pseudocode 10 shows how the binary insert works.

The method works as follows: First, it checks if the event list is empty (line

4). If it is, it just adds the event to the event list (line 5). Otherwise, it starts searching (using the same technique of binary search) for an event in the list that has the same occurrence time (lines 7 to 17). If there is such event, it inserts the new event directly after the found event (middle + 1), and shifts the event currently at that position and any subsequent events to the right (line 12). If there is no such event, the event where the binary insert has stopped should have the closest (or second closest) occurrence time to the occurrence time of the new event. In this case, the method compares the occurrence time of the new event to the occurrence time of the event where the method has stopped (lines 19 to 23). If the occurrence time of the new event is less (line 19), it inserts it at the position of that event (middle), and shifts the event currently at that position and any subsequent events to the right (line 20). If the occurrence time of the new event is higher (line 21), it inserts it directly after that event (middle + 1) and shifts the event currently at that position and any subsequent events to the right (line 22).

Pseudocode 10: Binary insert.

```
1  low = 1;
2  high = eventlist.size();
3  added = false;
4  if eventlist is empty then
5  |  eventlist.add(1, eventtoadd);
6  |  added = true;
7  else
8  |  while low ≤ high and added = false do
9  |  |  middle = (low + high)/2;
10 |  |  event = eventlist.get(middle);
11 |  |  if newevent.time = event.time then
12 |  |  |  eventlist.add(middle + 1, newevent);
13 |  |  |  added = true;
14 |  |  else if newevent.time < event.time then  high = middle − 1;
15 |  |  else  low = middle + 1;
16 |  end
17 end
18 if added = false then
19 |  if newevent.time < event.time then
20 |  |  eventlist.add(middle, newevent);
21 |  else if newevent.time > event.time then
22 |  |  eventlist.add(middle+1, newevent);
23 |  end
24 end
```

5.5 Simulation verification

In this section, the simulator is verified by showing that the obtained output which is shown in appendices A to D matches with the results obtained by hand

calculations. For comparison purposes, each line in the appendix is given a number for reference. Four scenarios are considered to cover various designed strategies.

5.5.1 First verification scenario

In the simulated scenario (see appendix A for output), there are three users (U_1, U_2 and U_3) , one broker (B_1) and one resource (R_1). U_1, U_2 and U_3 employ Price, Completion_time and Price-Completion_time strategies, respectively. B_1 employs User-Resource_price_difference strategy, while R_1 employs Price strategy. Furthermore, the parameters are set as follows: For users, both initial price and deadline determinators equal 0.6, while both increment in price and deadline determinators equal 0.1. Additionally, both maximum acceptable price per MI and maximum price to send per MI equal 0.01 and both maximum acceptable completion time and maximum deadline to send determinators equal two. The revenue determinator equals 0.2 for B_1, whilst the minimum acceptable price per MI equals 0.006 for R_1. The cost per MI and cost per time unit which are deducted from the resource revenue are equal to 0.002 and one, respectively. On the other hand, the cost per MI and cost per time unit which are deducted from the broker revenue are equal to 0.0005 and 0.25, respectively.

Each user has two jobs (J_{i1} and J_{i2}) with lengths equal 60000 MI each, while the resource has three processors (P_{11}, P_{12} and P_{13}) with speeds equal 300 MIPS each. The jobs belong to U_1 arrive at times 3 and 23. The jobs belong by U_2 arrive at times 103 and 123. Finally, jobs of U_3 arrive at times 203 and 223.

The waiting time for every user equals the current time plus 10 time units, while it equals the current time plus 5 time units for every broker. Furthermore, each entity needs at least one time unit to respond (time for a request or bid to reach an entity).

In the appendices, the names of the users are User_0, User_1 and User_2 and the names of the jobs belong to each of the users are Job_0 and Job_1. Furthermore, the name of the broker is Broker_0 and the name of the resource is Resource_0.

In order for a user request to be accepted by the resource, the sent broker price to the resource must be greater than or equal to the resource minimum acceptable price. This is because the resource employs price strategy. The resource minimum acceptable price is computed as in equation 5.16:

$RMAP = RMAPPMI \times JL = 0.006 \times 60000 = 360$

Since the user maximum acceptable price per MI is set to 0.01, the user maximum acceptable price is computed as in equation 5.2:

$MAP = MAPPMI \times JL = 0.01 \times 60000 = 600$

The initial price in the first round of negotiation is equal to 360 which is computed as in equation 5.1:

$IP = IPD \times MAP = 0.6 \times 600 = 360$

And the sent price from the broker to the resource is equal to 288 which is computed as in equation 5.14:

$PSTR = (1 - RD) \times UP = (1 - 0.2) \times 360 = 288 < RMAP$

The increment in price is equal to 60 which is computed as in equation 5.3:

$IIP = IIPD \times MAP = 0.1 \times 600 = 60$

So, the user price in the second round is equal to 420.

And the sent price from the broker to the resource is equal to 336.

$PSTR = 0.8 \times 420 = 336 < RMAP$

The user price in the third round is equal to 480.

And the sent price from the broker to the resource is equal to 384.

$PSTR = 0.8 \times 480 = 384 \geq RMAP$

It can be seen in line 4 that the submission time of J_{11} equals to 27. It can be seen above that each user needs three rounds in order to make an agreement for each of its jobs. The user starts the negotiation at time (t) = 3 (start of the first round). The broker takes an action after 1 time unit (t=4). Then, the resource takes an action at t=5 and decides not to reply. Next, the broker waits until its waiting time is reached, because it hasn't received a bid from the resource (t=9) and decides not to reply as well. Then, the user waits until the waiting time is reached too, because of the same reason (t=13) and then takes its action (start

of the second round). This is followed by an action by the broker at t=14. Then, the resource takes an action at t=15 and decides not to reply in this round too. Again, the broker waits until its waiting time is reached (t=19). Next, the user waits until its waiting time is reached at t = 23 and takes his action (start of the third round). Then, the broker takes an action at t = 24. Next, the resource decides to reply at t = 25 because its price threshold is met. The broker takes an action at t = 26 because it received an interesting bid this time and sends a bid to the user. The user accepts the broker's bid at t = 27. So, the time spent in negotiation equals to 24 (27-3). This occurrence of events is the same for other jobs except that the starting time is different.

So it can be concluded that the submission time of each job is computed as follows:

$ST = STON + 24$

Where ST is the submission time and $STON$ is the start time of the negotiation.

So, the submission times of users' jobs:

$J_{11} = 3 + 24 = 27$

$J_{12} = 23 + 24 = 47$

$J_{21} = 103 + 24 = 127$

$J_{22} = 123 + 24 = 147$

$J_{31} = 203 + 24 = 227$

$J_{32} = 223 + 24 = 247$

These match the output of simulation in lines 4, 13, 22, 31, 40 and 49.

Please note that for J_{22} and J_{32}, the resource sends completion times greater than their deadlines. But, the completion times are less than or equal to the users' maximum acceptable completion times. This has no effect in this scenario, but this has an effect in the scenario in the next section when the broker employs User-Resource_price_difference-Deadline_met strategy. This is because the broker won't send a bid to the user, if the received resource completion time is greater

than the user's deadline. So, the user has to increase its deadline and go through another round of negotiation in order for the resource completion time to be less than or equal to its deadline.

So, the resource sends a bid after the third round to the broker, and which in turn sends a bid to the user. Finally, the user accepts the broker's bid.

After an agreement is made:

- the user is charged 480 as in lines 3, 12, 21, 30, 39 and 48.
- the broker gets 96 as revenue, but the profit after executing the job is equal to 66.

  $Profit = BR - (CPMI \times JL) = 96 - (0.0005 \times 60000) = 66$

  This matches the output of simulation in line 6. For the next jobs, the profit is increased by 66 as in lines 15, 24, 33, 42 and 51.
- the resource gets 384 as revenue, but the profit is equal to 264. For the next jobs, the profit is increased by 264 as in lines 17, 26, 35, 44 and 53.

  $Profit = RR - (CPMI \times JL) = 384 - (0.002 \times 60000) = 264$

  This matches the output of simulation in line 8.

So, the overall profit generated by the broker before cost per time unit (CPTU) is deducted is computed as follows:

$Profit = (PGSJ \times NOJSB) = 66 \times 6 = 396$

Where $PGSJ$ is the profit generated from submitting the job and $NOJSB$ is the number of jobs submitted by the broker.

This matches the output of simulation in line 51.

While the overall profit generated after cost per time unit ($CPTU$) is deducted is computed as follows (see table 5.1):

$Profit = (PGSJ \times NOJSB) - (CPTU * TAES) = (66 \times 6) - (0.25 \times 529)$
$= 263.75$

Where $TAES$ is the time at the end of the simulation.

This matches the output of simulation in line 56.

On the other hand, the overall profit generated by the resource before cost per time unit (CPTU) is deducted is computed as follows:

$Profit = (PGEJ \times NOJER) = 284 \times 6 = 1584$

Where $PGEJ$ is the profit generated from executing the job and $NOJER$ is the number of jobs executed on the resource.

While the overall profit generated after $CPTU$ is deducted is computed as follows (see table 5.1):

This matches the output of simulation in line 53.

$Profit = (PGEJ \times NOJER) - (CPTU * TAES) = (284 \times 6) - (1 \times 529) = 1055$

This matches the output of simulation in line 58.

Because each user has 2 jobs, the overall cost paid by each user equals to 960. This match the output of simulation in lines 72 to 74.

The first user metric (job success rate) is computed as follows (see table 5.1):

$JSR = \frac{n}{tn} \times C1 = \frac{2}{2} \times 10 = 10$

Where JSR is the job success rate.

This matches the output of simulation in lines 88 to 90.

The second user metric (average cost per MI) is computed as follows (see table 5.1):

$ACPMI = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} L_i} \times C2 = \frac{480+480}{60000+60000} \times 1000 = 8$

Where $ACPMI$ is the average cost per MI.

This matches the output of simulation in lines 92 to 94.

Finally, the average satisfaction rate per job (third user metric) is different for different users. This is obviously because of the execution of their jobs at different times. In order to compute this metric, the initial deadlines and real completion times must be computed first.

In order to compute the initial deadlines, maximum deadlines to send for jobs belong to U_1 and maximum acceptable completion times for other users must be computed first as in equations 5.9 and 5.5:

$$J_{11} = CT + (MDTSD \times \frac{JL}{RMIPS}) = 3 + (2 \times \frac{60000}{300}) = 403$$
$$J_{12} = 23 + (2 \times \frac{60000}{300}) = 423$$
$$J_{21} = CT + (MACTD \times \frac{JL}{RMIPS}) = 103 + (2 \times \frac{60000}{300}) = 503$$
$$J_{22} = 123 + (2 \times \frac{60000}{300}) = 523$$
$$J_{31} = 203 + (2 \times \frac{60000}{300}) = 603$$
$$J_{32} = 223 + (2 \times \frac{60000}{300}) = 623$$

The initial deadlines of the jobs belong to U_1 are computed as in equation 5.7, while the initial deadlines of other jobs are computed as in equation 5.4:

$$J_{11} = IDD \times (MDTS - CT) + CT = 0.6 \times (403 - 3) + 3 = 243$$
$$J_{12} = 0.6 \times (423 - 23) + 23 = 263$$
$$J_{21} = IDD \times (MACT - CT) + CT = 0.6 \times (503 - 103) + 103 = 343$$
$$J_{22} = 0.6 \times (523 - 123) + 123 = 363$$
$$J_{31} = 0.6 \times (603 - 203) + 203 = 443$$
$$J_{32} = 0.6 \times (623 - 223) + 223 = 463$$

The real completion times of the jobs are computed as in equation 5.19:

$$J_{11} = TPF + \frac{JL}{PS} + TIO = 27 + \frac{60000}{300} + 1 = 228$$
$$J_{12} = 47 + \frac{60000}{300} + 1 = 248$$
$$J_{21} = 127 + \frac{60000}{300} + 1 = 328$$
$$J_{22} = 228 + \frac{60000}{300} + 1 = 429$$
$$J_{31} = 248 + \frac{60000}{300} + 1 = 449$$
$$J_{32} = 328 + \frac{60000}{300} + 1 = 529$$

Now, the metrics are computed as follows (see table 5.1):

$$ASRPJ - U_1 = \frac{\sum_{i=1}^{n} ID_i - CT_i}{n} \times C3 = \frac{(243-228)+(263-248)}{2} \times 1 = 15$$
$$ASRPJ - U_2 = \frac{(343-328)+(363-429)}{2} \times 1 = -25.5$$
$$ASRPJ - U_3 = \frac{(443-449)+(463-529)}{2} \times 1 = -36$$

Where *ASRPJ* is the average satisfaction rate per job.

These match the output of simulation in lines 96 to 98.

Figure 5.2 shows the assignment of jobs to processors in this scenario.

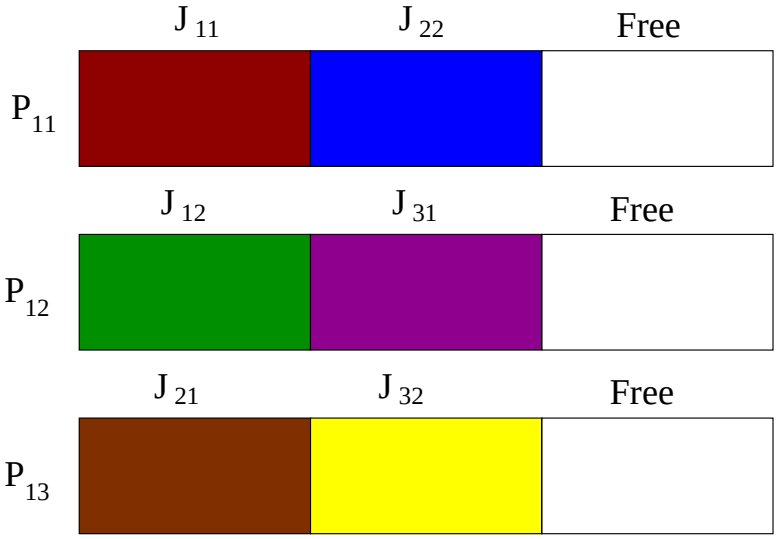

Figure 5.2: Assignment of jobs to processors in the first verification scenario.

5.5.2 Second verification scenario

This scenario (see appendix B for output) is the same as the first scenario except the broker employs User-Resource_price_difference-Deadline_met strategy instead of User-Resource_price_difference strategy. The change of broker strategy has an effect on profits generated by the broker and resource from executing J_{22} and J_{32} and therefore the overall profits generated by the broker and the resource. This has also an effect on the price paid for those two jobs and therefore on average cost per MI of those users.

For J_{22}, the maximum acceptable completion time in the first round is computed as in equation 5.5:

$MACT = CT + (MACTD \times \frac{JL}{RMIPS}) = 123 + (2 \times \frac{60000}{300}) = 523$

The initial deadline is computed as in equation 5.4:

$ID = IDD \times (MACT - CT) + CT = 0.6 \times (523 - 123) + 123 = 363$

The maximum acceptable completion time in the second round is computed as follows:

$MACT = 133 + (2 \times \frac{60000}{300}) = 533$

The increment in deadline is computed as in equation 5.6:

$IID = IIDD \times (MACT - CT) + (CT - TPR) = 0.1 \times (533 - 133) + (133 - 123)$ $= 50$

Since the time between any two successive rounds is the same (ten time units), the increment in deadline is the same in all rounds.

Thus, the deadline in the third deadline is computed as follows:

$Deadline = ID + (2 \times IID) = 363 + (2 \times 50) = 463$

The earliest possible completion time ($EPCT$) the resource can afford based on the availability of its processors is computed as in equation 5.18:

$EPCT = TPF + \frac{JL}{MIPS} + TIO + TTRB = 228 + \frac{60000}{300} + 1 + 1 = 430$

The resource makes the check mentioned in the listing in section 5.3 for resources that employ Price strategy and observes that:

$Deadline < ECT + (0.3 \times (ECT - CT)) = 430 + (0.3 \times (430 - 145)) = 515.5$

Because the deadline is less than 515.5, the resource sends completion time equals to 515.5.

The broker doesn't reply to user request because the resource completion time is greater than the user deadline. So, the user waits until its waiting time is reached and issues a new request with a new deadline and price (start of the fourth round).

$Deadline = 463 + 50 = 513$

$Price = 480 + 60 = 540$

Now

$Deadline \geq ECT + (0.3 \times (ECT - CT)) = 430 + (0.3 \times (430 - 155)) = 512.5$

So, the resource sends a completion time that is equal to the user deadline and therefore the broker replies to the user request. Then, the user accepts the broker's bid and is charged 540 as shown in line 30.

The negotiation of J_{32} goes through the same steps except the occurrence times are different and thus the user is charged 540 too (see line 48).

5.5.3 Third verification scenario

This scenario (see appendix C for output) has the same parameters of the first scenario except the resource employs Deadline strategy instead of Price strategy. Minimum acceptable deadline determinator equals 0.6.

In this scenario, the resource replies when the resource minimum acceptable deadline is less than or equal to the users deadline.

For J_{11}, the maximum deadline to send ($MDTS$) in the first round is computed as in equation 5.9:

$MDTS = CT + (MDTSD \times \frac{JL}{RMIPS}) = 3 + (2 \times \frac{60000}{300}) = 403$

The initial deadline is computed as in equation 5.7:

$ID = IDD \times (MDTS - CT) + CT = 0.6 \times (403 - 3) + 3 = 243$

In order to compute the resource minimum acceptable deadline (MAD), earliest possible completion time for the job ($EPCT$) must be computed first as in equation 5.18:

$EPCT = TPF + \frac{JL}{MIPS} + TIO + TTRB = 5 + \frac{60000}{300} + 1 + 1 = 207$

The resource MAD is computed as in equation 5.17:

$MAD = EPCT + MADD \times (EPCT - CT) = 207 + 0.6 \times (207 - 5) = 328.2 \nleq 243$

In the second round, $MDTS$ is computed as follows:

$MDTS = 13 + (2 \times \frac{60000}{300}) = 413$

The increment in deadline (IID) is computed as in equation 5.6:

$IID = IIDD \times (MDTS - CT) + (CT - TPR) = 0.1 \times (413 - 13) + (13 - 3)$
$= 50$

Since the time between any two successive rounds is the same (ten time units), the increment in deadline is the same in all rounds.

So, the user's deadline equals 293.

$Deadline = ID + IID = 243 + 50 = 293$

For the resource, $EPCT$ is computed as follows:

$EPCT = 15 + \frac{60000}{300} + 1 + 1 = 217$

The resource MAD is computed as follows:

$MAD = 217 + 0.6 \times (217 - 15) = 338.2 \nleq 293$

In the third round, the user's deadline equals 343.

For the resource, $EPCT$ is computed as follows:

$EPCT = 25 + \frac{60000}{300} + 1 + 1 = 227$

The resource MAD is computed as follows:

$MAD = 227 + 0.6 \times (227 - 25) = 348.2 \nleq 343$

In the fourth round, the user's deadline equals 393.

For the resource, $MAD = 358.2 \leq 393$. So, the resource replies to the broker which in turn sends a bid to the user. The user then accepts the bid. From the first scenario, it can be seen that the user's initial price is 360 and the increment in price is 60. So, the user's price is 540 ($360+3\times60$) after four rounds of negotiation. This matches the output in line 3.

Tables 5.4, 5.5 and 5.6 shows the parameters' values sent through the negotiations for jobs belong to U_1, U_2 and U_3, respectively.

Table 5.4: Parameters' values sent through the negotiations of U_1.

J_{11}	
Round 1	$MDTS = 403$, $Deadline = 243$, $Price = 360$, $EPST = 207$ and $MAD = 328.2 \nleq ID$
Round 2	$MDTS = 413$, $Deadline = 293$, $Price = 420$, $EPST = 217$ and $MAD = 338.2 \nleq Deadline$
Round 3	$MDTS = 423$, $Deadline = 343$, $Price = 480$, $EPST = 227$ and $MAD = 348.2 \nleq Deadline$
Round 4	$MDTS = 433$, $Deadline = 393$, $Price = 540$, $EPST = 237$ and $MAD = 358.2 \leq Deadline$
J_{12}	
Round 1	$MDTS = 423$, $Deadline = 263$, $Price = 360$, $EPST = 227$ and $MAD = 348.2 \nleq ID$
Round 2	$MDTS = 433$, $Deadline = 313$, $Price = 420$, $EPST = 237$ and $MAD = 358.2 \nleq Deadline$
Round 3	$MDTS = 443$, $Deadline = 363$, $Price = 480$, $EPST = 247$ and $MAD = 368.2 \nleq Deadline$
Round 4	$MDTS = 453$, $Deadline = 413$, $Price = 540$, $EPST = 257$ and $MAD = 378.2 \leq Deadline$

It can be seen in table 5.5 that in the fifth round the deadline of J_{22} became

Table 5.5: Parameters' values sent through the negotiations of U_2.

J_{21}	
Round 1	$MACT = 503$, $Deadline = 343$, $Price = 360$, $EPST = 307$ and $MAD = 428.2 \nleq ID$
Round 2	$MACT = 513$, $Deadline = 393$, $Price = 420$, $EPST = 317$ and $MAD = 438.2 \nleq Deadline$
Round 3	$MACT = 523$, $Deadline = 443$, $Price = 480$, $EPST = 327$ and $MAD = 448.2 \nleq Deadline$
Round 4	$MACT = 533$, $Deadline = 493$, $Price = 540$, $EPST = 337$ and $MAD = 458.2 \leq Deadline$
J_{22}	
Round 1	$MACT = 523$, $Deadline = 363$, $Price = 360$, $EPST = 327$ and $MAD = 448.2 \nleq ID$
Round 2	$MACT = 533$, $Deadline = 413$, $Price = 420$, $EPST = 337$ and $MAD = 458.2 \nleq Deadline$
Round 3	$MACT = 543$, $Deadline = 463$, $Price = 480$, $EPST = 440$ and $MAD = 617 \nleq Deadline$
Round 4	$MACT = 553$, $Deadline = 513$, $Price = 540$, $EPST = 440$ and $MAD = 611 \nleq Deadline$
Round 5	$MACT = 563$, $Deadline = 563$, $Price = 600$, $EPST = 440$ and $MAD = 605 \nleq Deadline$

the same as the user $MACT$ without being greater than or equal to the resource MAD. That is why the job couldn't be executed.

Figure 5.3 shows the assignment of jobs to processors in this scenario.

Table 5.6: Parameters' values sent through the negotiations of U_3.

J_{31}	
Round 1	$MACT = 603$, $Deadline = 443$, $Price = 360$, $EPST = 440$ and $MAD = 581 \nleq ID$
Round 2	$MACT = 613$, $Deadline = 493$, $Price = 420$, $EPST = 440$ and $MAD = 575 \nleq Deadline$
Round 3	$MACT = 623$, $Deadline = 543$, $Price = 480$, $EPST = 440$ and $MAD = 569 \nleq Deadline$
Round 4	$MACT = 633$, $Deadline = 593$, $Price = 540$, $EPST = 440$ and $MAD = 563 \leq Deadline$
J_{32}	
Round 1	$MACT = 623$, $Deadline = 463$, $Price = 360$, $EPST = 440$ and $MAD = 569 \nleq ID$
Round 2	$MACT = 633$, $Deadline = 513$, $Price = 420$, $EPST = 460$ and $MAD = 595 \nleq Deadline$
Round 3	$MACT = 643$, $Deadline = 563$, $Price = 480$, $EPST = 460$ and $MAD = 589 \nleq Deadline$
Round 4	$MACT = 653$, $Deadline = 613$, $Price = 540$, $EPST = 460$ and $MAD = 583 \leq Deadline$

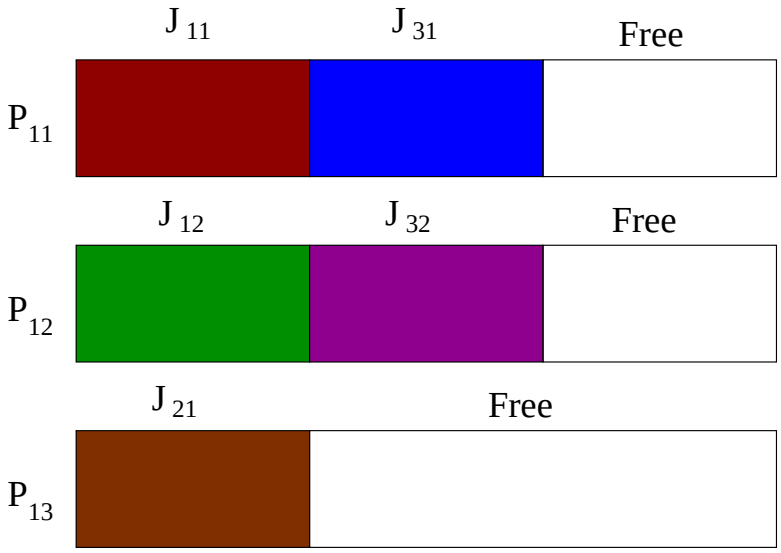

Figure 5.3: Assignment of jobs to processors in the third verification scenario.

5.5.4 Fourth verification scenario

Dissimilar to the first and third scenarios, the resource employs Price-Deadline strategy in this scenario (see appendix D for output). In Price-Deadline strategy, both user price and deadline must be greater than or equal to resource minimum acceptable price and deadline, respectively. It appears in the first and third scenarios that a larger number of rounds is needed for the resource minimum acceptable deadline (five rounds except J_{22} that fails to execute after the fifth round) to be met than the number of rounds to meet the resource minimum acceptable price (four rounds). Thus, both resources with Price-Deadline and Deadline strategies are going to go through the same number of rounds until they reply and therefore the output of this scenario is going to be the same as the output of the third scenario (see appendices C and D).

5.6 Results and discussion

5.6.1 Job progression

Figures 5.4 to 5.7 show the job submission, allocation, start and finish times of users with inter-arrival times follow negative exponential distribution with differ-

ent means, while figure 5.8 shows the job submission, allocation, start and finish times with dynamic submission (see section 3.2.2 for descriptions of these times). The x-axis shows the job number. For example, Job number 5 means the fifth job of all users. In this experiment, there are 108 users, 10 brokers and 27 resources. This simulation follows the configuration in section 5.3 except there is no strategy under consideration in this experiment. Each user has 10 jobs numbered from 0 to 9. It is observed from figures 5.4 to 5.7 that when the mean is increased the differences between job submission times get larger and therefore the system load gets lower. The differences between submission times are large when the submission is dynamic and thus the system load is low. It is also observed that the differences between the jobs' allocation and finish times become smaller when the mean is increased. The reason is that the job arrival rate becomes lower and therefore the resources are going to be less utilised. By having less utilised resources, the time needed to complete the execution of jobs is going to be small. It can be seen from the figures that the difference between the submission and allocation times of every job is very small. Thus, it can be concluded that the time spent in negotiation for every job is small if compared to the difference between start and finish times (execution time).

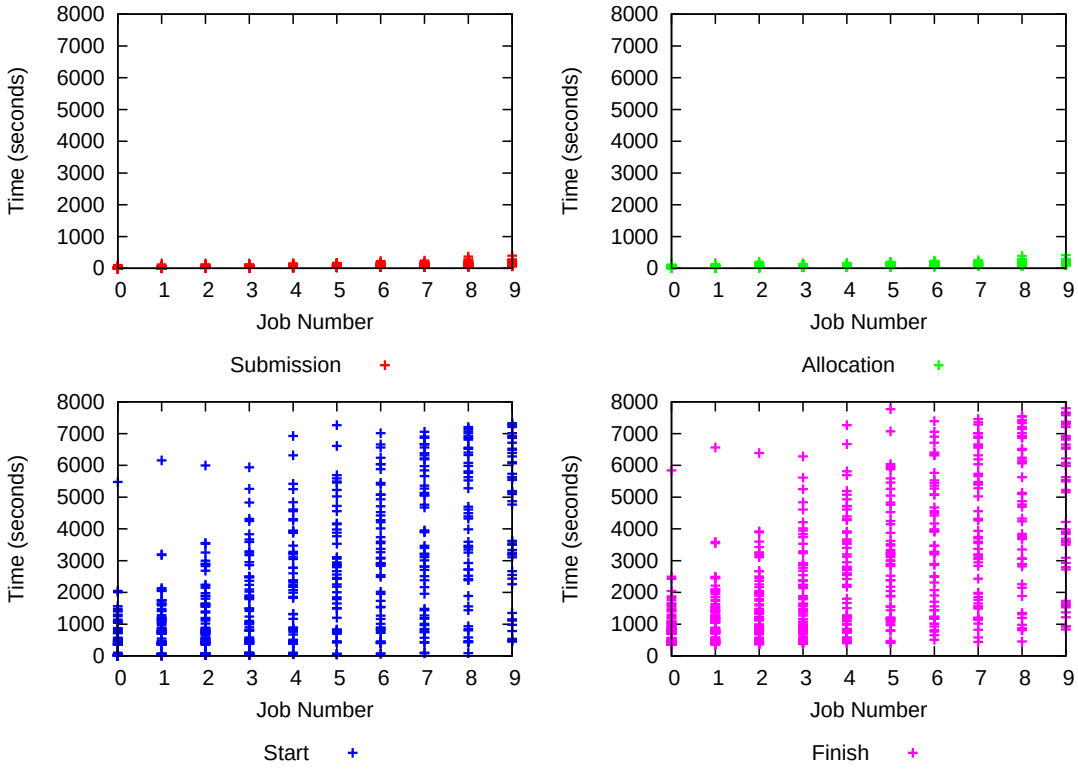

Figure 5.4: Job submission, allocation, start and finish times with inter-arrival times follow negative exponential distribution with mean 15.

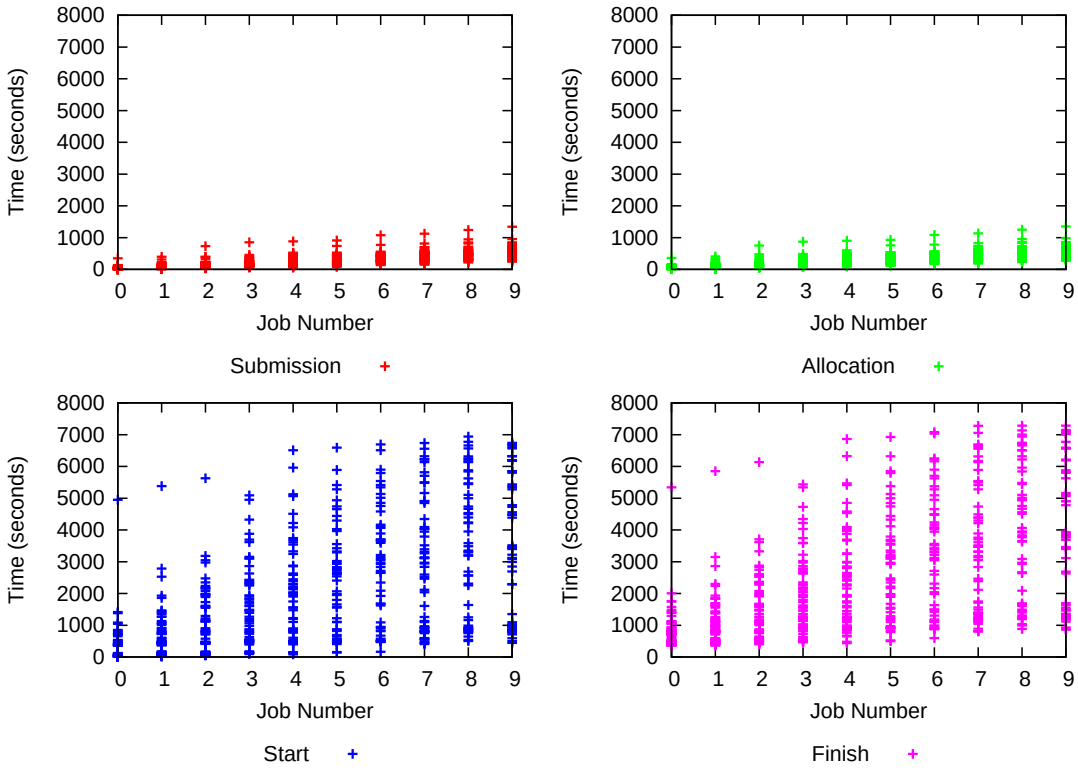

Figure 5.5: Job submission, allocation, start and finish times with inter-arrival times follow negative exponential distribution with mean 50

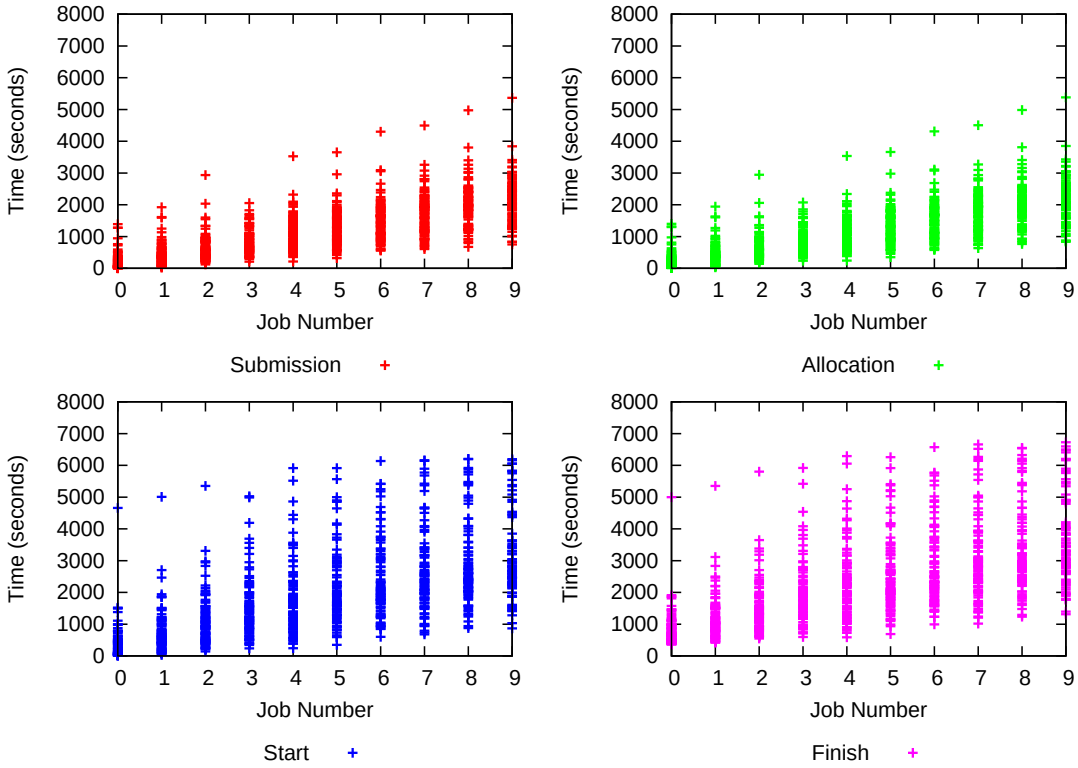

Figure 5.6: Job submission, allocation, start and finish times with inter-arrival times follow negative exponential distribution with mean 200.

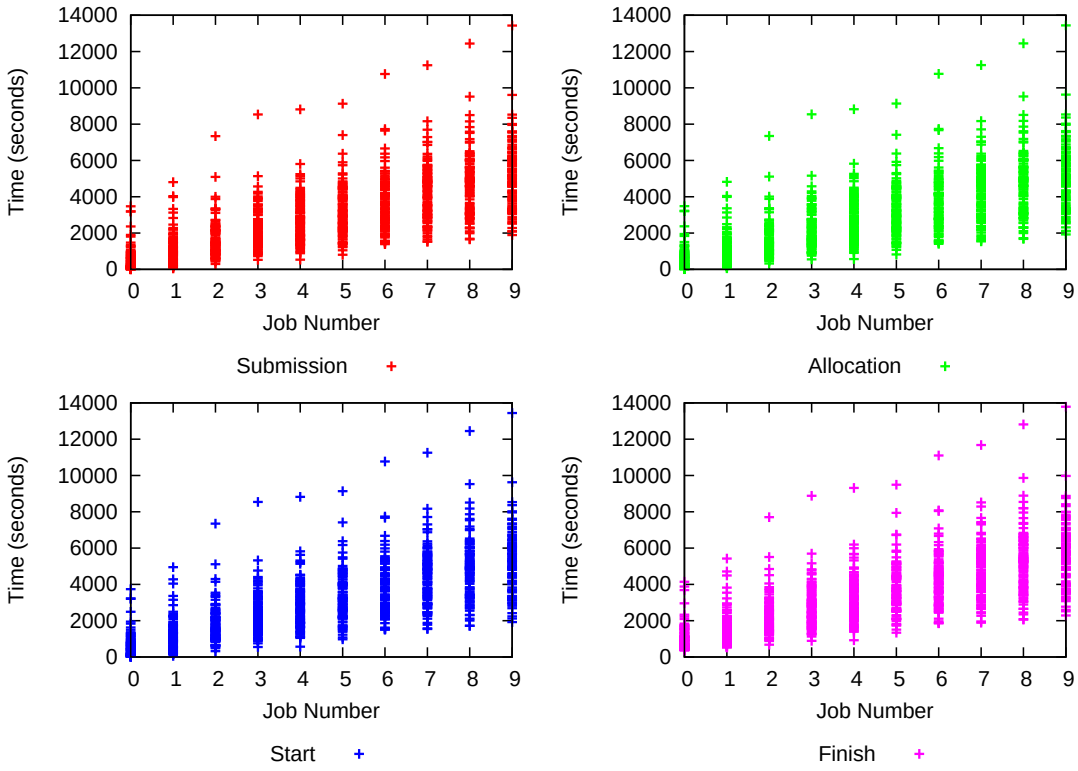

Figure 5.7: Job submission, allocation, start and finish times with job arrivals follow negative exponential distribution with mean 500.

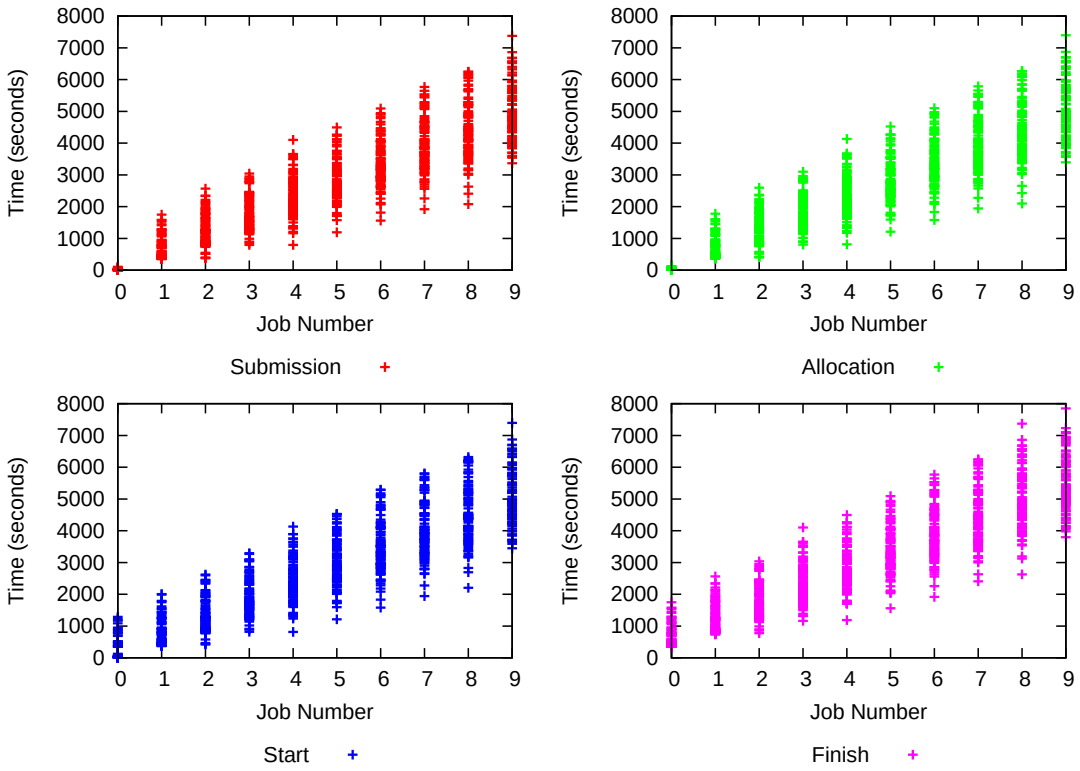

Figure 5.8: Job submission, allocation, start and finish times with dynamic submission.

In the next two sections, the performance of various strategies is measured with static and dynamic submission of jobs.

5.6.2 Static submission of jobs

In this experiment, we evaluate the performance of various entity strategies with different values of parameters that are shown in section 5.3. Figures 5.9 to 5.24 show the results when the submission of jobs is static. For the user strategy that is under consideration, the initial price and deadline determinators are varied from 0.1 to 1.0 in steps of 0.1, whilst increment in price and deadline determinators are varied from 0.01 to 0.4 (0.01, 0.05, 0.1, 0.2, 0.3 and 0.4). For the broker strategy that is under consideration, the revenue determinator is varied in the same way as initial price and deadline determinators. Finally, the two parameters that are varied for the resource strategy that is under consideration are the minimum acceptable price per MI which is varied between 0.001 and 0.01 in steps of 0.001 and minimum acceptable deadline determinator which is varied as revenue deter-

minator. Figures 5.9 to 5.20 show the performance of different users strategies based on 3 metrics: the job success rate, average cost per Million Instructions and average satisfaction rate per job. It is clearly seen in these figures that Completion_time and Price-Completion_time strategies have similar performance. This is because the simulated environment contains resources with various prices from cheap to expensive. Furthermore, the environment contains resources which don't care about the price paid for the execution. So, the price threshold can be easily met. However, the deadline threshold is going to have the large effect on accepting the bids, especially when the resources are highly utilised. In both strategies, both user price and deadline will be incremented in each round of negotiation until an agreement is made with the resource which its completion time suits the user deadline threshold.

It can be observed from figures 5.9, 5.12, 5.15 and 5.18 that Price strategy has the best job success rate, because it doesn't care when the jobs belong to the user who employs it will finish their execution. So, it is easy for the user to find suitable resources for executing its jobs. Moreover, it is observed that varying the determinators has no influence on the job success rate of Price strategy. This is due the existence of resources which have different prices from low to high and resources which are careless about the price paid for them in return of execution. For instance, if the initial price is low, the user executes (after negotiations) its jobs on cheap resources or resources that don't care about the price paid for execution. Otherwise, the user executes its jobs (or most of them) on expensive resources. Alternatively, the other two strategies have similar job success rate and which is worse than the job success rate of Price strategy. The reason is that when the resources are highly utilised, they start sending completion times that are greater than the maximum acceptable completion time of the users who employ these strategies. Therefore, those users neglect the resources' bids and this ends in not finding suitable resources for executing some of their jobs.

Four diagrams (figures 5.10, 5.13, 5.16 and 5.19) show that Price strategy has also the best average cost per MI. Because, this strategy won't accept bids with

prices greater than its price threshold. But, its average satisfaction rate per job is the worst as observed from figures 5.11, 5.14, 5.17 and 5.20. This is due to the fact that it doesn't care when the jobs will finish their execution. So, it considers any bid that meets its price threshold even if the completion time is so long. Although Price-Completion_time strategy uses price threshold to decide if to accept bids or not, its performance is worse (considering average cost per MI metric) than the performance of Price strategy. The reason is that it also works on finding a bid with completion time that meets its deadline threshold too, and therefore this usually results in paying higher price (but still less than or equal to its maximum acceptable price).

It is clearly seen in figure 5.11 that increasing the initial price tends to improve the average satisfaction rate per job of Price strategy. This is due to the ability of executing the jobs on more expensive and therefore less utilised resources. Less utilised resources can execute the jobs in less time and thus better satisfaction for jobs' deadlines.

Figure 5.9 shows that increasing the initial price also tends to increase the number of accepted jobs of users who employ Completion_time and Price-Completion_time strategies. This is because executing some of their jobs on more expensive resources that is usually less utilised than cheap resources and therefore can meet the deadline thresholds of the jobs. It is observed from figure 5.15 that increasing the initial deadline tends to increase the number of accepted jobs of users who employ the same strategies. The reason is that having relaxed deadlines for jobs results in having a larger chance to find suitable resources for executing them.

Also, it appears in figures 5.12 and 5.18 that having a larger increment in price or deadline in each round of negotiation tends to increase the number of accepted jobs of users that employ Completion_time and Price-Completion_time strategies. Because, the possibility of making an agreement with resources (through brokers) in less number of rounds is going to be higher. This reduction in number of rounds needed to find a suitable resource is important, because the resources have limitations (5 in the performed simulations) on the number of rounds they go

through for each job request.

It is observed from figure 5.13 that increasing the increment in price has nearly no influence on the average cost per MI of Price strategy, but has a large effect on the same metric of other user strategies. This is because Price strategy only uses price threshold to determine if to accept or reject a bid, and which can be satisfied in most cases from the first round of negotiation due to the existence of cheap resources or resources that are careless about the price they receive in return of execution. On the other hand, the users who employ other strategies have to initiate a number of rounds of negotiation until their deadline thresholds are met. For these strategies, the final price they reach until an agreement is made is going to be higher, when the increment in price is larger in each round.

It appears in figure 5.21 that User-Resource_price_difference strategy has better overall performance (generated more profit) than the other broker strategy. This is because it considers the resource bid even if its completion time is longer than the deadline of the user. This leaves for the users the decision to accept the bids with completion times longer than their deadlines. Thus, there is a larger probability to generate more profit, because some users are going to accept the bids with completion times longer than their deadlines. For instance, some users don't care when their jobs are expected to complete, or they prefer to continue negotiating until an agreement is made or they decide that they can't negotiate anymore. In the case of employing the other strategy, not considering the bid removes the possibility of negotiating the time needed for finishing the execution of the job from the first round of negotiation. It can be concluded from the same figure that the brokers' aim must not be getting too low revenue nor too high revenue from acting on behalf of users. If it is too low, more of their bids will be accepted by users, but they get small revenue. Hence, the profit will be small too after expenses deductions. On the other hand, if the revenue they aim to get is too high, the prices they send to resources will be too low (in order to keep the rest if their offers to resources are accepted) and in most cases the prices won't satisfy the resources' required profit from executing the jobs. This leads to rejecting most

of their requests to resources. As a result, the broker's objective has to be getting reasonable revenue.

Figure 5.23 shows that (resource) Price strategy usually generated more profit than Price-Deadline strategy except when the Minimum Acceptable Price Per Million Instructions (MAPPMI) equals 0.001. This is because Price strategy doesn't consider the sent deadline to determine if it has to send a bid or standby. As a result, it sends more bids and thus larger probability to generate more profit. When MAPPMI equals 0.001, Price-Deadline strategy has better performance than the performance of Price strategy, despite the resource which employs it has executed less number of jobs than the resource which employs Price strategy (see figure 5.22). This is because of the use of completion time threshold to determine if to send a bid or wait. This leads to increasing the users' prices and deadlines many times, until their deadlines become greater than or equal to the minimum acceptable deadline of Price-Deadline strategy. So, the price paid for the execution to the resource with Price-Deadline strategy is going to be higher than the price paid for the execution to the resource with Price strategy, in spite of having the same MAPPMI for both strategies. On the other hand, Deadline strategy is usually found to be inferior (see figure 5.24) when compared to Price-Deadline strategy and this results from not concerning the request sent price, when it decides to reply or not. Therefore, the resources which employ it is going to be the preferable choice for users who want only to spend a small amount of money on executing their jobs. This leads to generating small amount of revenue (profit) by the resources which employ it, because of the allocation of their processors to cheap jobs in the majority of cases.

Figure 5.24 also shows that increasing the minimum acceptable deadline determinator tends to decrease the profit generated by the two resources which employ Deadline and Price-Deadline strategies. Increasing this determinator increases the completion time threshold value as well. As a result, the possibility of users' deadlines to be greater than or equal to this threshold is going to be less and therefore less number of jobs will be executed on those resources.

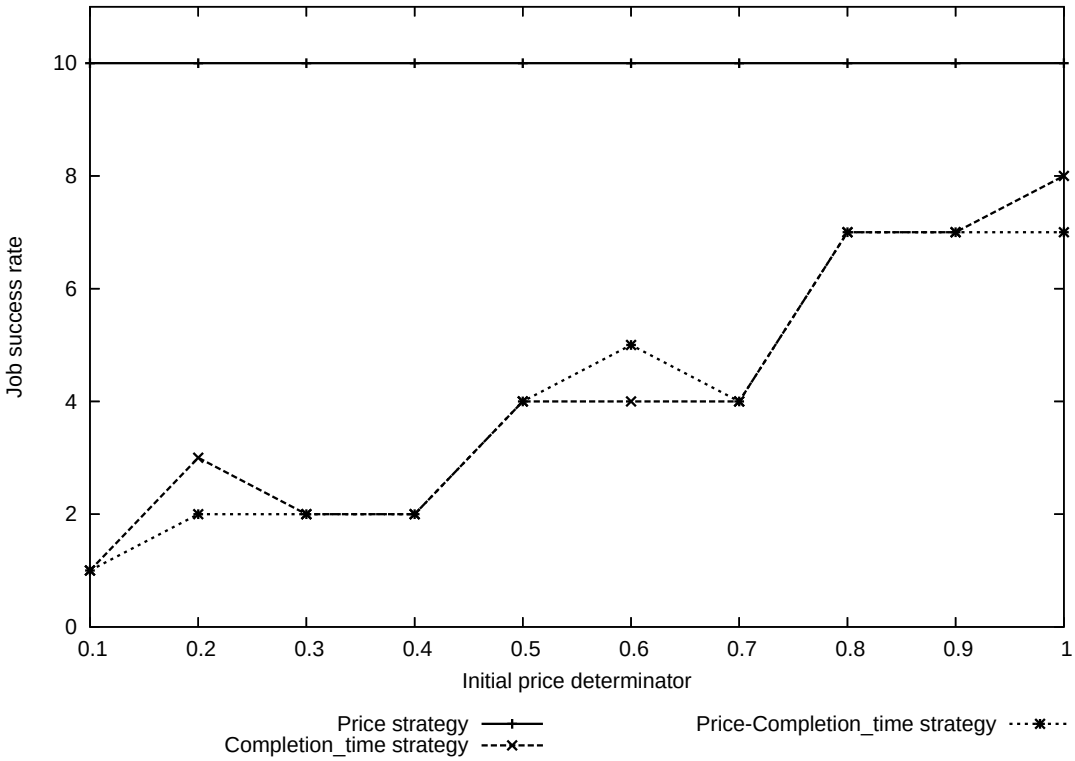

Figure 5.9: User strategy's job success rate for different initial price determinators with static submission.

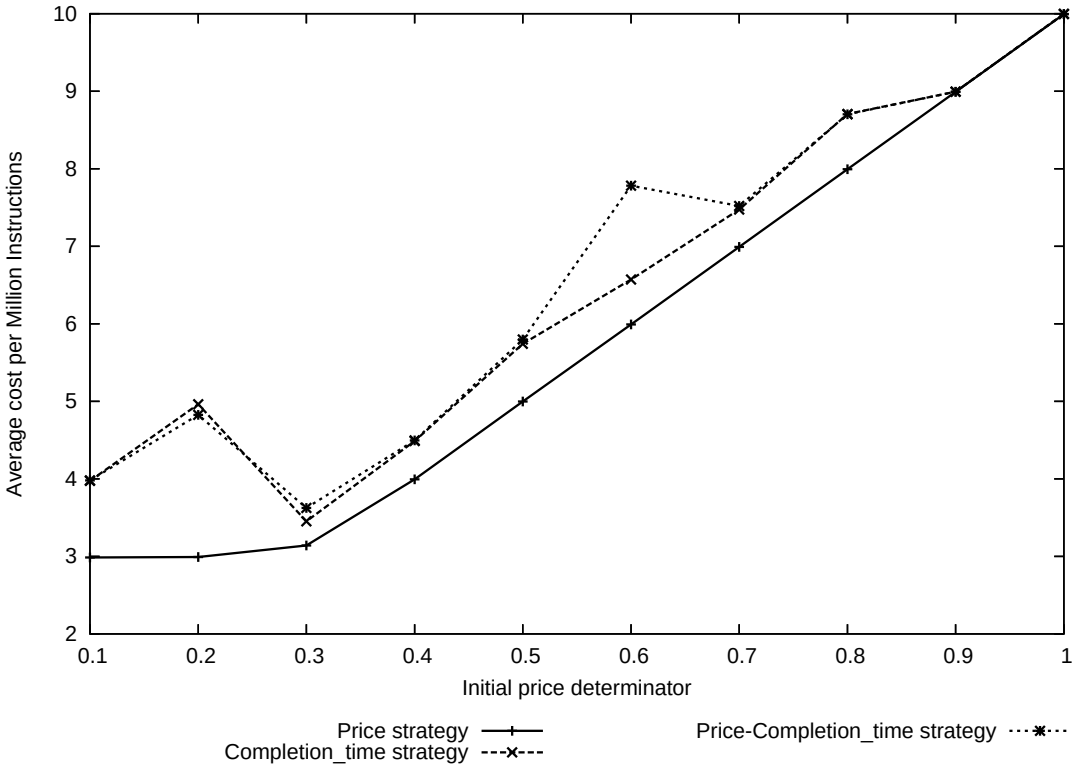

Figure 5.10: User strategy's average cost per Million Instructions for different initial price determinators with static submission.

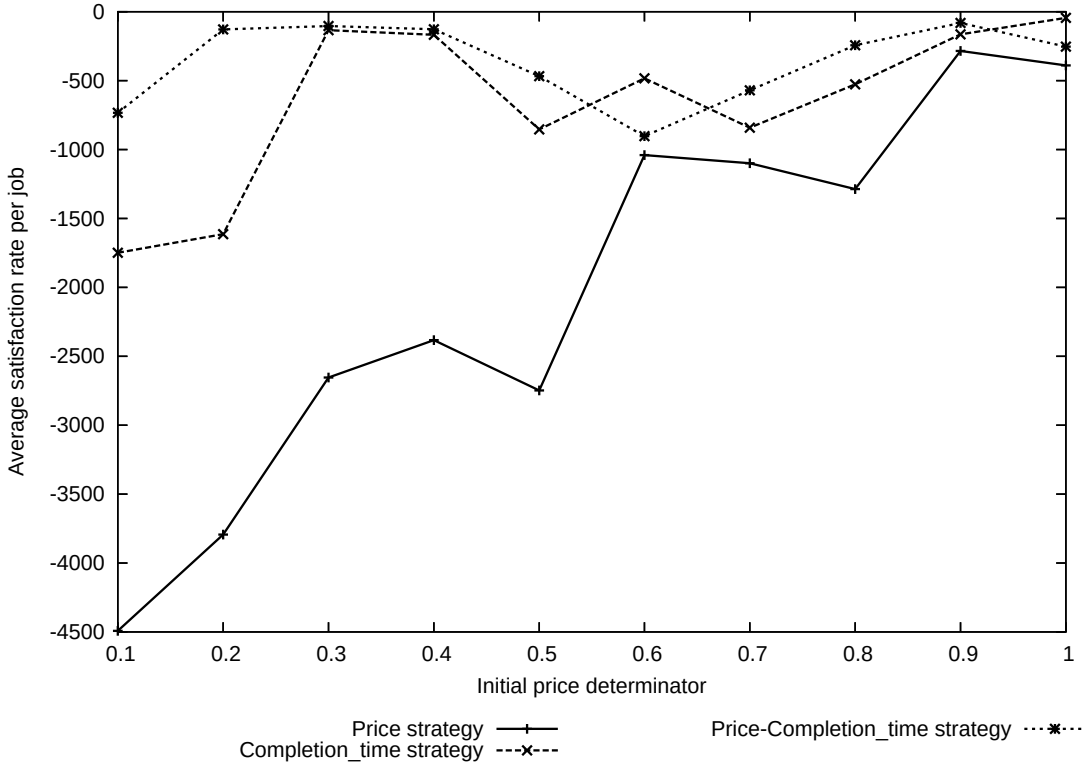

Figure 5.11: User strategy's average satisfaction rate per job for different initial price determinators with static submission.

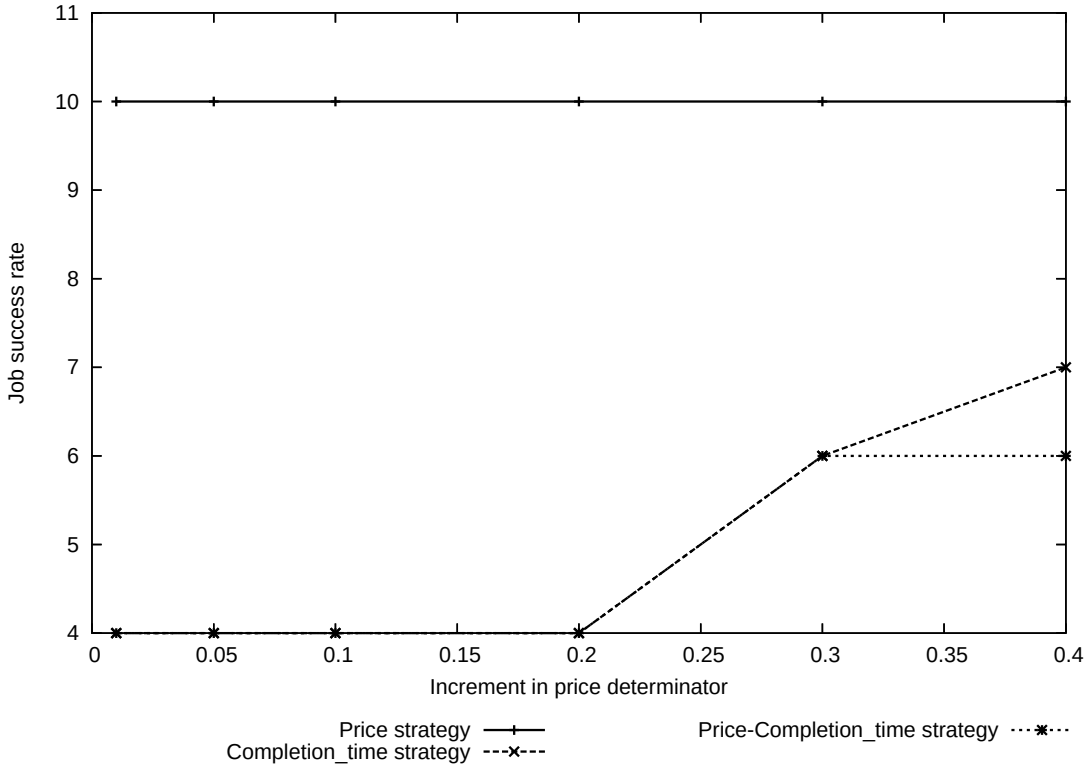

Figure 5.12: User strategy's job success rate for different increment in price determinators with static submission.

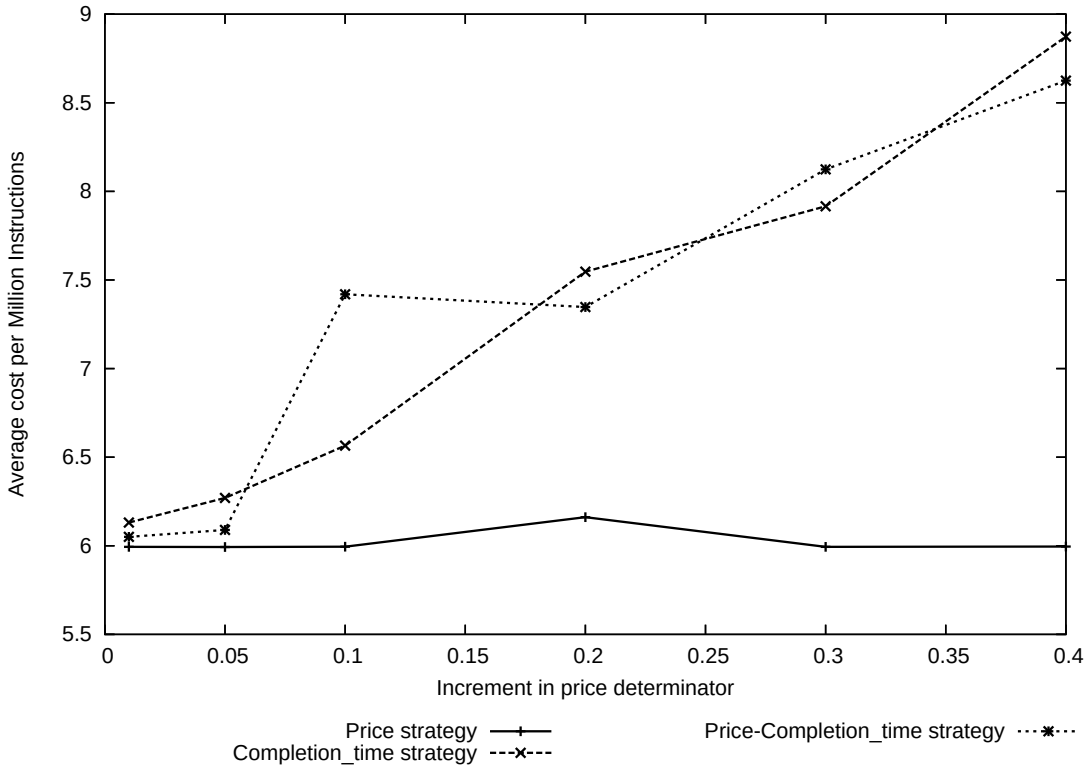

Figure 5.13: User strategy's average cost per Million Instructions for different increment in price determinators with static submission.

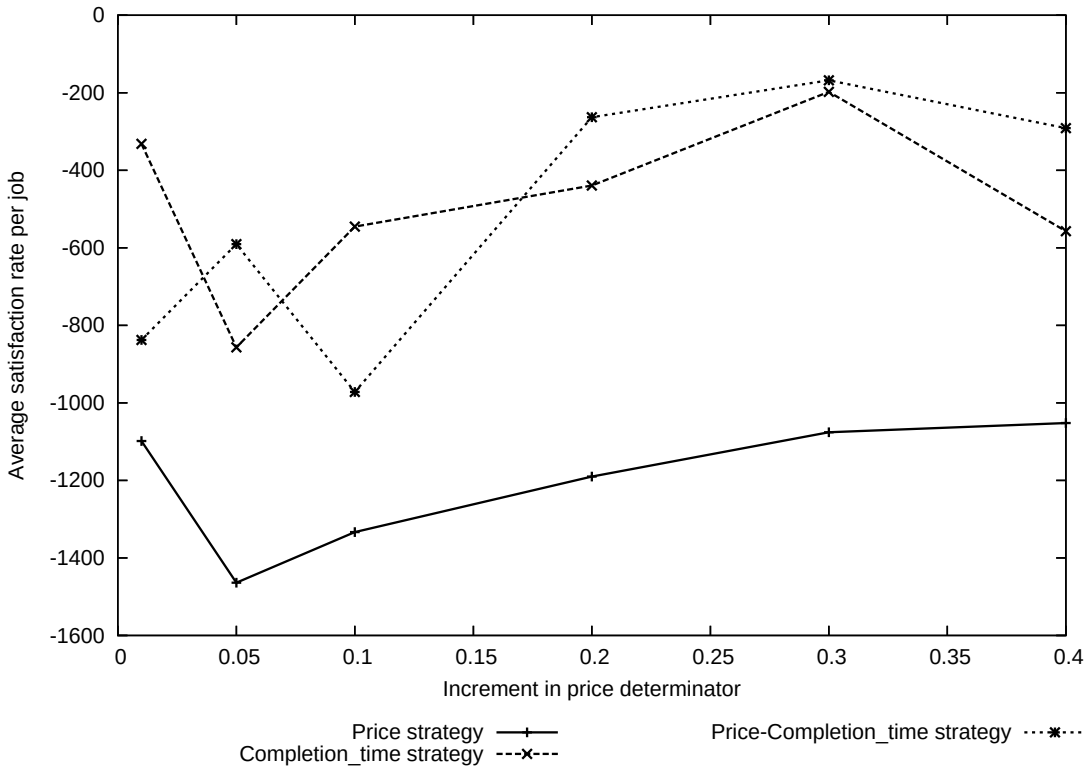

Figure 5.14: User strategy's average satisfaction rate per job for different increment in price determinators with static submission.

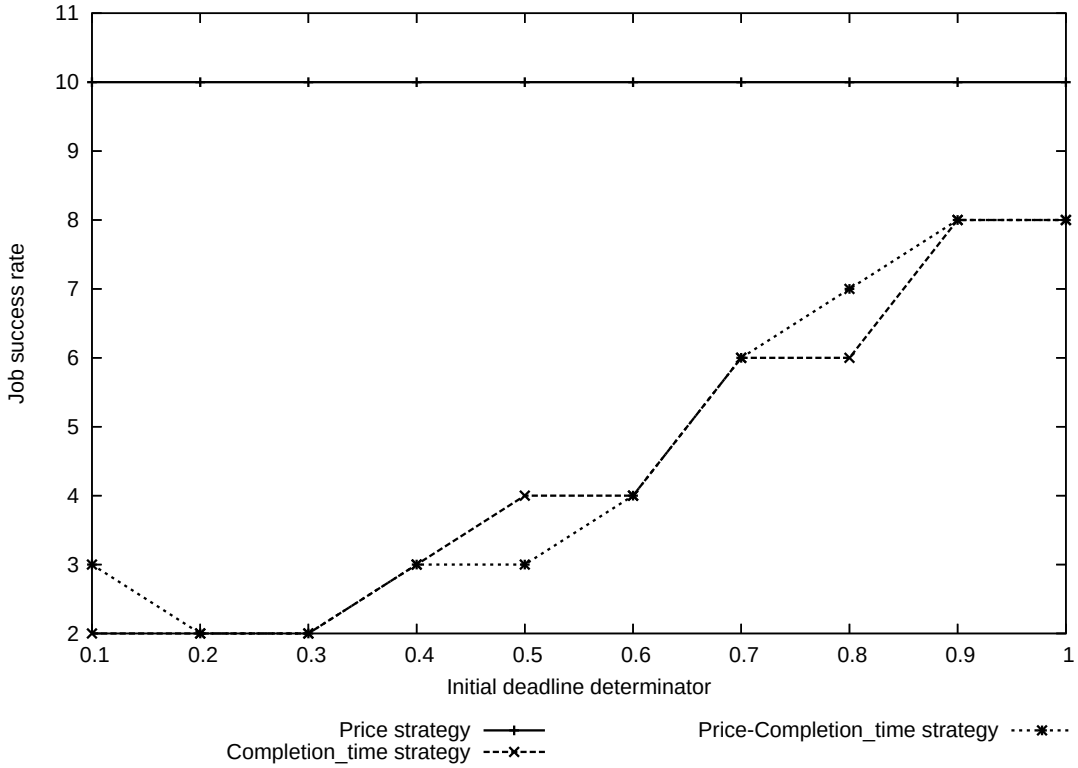

Figure 5.15: User strategy's job success rate for different initial deadline determinators with static submission.

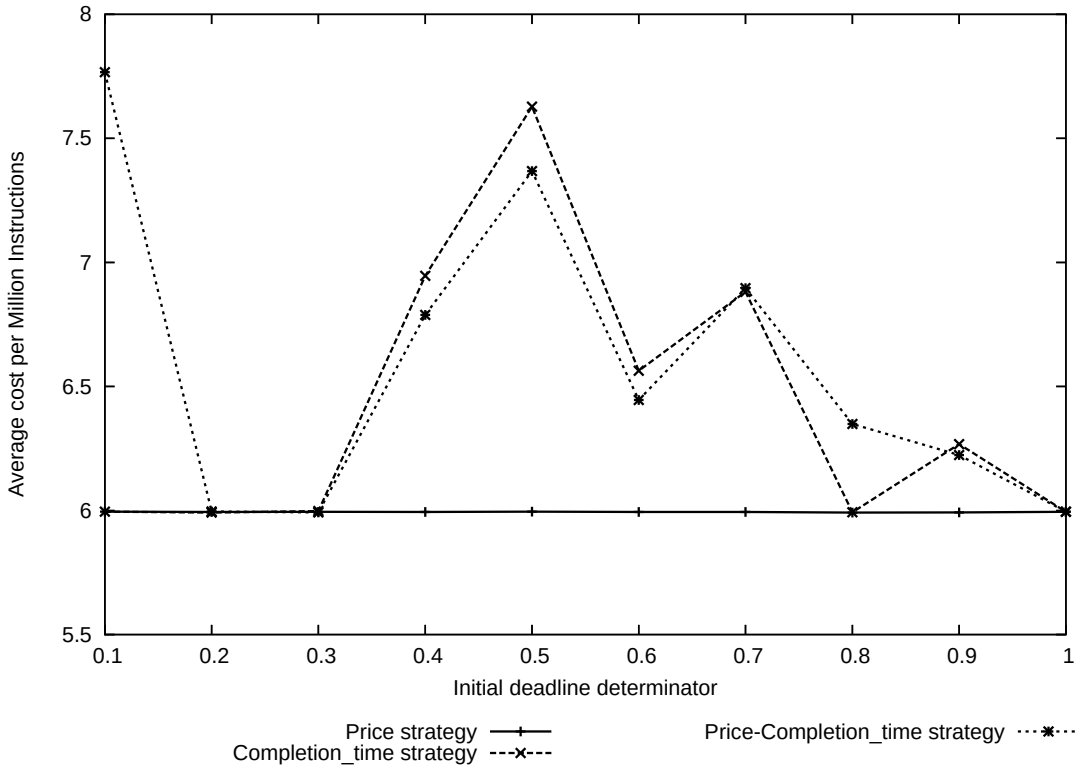

Figure 5.16: User strategy's average cost per Million Instructions for different initial deadline determinators with static submission.

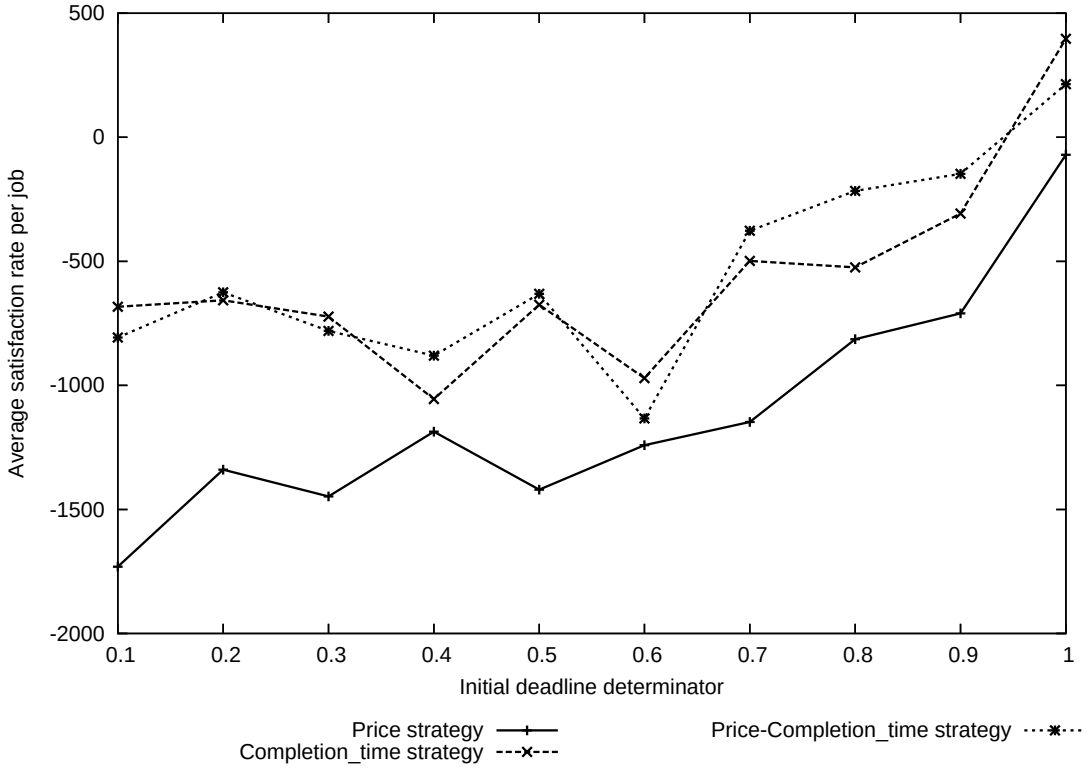

Figure 5.17: User strategy's average satisfaction rate per job for different initial deadline determinators with static submission.

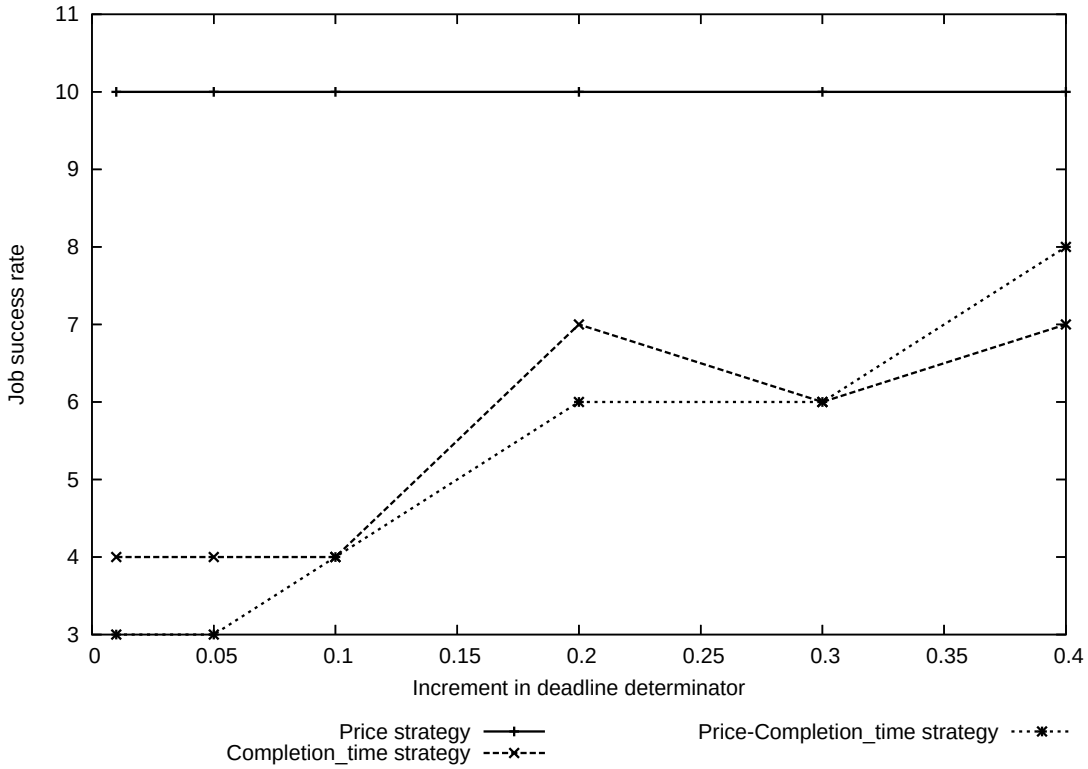

Figure 5.18: User strategy's job success rate for different increment in deadline determinators with static submission.

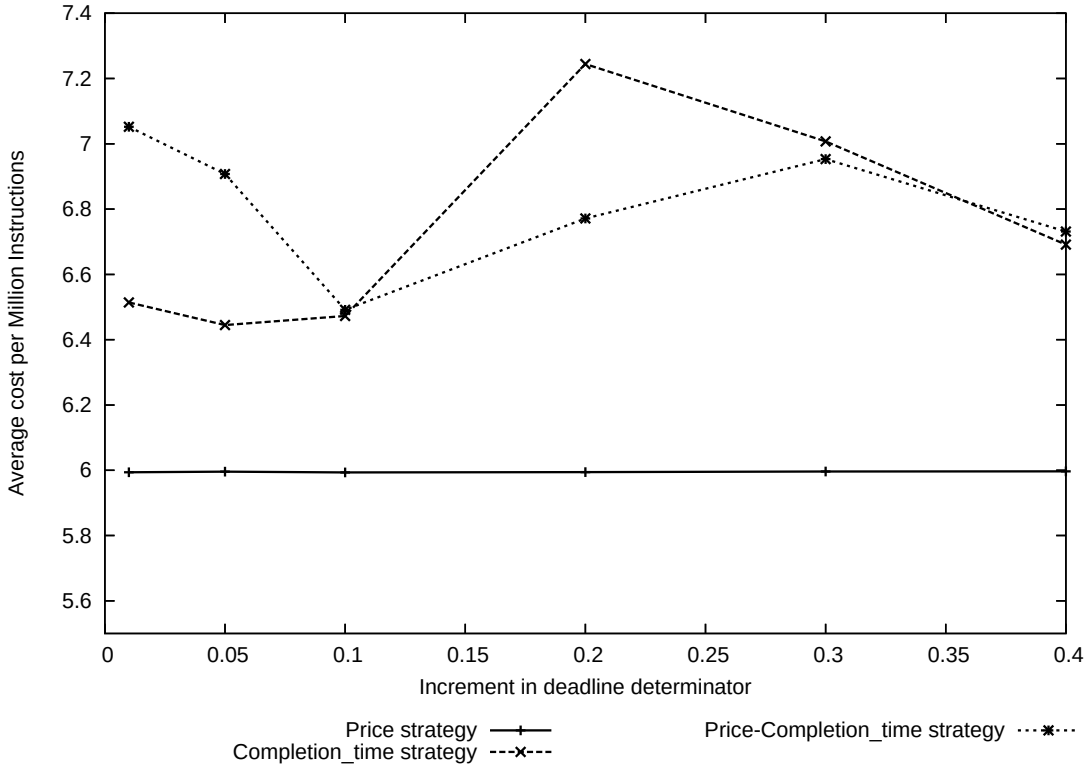

Figure 5.19: User strategy's average cost per Million Instructions for different increment in deadline determinators with static submission.

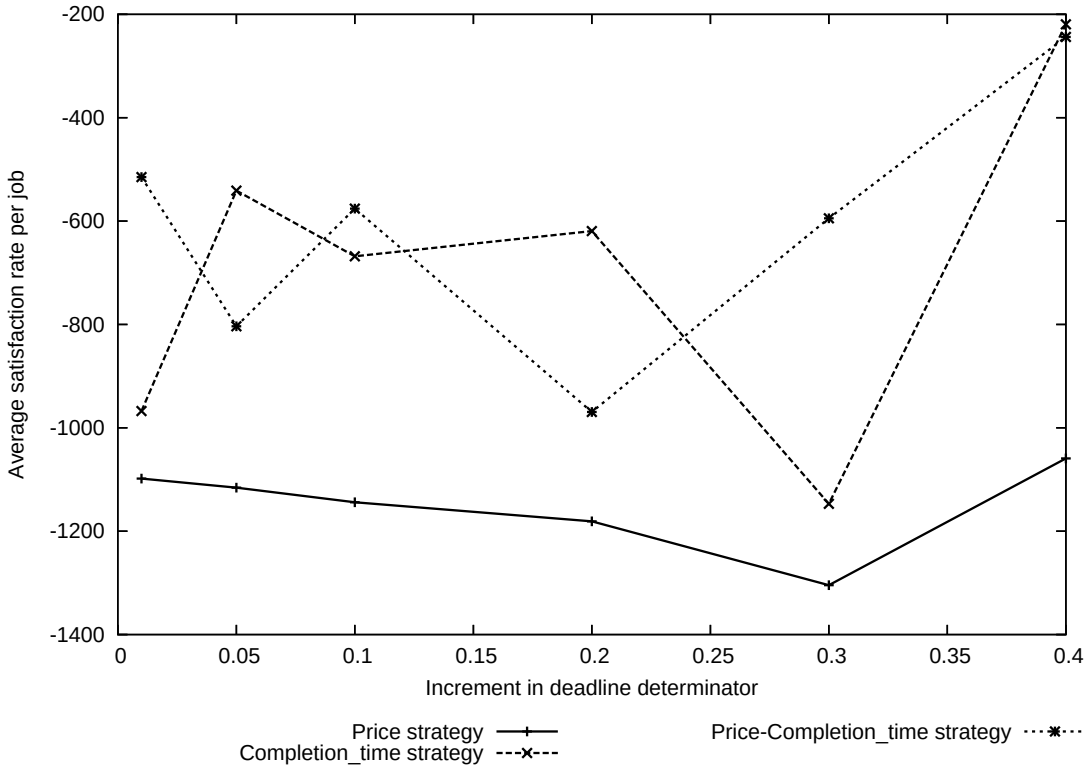

Figure 5.20: User strategy's average satisfaction rate per job for different increment in deadline determinators with static submission.

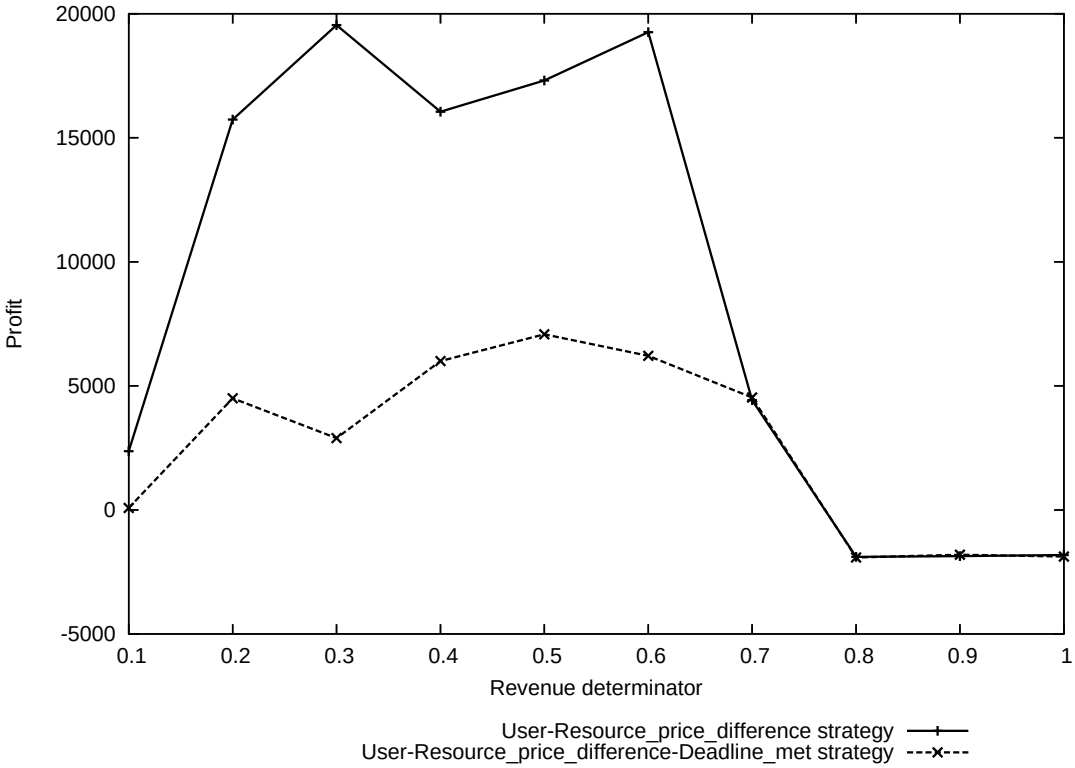

Figure 5.21: Profit generated by two broker strategies for different revenue determinators with static submission.

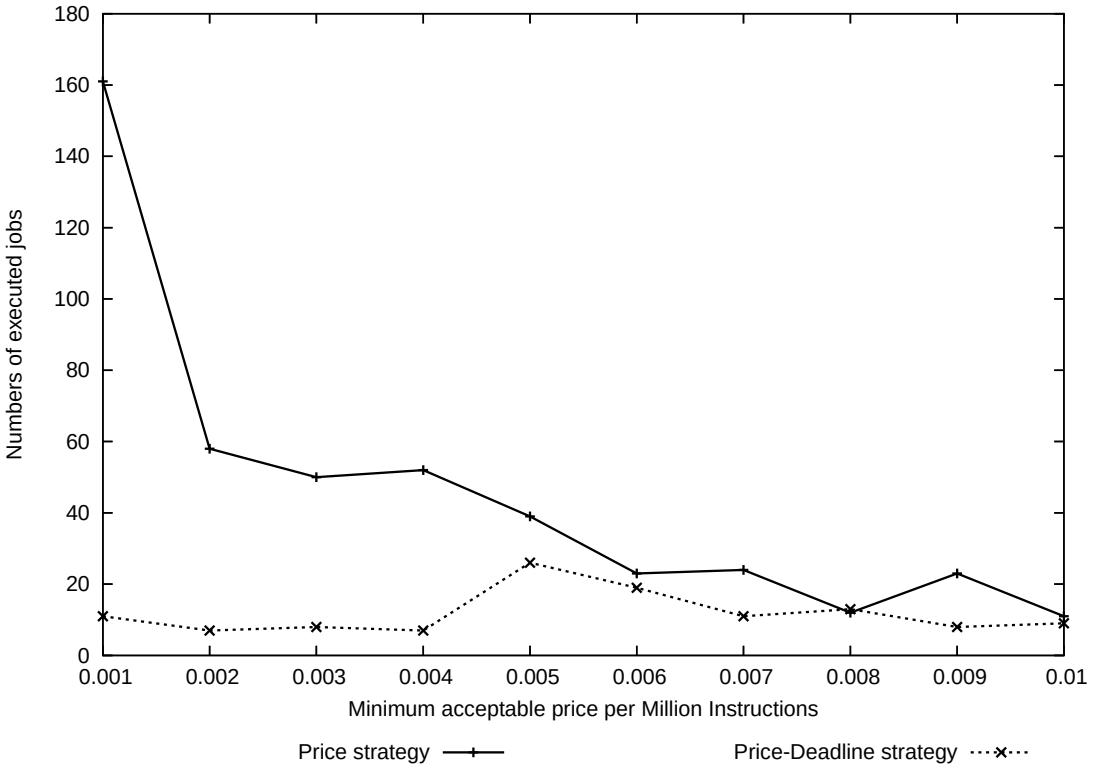

Figure 5.22: Number of jobs executed by two resource strategies for different minimum acceptable prices per Million Instructions with static submission.

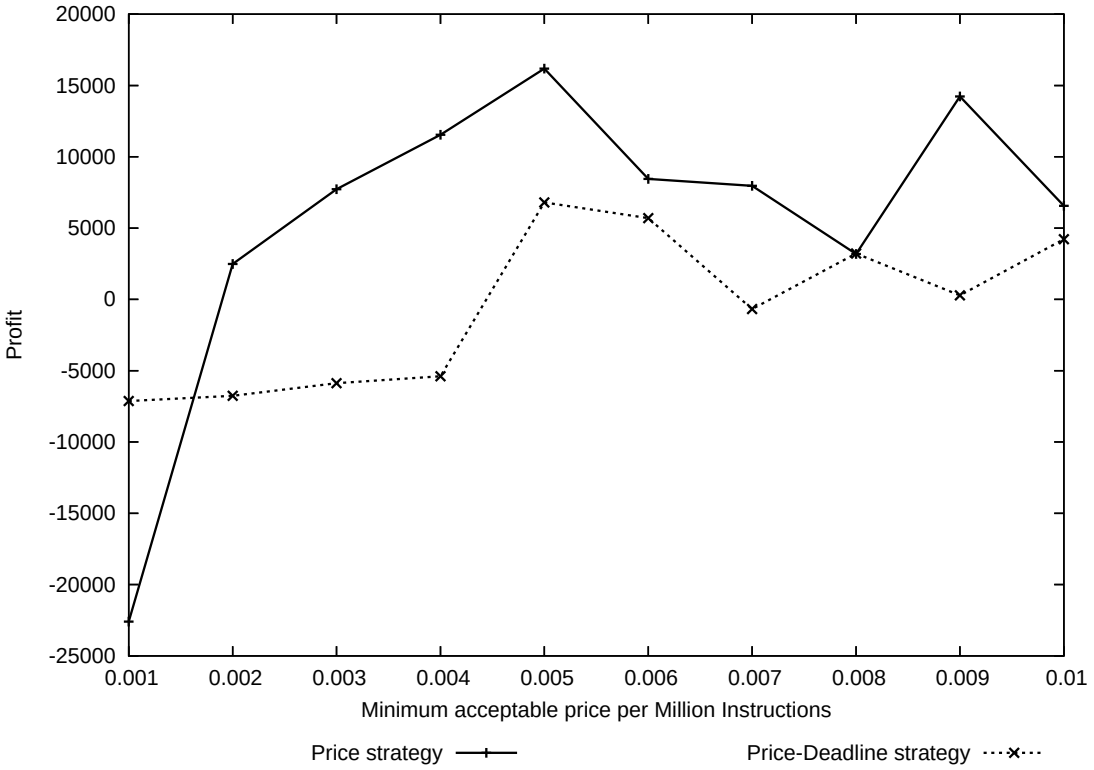

Figure 5.23: Profit generated by two resource strategies for different minimum acceptable prices per Million Instructions with static submission.

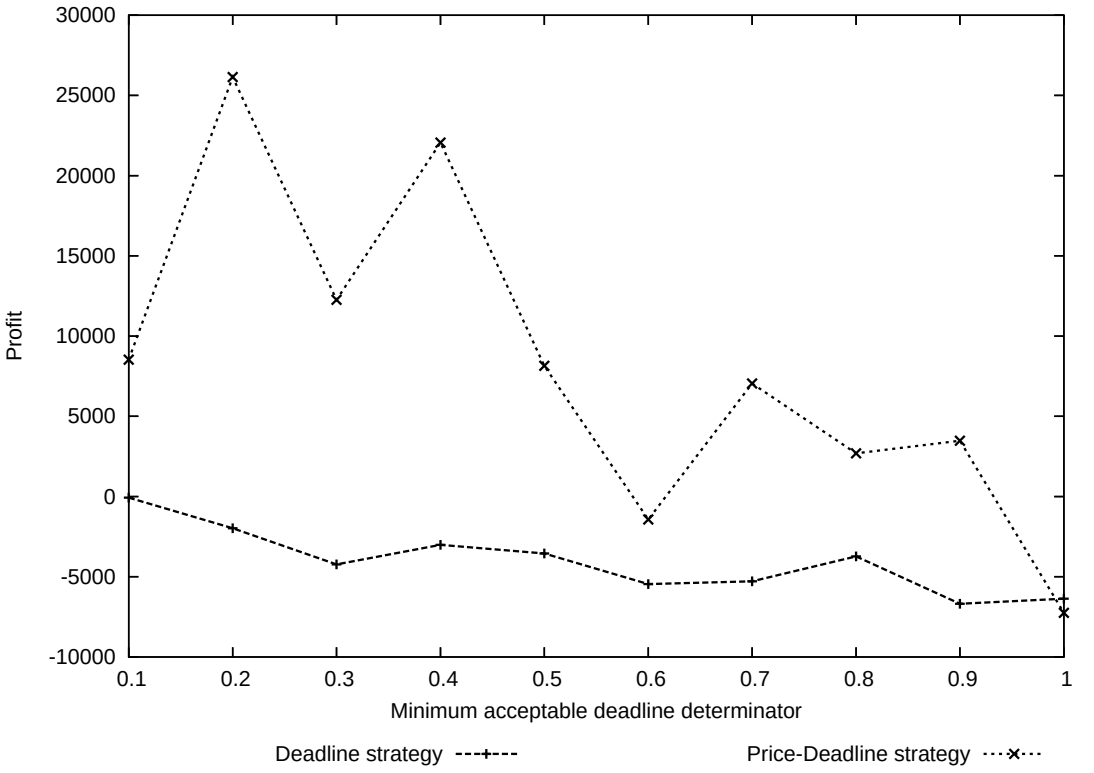

Figure 5.24: Profit generated by two resource strategies for different minimum acceptable deadline determinators with static submission.

5.6.3 Dynamic submission of jobs

In this experiment, the performance of different entity strategies is evaluated with dynamic submission (figures 5.25 to 5.39). Furthermore, the parameters are varied in the same way as described for the previous experiment. However, the following things can be noticed from the obtained results of performing simulations with dynamic submission:

- The difference between the performances of the user strategies in dynamic submission (figures 5.25 to 5.36) is usually smaller than the difference between the performances in static submission. The reason is the low job arrival rate in dynamic submission due to the long lengths of submitted jobs and which is the nature of jobs executed on the Grid. So, the time between the submission of two successive jobs of the same user is going to be long, since the execution of each job takes long time. As a result, the resources' processors will have good availability status in this experiment and therefore resources can satisfy the majority of users' deadlines.

- Both broker strategies have usually better performance in dynamic submission, because the overall number of jobs executed in dynamic submission is larger than the overall number of jobs executed in static submission. In other words, the failure rate is lower in dynamic submission. The low failure rate results from the good availability status of resources' processors as mentioned above.

- When MAPPMI is varied in dynamic submission (see figure 5.38), the performance of Price strategy tends to be better than the performance of the same strategy in static submission and which is the opposite to the Price-Deadline strategy. This is because the resource which employs Price strategy can execute a large number of jobs without a large effect on the availability of its processors due to the long times between the submissions of successive jobs as mentioned before. On the other hand, the degradation of Price-Deadline strategy results from the competition with resources which employ

Price and Deadline strategies and which in dynamic submission can execute more jobs especially the jobs belong to users who concern the time needed to complete the execution of their jobs. Price or Deadline strategy needs only one threshold to be met in order to reply, while Price-Deadline strategy needs two thresholds.

- When minimum acceptable deadline determinator is varied in dynamic submission (see figure 5.39), the Deadline strategy has usually better performance than the performance of the Price-Deadline strategy and which is the opposite of static submission. In static submission, the bad performance of Deadline strategy results from the early allocation of the processors to cheap jobs leaving it later unable to meet the deadlines of most of the jobs which arrive in short period of time. Nevertheless, the resources which employ this strategy in this experiment can have enough time for executing the cheap jobs before the arrival of the new jobs, because of the relaxed inter-arrival times of jobs in dynamic submission.

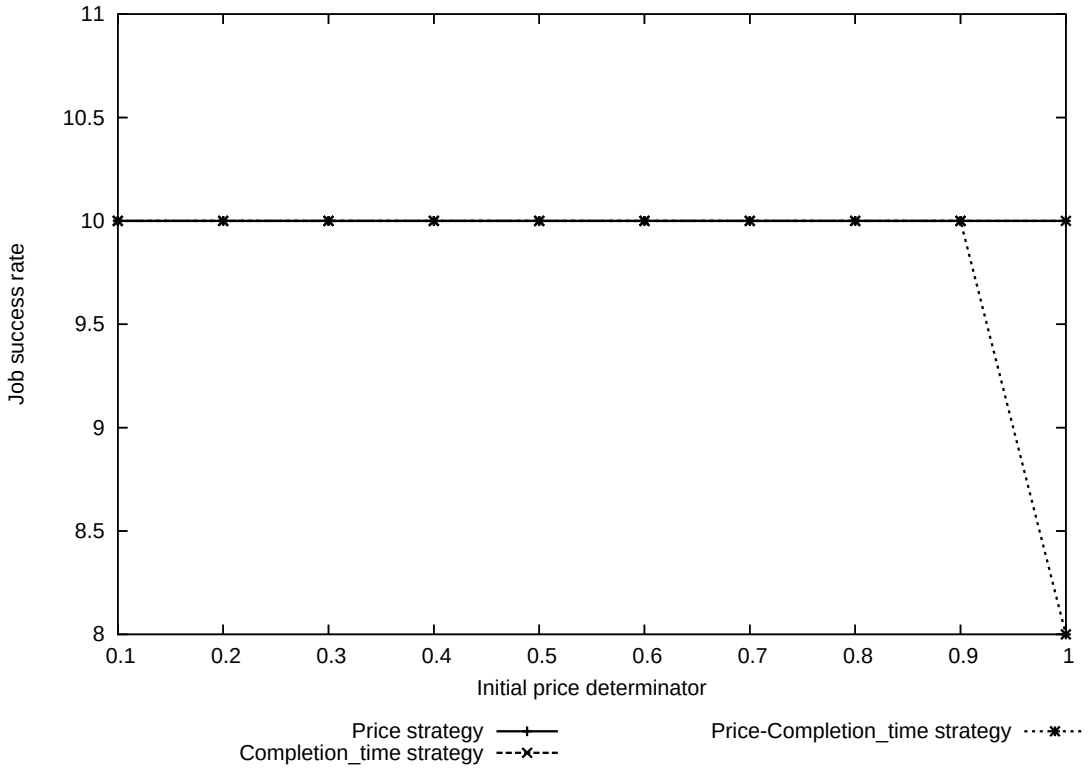

Figure 5.25: User strategy's job success rate for different initial price determinators with dynamic submission.

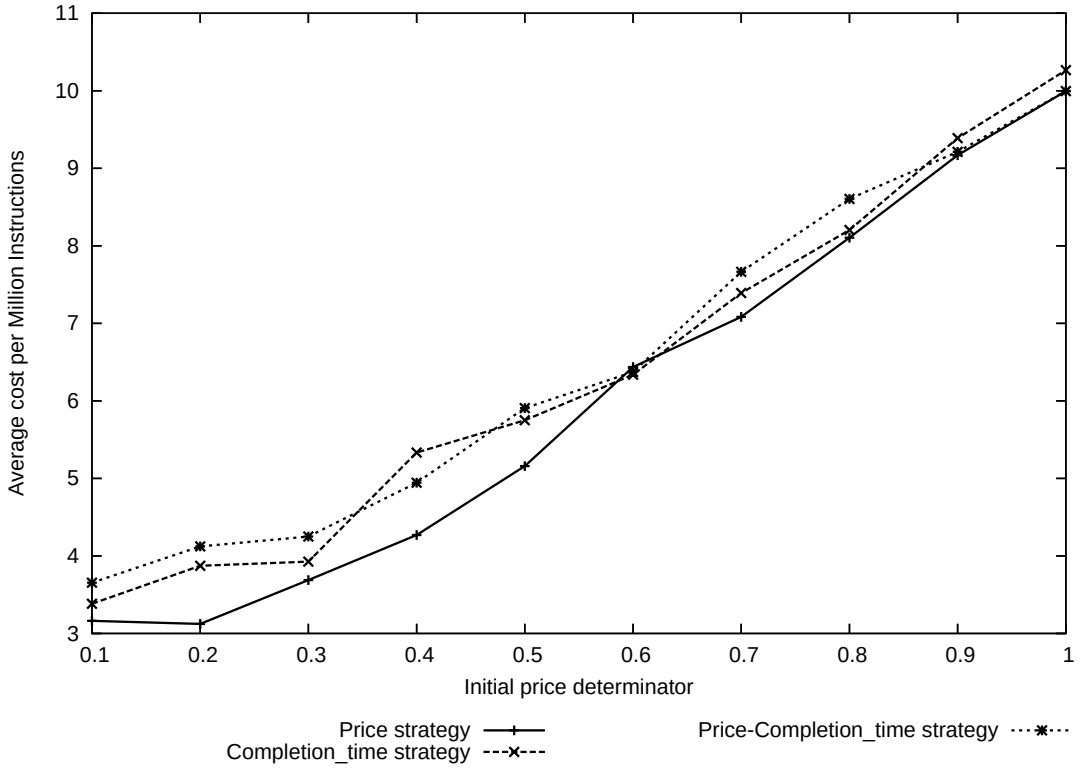

Figure 5.26: User strategy's average cost per Million Instructions for different initial price determinators with dynamic submission.

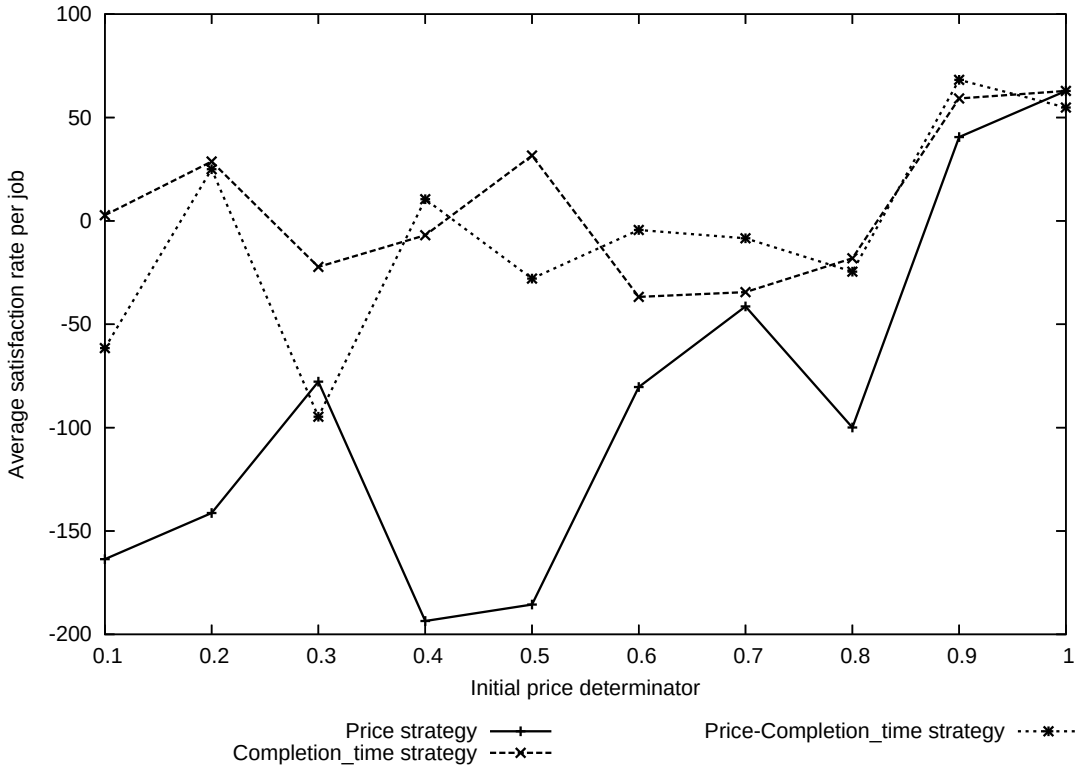

Figure 5.27: User strategy's average satisfaction rate per job for different initial price determinators with dynamic submission.

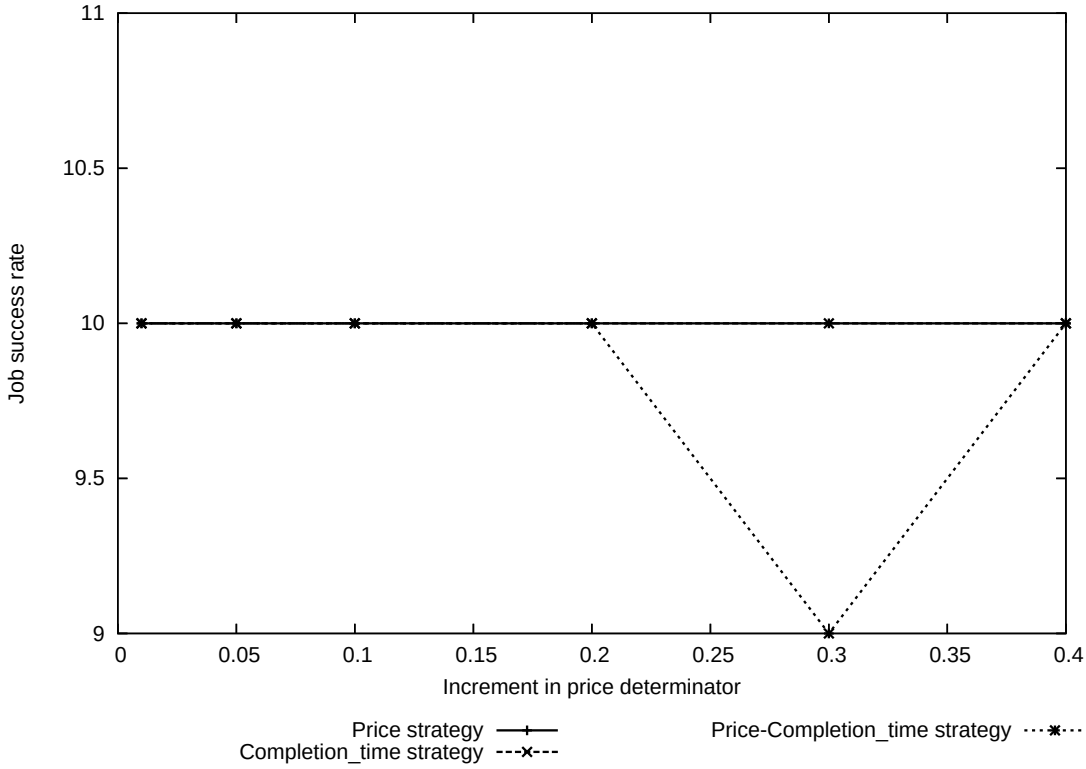

Figure 5.28: User strategy's job success rate for different increment in price determinators with dynamic submission.

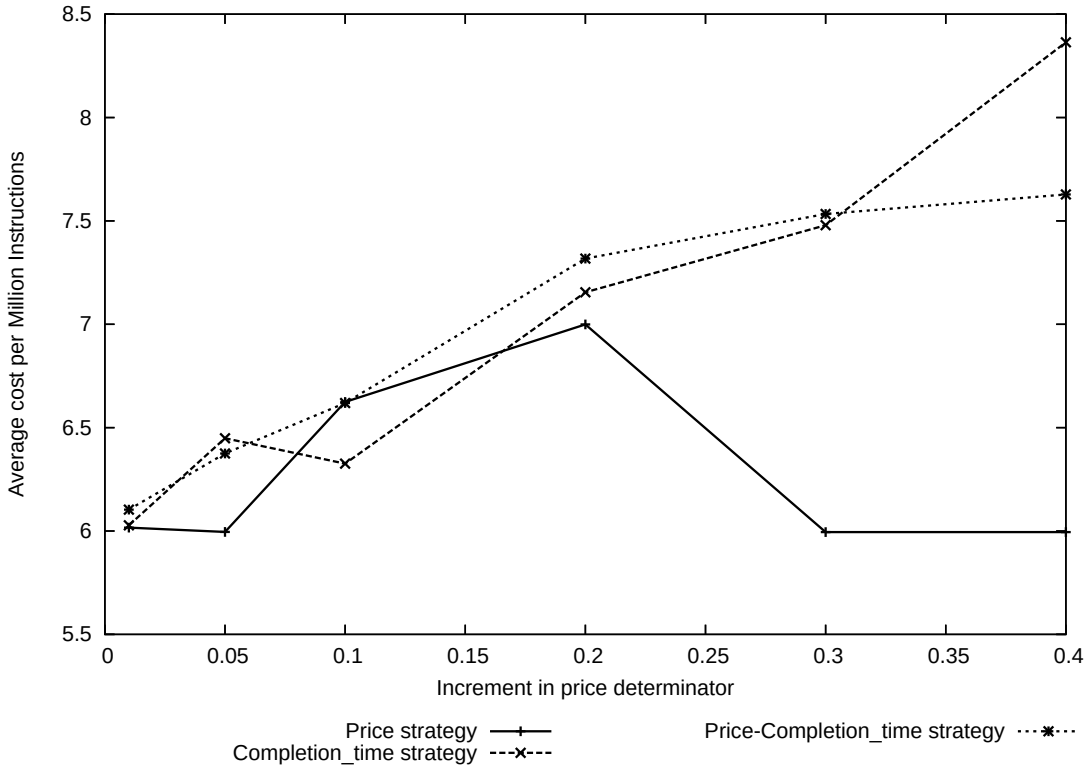

Figure 5.29: User strategy's average cost per Million Instructions for different increment in price determinators with dynamic submission.

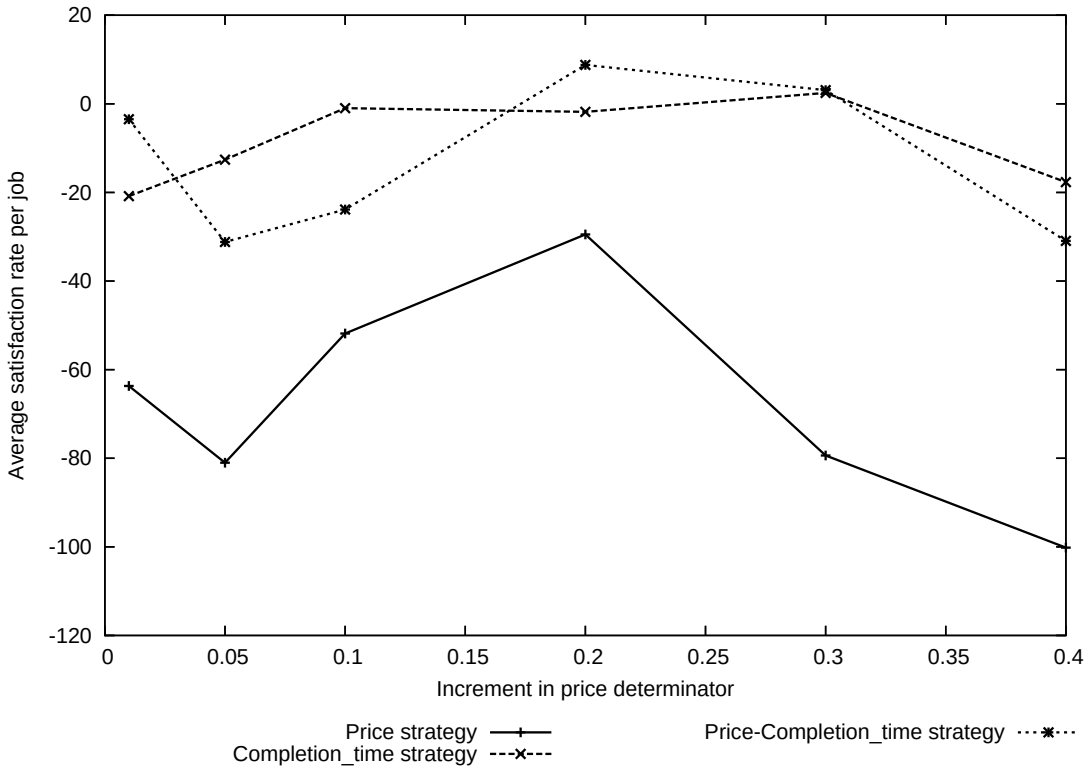

Figure 5.30: User strategy's average satisfaction rate per job for different increment in price determinators with dynamic submission.

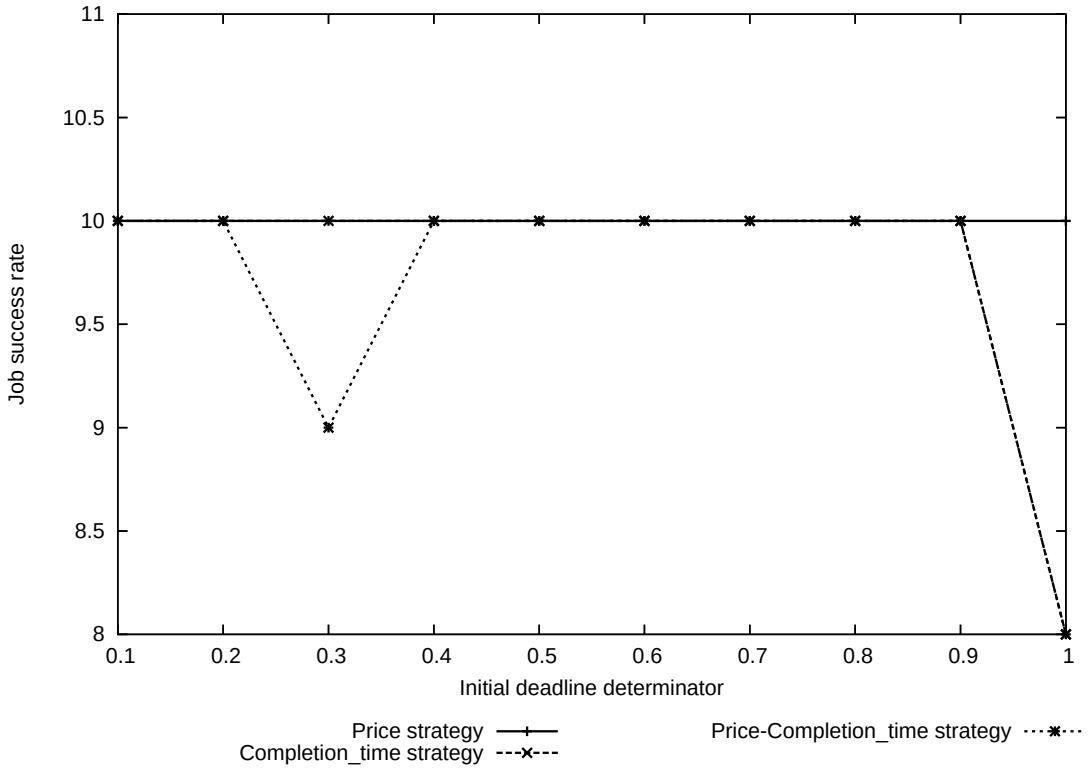

Figure 5.31: User strategy's job success rate for different initial deadline determinators with dynamic submission.

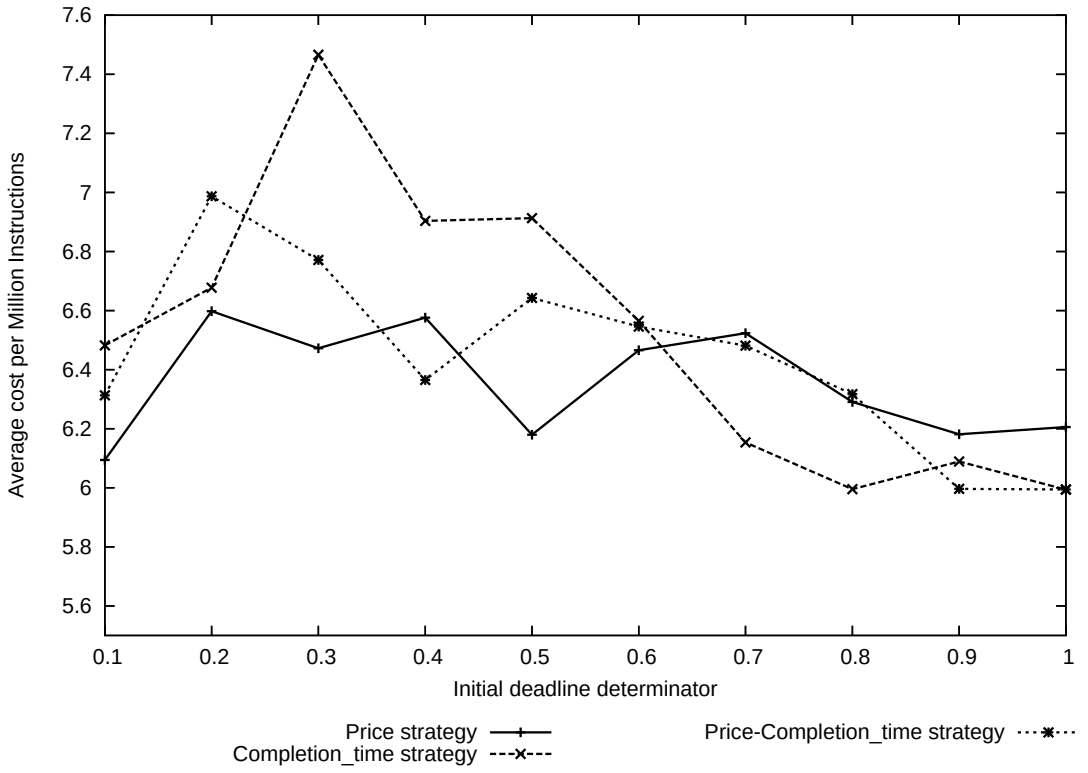

Figure 5.32: User strategy's average cost per Million Instructions for different initial deadline determinators with dynamic submission.

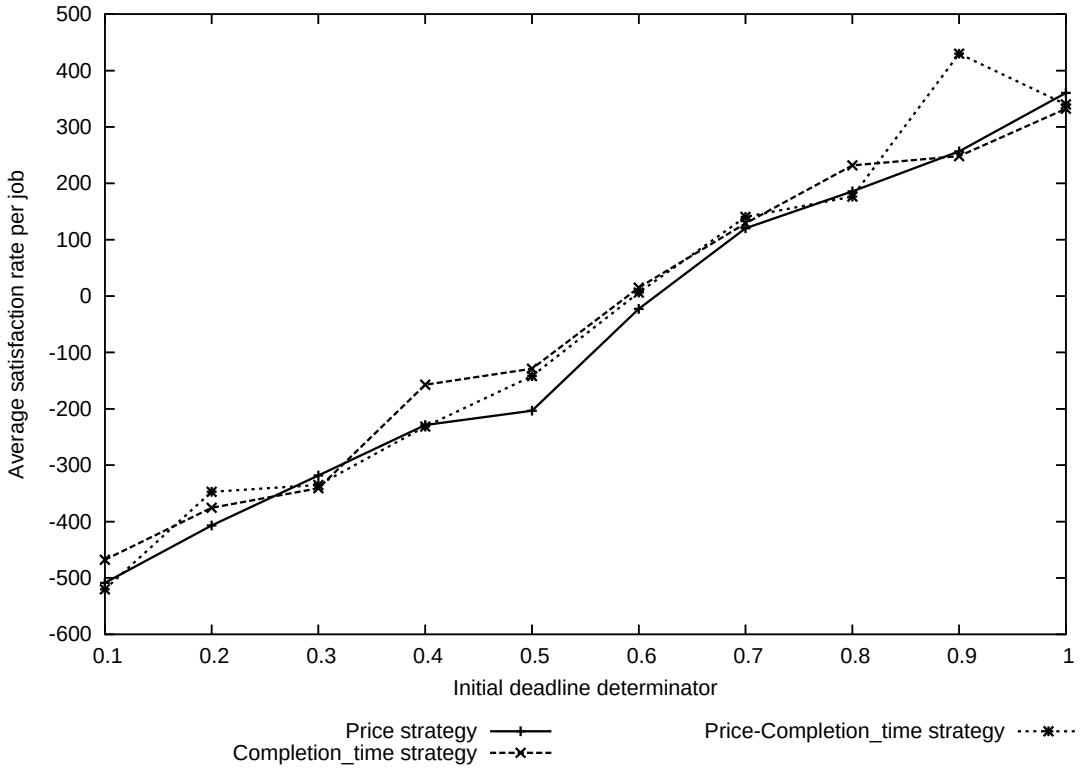

Figure 5.33: User strategy's average satisfaction rate per job for different initial deadline determinators with dynamic submission.

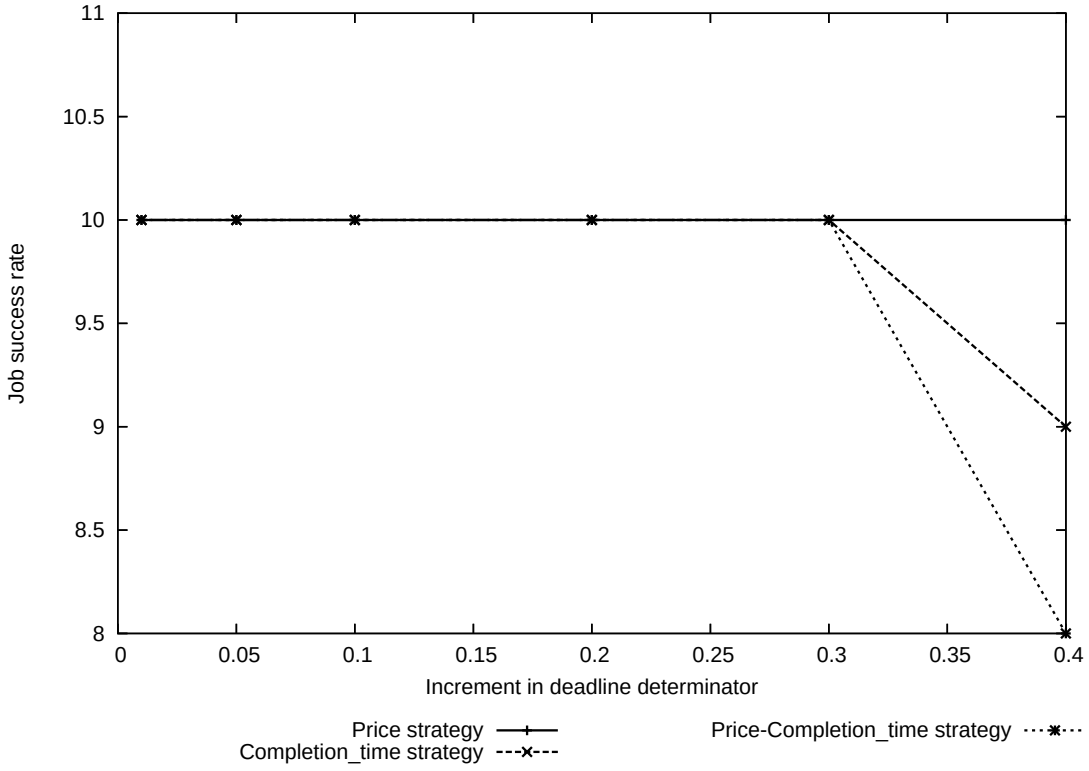

Figure 5.34: User strategy's job success rate for different increment in deadline determinators with dynamic submission.

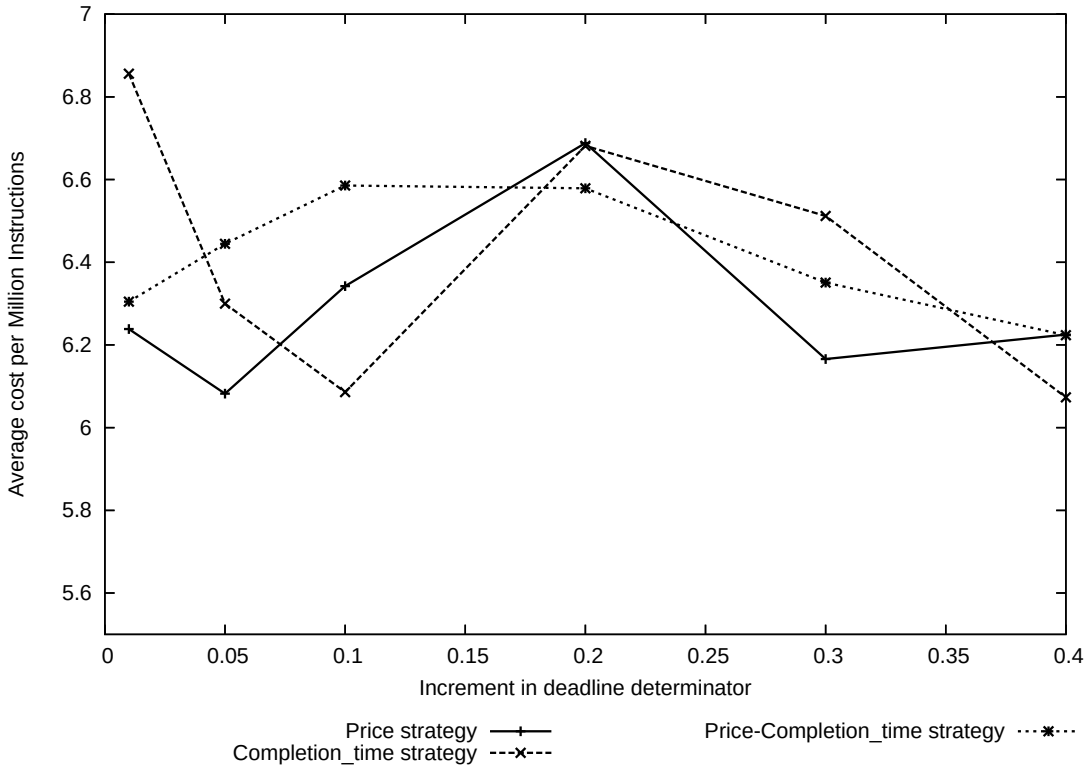

Figure 5.35: User strategy's average cost per Million Instructions for different increment in deadline determinators with dynamic submission.

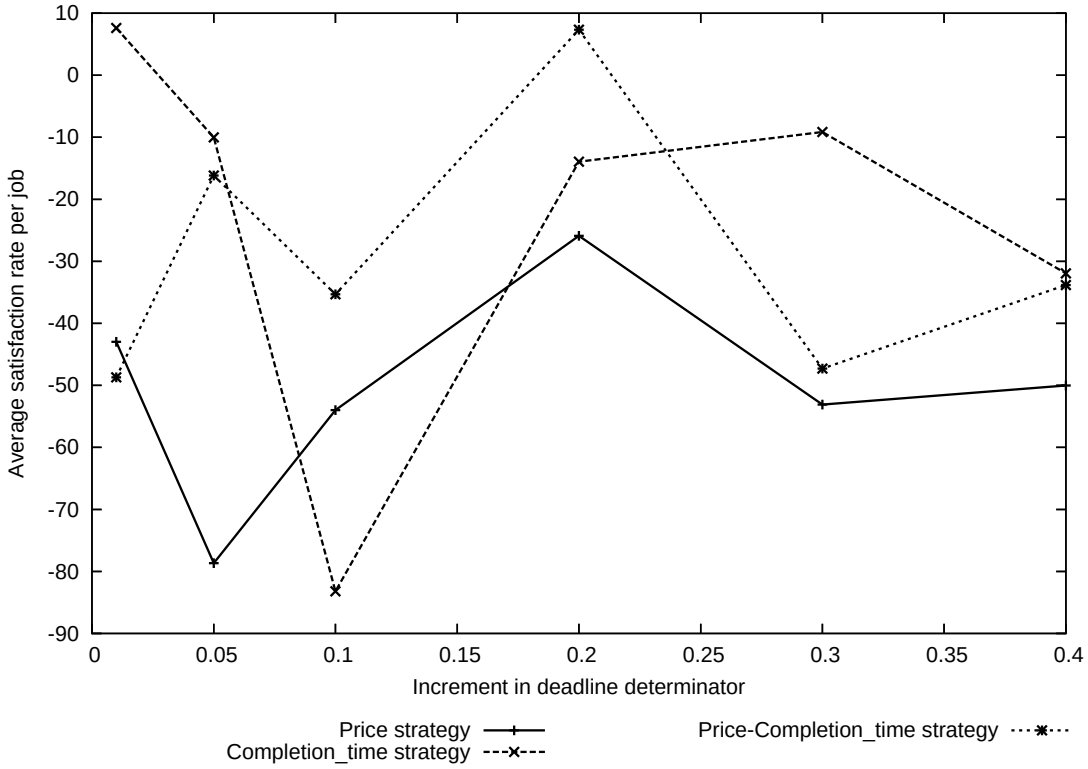

Figure 5.36: User strategy's average satisfaction rate per job for different increment in deadline determinators with dynamic submission.

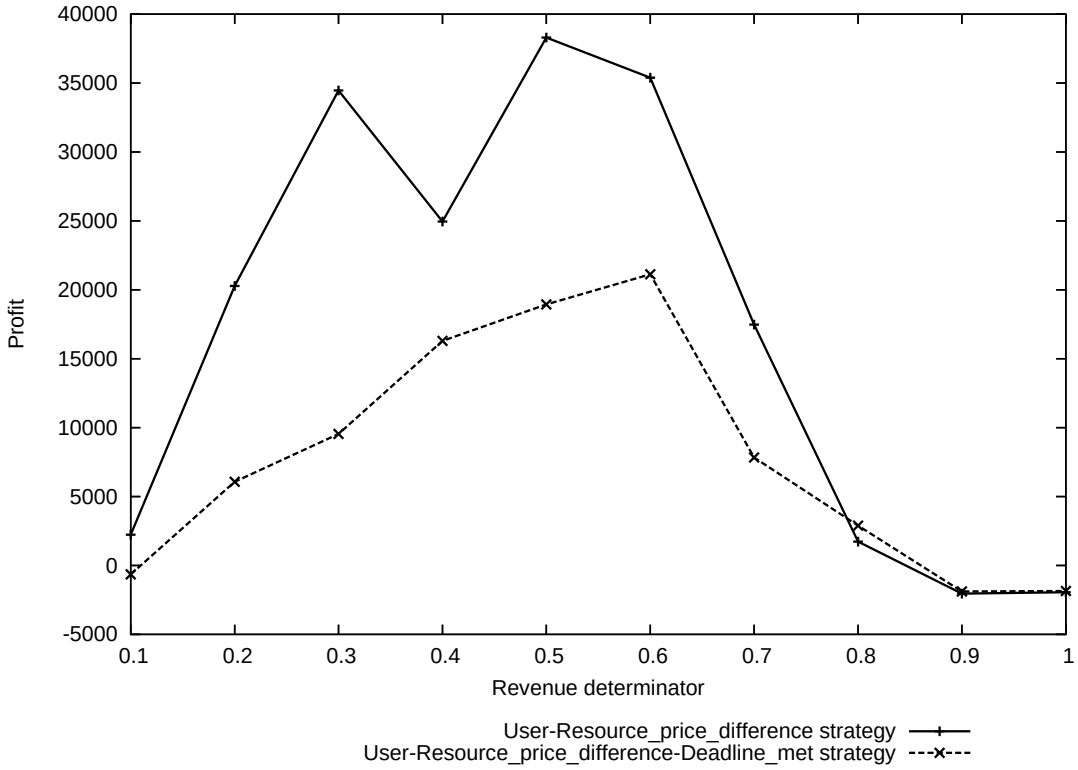

Figure 5.37: Profit generated by two broker strategies for different revenue determinators with dynamic submission.

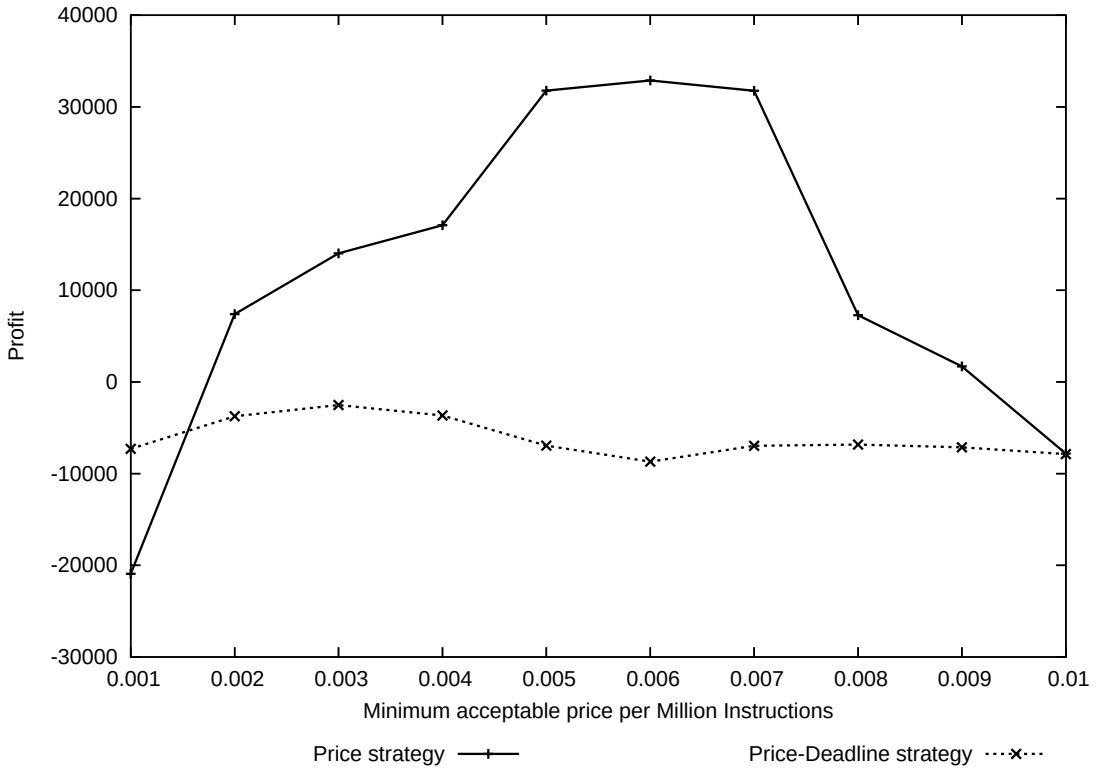

Figure 5.38: Profit generated by two resource strategies for different minimum acceptable prices per Million Instructions with dynamic submission.

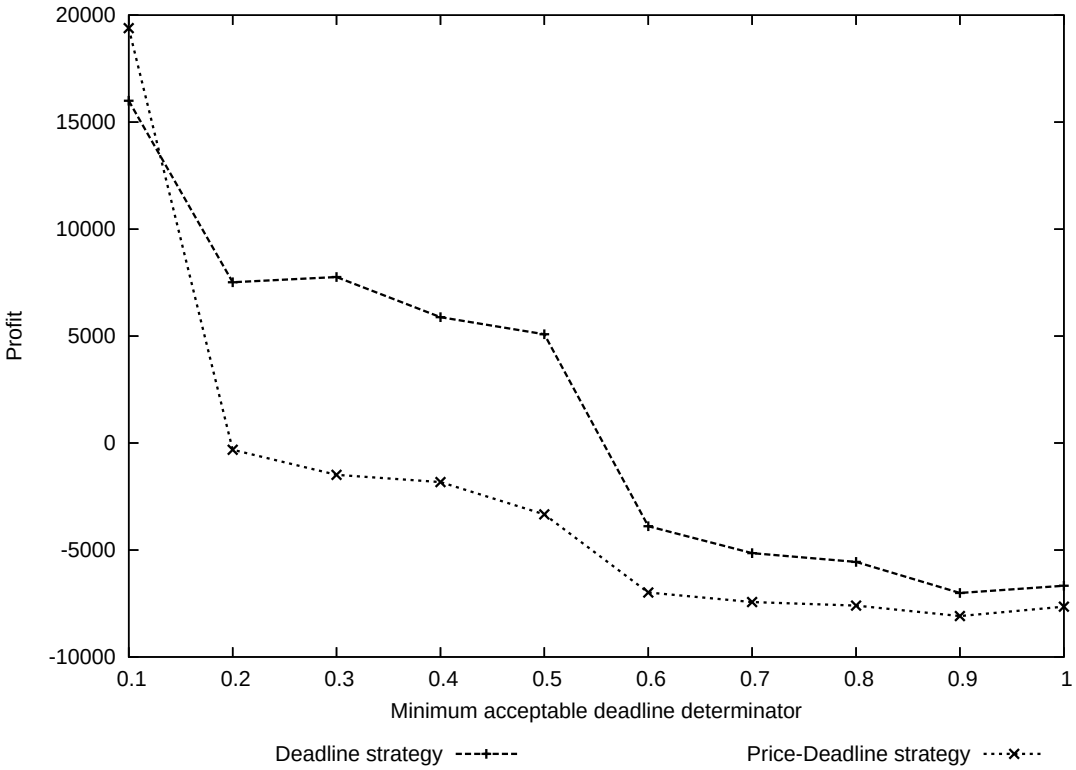

Figure 5.39: Profit generated by two resource strategies for different minimum acceptable deadline determinators with dynamic submission.

5.7 Chapter summary

In this chapter, we simulated and evaluated the performance of different entity strategies for a variety of scenarios. Two kinds of submission of jobs have been considered: *static* and *dynamic* submission. The evaluation has been done for:

- Price, Completion_time and Price-Completion_time strategies for users.
- User-Resource_price_difference and User-Resource_price_difference-Deadline_met strategies for brokers.
- Price, Deadline and Price-Deadline strategies for resources.

For users, the Price strategy has been superior in terms of job success rate and average cost per MI, but inferior in term of average satisfaction rate per job when the submission of jobs is static. For brokers, User-Resource_price_difference strategy has got a better performance than the other strategy. For resources, the price strategy has got the best performance except when the Minimum acceptable price per MI is too low (0.001). However, in dynamic submission:

- the difference between the performances of different user strategies shrinks.
- the brokers strategies have got better performance, because of low failure rate.
- the performances of resource Price and Deadline strategies improve, whilst the performance of resource Price-Deadline strategy deteriorates.

It can be observed from the static submission scenario results that the price strategy is the best strategy for the user to employ if it:

- concerns the percentage of its jobs that can be executed on Grid resources.
- wants to pay less for executing its jobs.

But, the user has to employ Completion_time or Price-Completion_time strategy, if it cares about how long its jobs will take to complete.

The brokers and resources have to employ User-Resource_price_difference and Price strategies, respectively in order to generate more profit. However, resource Price strategy has a poor performance when the resource minimum acceptable price is too low. On the other hand, employing any user strategy won't make much difference in performance, when the submission is dynamic.

It can also be observed that the characteristics of entities have an influence on the performance of strategies. For instance, user Completion_time and Price-Completion_time strategies have similar performance, because of the existence of resources with different prices from cheap to expensive and existence of resources which don't care about the price paid for the use of their processors. So, the price threshold won't have a large effect on the performance. However, these strategies will not have similar performance, if the Grid only contains expensive resources. For brokers, it is observed from the obtained results that broker strategies have the best performance when their revenue determinators are neither too low nor too high. Thus, it is concluded that the broker's aim has to be receiving reasonable revenue from acting on behalf of users.

Chapter 6

Conclusions and future work

6.1 Summary

Conventional (traditional) scheduling considers the overall system performance to evaluate the scheduling quality such as utilisation and schedule length. Furthermore, conventional scheduling doesn't consider pricing of resource usage. However, pricing is important because it gives resource providers an incentive to supply their resources to the Grid. Moreover, pricing is important because it enforces the users to utilise the resources just when they need them because they have to pay for their use. Thus, economic scheduling needs to be considered for Grid computing environment.

To support this, a framework for economic scheduling in Grid computing using tender model has been developed. Unlike other work that concerns only the performance of either users and resources and rarely both, all Grid entities including brokers have their strategies that aim to maximise their utilities.

In this thesis, we have also:

- distinguished between different classes of entities such as users, brokers and resources based on what they send and receive. Thus,
  - The entity is a user if it sends jobs and receives bids.

  - The entity is a resource if it sends bids and receives jobs.
  - The entity is a broker if it sends jobs and bids and receives jobs and bids.

- designed strategies for different classes of entities such as:
  - Price, Completion_time and Price-Completion_time for users.
  - Price, Deadline and Price-Deadline strategies for resources.
  - User-Resource_price_difference and User-Resource_price_difference-Deadline_met for brokers

- developed a Java-based simulator, called MICOSim, that supports event-driven simulation of economic scheduling in Grid computing using tendering to allow performance evaluation under different scenarios. MICOSim can perform a simulation of a large number of entities (more than one hundred entities) in a short time due to the employment of binary insert. Binary insert is an modified version of well known binary search.

- evaluated the performance of the designed strategies through a series of simulations by varying a number of parameters.

The evaluation shows that the price strategy is the best strategy for the user to employ if it concerns the percentage of its jobs that can be executed on Grid resources, when the system load is usually high as in the static submission scenario. Furthermore, the user has to employ it, if it wants to pay less for executing its jobs. But, the user has to employ Completion_time or Price-Completion_time strategy, if it cares about how long its jobs will take to complete. The brokers and resources have to employ User-Resource_price_difference and Price strategies, respectively in order to generate more profit. However, resource Price strategy has a poor performance when the resource minimum acceptable price is too low. On the other hand, employing any user strategy won't make much difference in performance, when the system load is low as in the dynamic submission scenario.

Moreover, the broker and resource strategies that are superior when the system load is high stay superior when the system load is low as well.

The dynamic submission differs from static submission in that the worst resource strategy is Price-Deadline strategy in dynamic submission, while it is Deadline strategy in static submission. Secondly, the job success rate of users who employ Completion_time and Price-Completion_time strategies improves, while the average satisfaction rate per job of users who employ Price strategy improves when the submission of jobs is dynamic. Finally, the overall profit generated by all brokers and resources is higher in dynamic submission. It is observed from above that the entity strategy's performance is influenced by the behaviour of other entities such as the submission time of user's jobs. For better performance, a strategy can keep records for the interactions occurred between the entity which employs it and other entities to have an expectation of their course of actions in the future. Thus, it can send offers based on its expectations. For example, a strategy knows from its records that resource x always accepts requests for execution from its user with a specified price. So, the strategy sends a lower price and sees if resource x is going to accept the request this time too. If not, the strategy increases the price and sends a new request. Otherwise, the strategy can keep decreasing the price until the resource stops accepting its requests. By doing so, the user saves some money and therefore is going to have better performance. It is also observed that the characteristics of entities have an effect on the performance of strategies too. For instance, user Completion_time and Price-Completion_time strategies have similar performance, because of the existence of resources with various prices from cheap to expensive and existence of resources which don't care about the price paid for the execution. So, the price threshold won't have a large effect on the performance as mentioned in the previous chapter. However, these strategies will not have similar performance, if the Grid only contains expensive resources. For brokers, it appears from the results that broker strategies have the best performance when their revenue determinators are neither too low nor too high. Thus, it is concluded that the broker's aim has to be receiving reasonable

revenue (neither too low nor too high) from acting on behalf of users.

6.2 Future work

There are four main areas of consideration: extending the framework to support data Grids, supporting more market models, supporting accounting and undertaking further experimental investigations.

The framework that has been proposed can be extended to support storage use in addition to CPU. In this case, the user can request to execute a job, use of storage or both. This needs adding or making modifications to the parameters that the request involves in addition to doing the same thing for the parameters in the reply. Furthermore, the framework can be generalised to support more market models including commodity and auction models. Accounting can also be supported by having methods that deals with accounting. This includes keeping a track of resource usage, defining charging mechanism for user usage and defining ways of paying for the usage (e.g. credit card and cheque).

Finally, the evaluation of the entity strategies can be done under more scenarios. For example, considering the scenario where users have different numbers of jobs with different lengths, and resources have different numbers of processors with different speeds. Additionally, considering the scenario where resources accept only specific job classes. In this way, it can be seen how much effect the entities' characteristics have on the performance of the employed strategies.

Bibliography

[1] The tale of two exchanges: NYSE and Nasdaq. Available at http://www.taxopedia.com/articles/basics/03/103103.asp.

[2] Trading on the AMEX. Available at http://www.amex.com/?href=/atamex/aboutAmex/mktStructure/at_mktStructure.html.

[3] Amazon elastic compute cloud (amazon ec2) - limited beta. Available at http://www.amazon.com/b?ie=UTF8&node=201590011, 2006.

[4] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. *Future Gener. Comput. Syst.*, 18(8):1061–1074, 2002.

[5] Agorics Inc. Going, going, gone! A survey of auction types, 1996.

[6] A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra. A unified resource scheduling framework for heterogeneous computing environments. In *Heterogeneous Computing Workshop*, pages 156–165, 1999.

[7] M. Assuncao and R. Buyya. An evaluation of communication demand of auction protocols in grid environments. Technical report, University of Melbourne, Melbourne, Australia, 2006.

[8] F. Azzedin and M. Maheswaran. Evolving and managing trust in grid computing systems. In *IEEE canadian conference on electrical and computer engineering*, 2002.

[9] F. Azzedin and M. Maheswaran. Towards trust-aware resource management in grid computing systems. In *First IEEE International Workshop on Security and Grid Computing*, 2002.

[10] A. Barmouta and R. Buyya. Gridbank: A grid accounting services architecture(gasa) for distributed systems sharing and integration. In *International parallel and distributed processing symposium*, 2003.

[11] A. Beguelin, J. Dongarra, A. Geist, and V. Sunderam. Visualization and debugging in a heterogeneous environment. *Computer*, 26(6):88–95, 1993.

[12] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid replication strategies in optorsim. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, pages 46–57, London, UK, 2002. Springer-Verlag.

[13] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Optorsim: A Grid simulator for studying dynamic data replication strategies. *IJHPCA*, 17(4):403–416, Winter 2003.

[14] F. Berman and R. Wolski. The apples project: A status report. In *Proceedings of the 8th NEC Research Symposium,*, 1997.

[15] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, pages 39–65. ACM Press, 1996.

[16] V. Berstis. Fundamentals of grid computing. Technical report, IBM corporation, 2002.

[17] M. Bsoul, I. Phillips, and C. Hinde. A framework for economic scheduling in grid computing using tender/contract-net model. In *pgnet*, 2006.

[18] R. Buyya. *Economic-based distributed resource management and scheduling for grid computing*. PhD thesis, Monash university, Melbourne, Australia, 2002.

[19] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *HPC ASIA'2000*, 2000.

[20] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service oriented grid computing. In *IPDPS '01: Proceedings of the 10th Heterogeneous Computing Workshop HCW 2001 (Workshop 1)*, pages 83–97. IEEE Computer Society, 2001.

[21] R. Buyya and M. Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[22] R. Buyya and S. Vazhkudai. Compute power market: Towards a market-oriented grid. In *CCGRID*, pages 574–581, 2001.

[23] J. Cao, S. Jarvis, D. Spooner, J. Turner, D. Kerbyson, and G. Nudd. Performance prediction technology for agent-based resource management in grid environments. In *International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops*, 2002.

[24] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. *ccgrid*, 00:430–437, 2001.

[25] H. Casanova and J. Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Su-*

percomputer Applications and High Performance Computing, 11(3):212–223, 1997.

[26] 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 60–78. IEEE Computer Society, 2000.

[27] H. Chen and M. Maheswaran. Distributed dynamic scheduling of composite tasks on grid computing systems. In *IPDPS*, 2002.

[28] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical report, University of California, 2000.

[29] L. Chunlin and L. Layuan. An agent-based approach for grid computing, 2003.

[30] J. Cohen, J. Darlington, and W. Lee. Payment and negotiation for the next generation grid and web. In *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, 2005.

[31] C. Courcoubetis, M. Dramitinos, and G. Stamoulis. An auction mechanism for bandwidth allocation over paths, 2001.

[32] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing, 2001.

[33] C. Dumitrescu and I. T. Foster. Gangsim: a simulator for grid scheduling studies. In *CCGRID*, pages 1151–1158, Cardiff, UK, 2005.

[34] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *International symposium on cluster computing and the grid*, 2002.

[35] C. Ernemann, V. Hamscher, and R. 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. Springer-Verlag, 2002.

[36] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2221, pages 188–206, London, UK, 2001. Springer-Verlag.

[37] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[38] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15, 2001.

[39] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. A. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In *Heterogeneous Computing Workshop*, pages 184–199, 1998.

[40] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore. Smartnet: A scheduling framework for metacomputing. In *Second International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN '96)*, 1996.

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

[42] N. Fujimoto and K. Hagihara. A comparison among grid scheduling algorithms for independent coarse-grained tasks. In *SAINT 2004 Workshops*, pages 674–680, 2004.

[43] A. Galstyan, K. Czajkowski, and K. Lerman. Resource allocation in the grid using reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2004.

[44] J. Gomoluch and M. Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Middleware Workshops*, pages 211–218, 2003.

[45] J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In *HPDC*, pages 43–50, 2000.

[46] L. He. Designing economic-based distributed resource and task allocation mechanisms for self-interested agents in computational grids. Available at `http://gracehopper.org/2004/Proceedings/PDF/phd_He.pdf#search=%22Linli%20He%20designing%22`.

[47] L. He and T. R. Ioerger. Task-oriented computational economic-based distributed resource allocation mechanisms for computational grids. In *IC-AI*, pages 462–468, 2004.

[48] X. He, X. Sun, and G. V. Laszewski. A qos guided scheduling algorithm for grid scheduling, 2003.

[49] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 214–227. Springer-Verlag, 2000.

[50] A. Holub. Programming java threads in the real world, part 1. Available at `http://www.javaworld.com/jw-09-1998/jw-09-threads.html`.

[51] M. Humphrey and M. R. Thompson. Security implications of typical grid computing usage scenarios. *Cluster Computing*, 5(3):257–264, 2002.

[52] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *International Workshop on Grid Computing*, 2001.

[53] C. Kenyon and G. Cheliotis. Forward price dynamics and option prices for network commodities. Computing in Economics and Finance 2002 372, Society for Computational Economics, July 2002.

[54] C. Kenyon and G. Cheliotis. Creating services with hard guarantees from cycle-harvesting systems. In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, pages 224–231. IEEE Computer Society, 2003.

[55] C. Kenyon and G. Cheliotis. Grid resource commercialization: economic engineering and delivery scenarios. *Kluwer Academic Publishers*, pages 465–478, 2004.

[56] D. D. Kouvatsos and I. Mkwawa. Multicast communication in grid computing networks with background traffic. *IEE Proceedings - Software*, 150(4):257–264, 2003.

[57] G. Laszewski. Grid computing: Enabling a vision for collaborative research. In *PARA '02: Proceedings of the 6th International Conference on Applied Parallel Computing Advanced Scientific Computing*, pages 37–52. Springer-Verlag, 2002.

[58] A.M. Law and M.G. McComas. Secrets of successful simulation studies. *Industrial Engineering*, 22, 1990.

[59] A. Legrand. Simgrid 3.0 is out. Available at `http://gforge.inria.fr/forum/forum.php?forum_id=234`.

[60] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. *ccgrid*, 00:138–145, 2003.

[61] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous

computing systems. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, pages 30–44. IEEE Computer Society, 1999.

[62] M. Maheswaran, T. Braun, and H. Siegel. Heterogeneous distributed computing. In *Encyclopedia of Electrical and Electronics Engineering*, 1999.

[63] M. Maheswaran and K. Krauter. A parameter-based approach to resource discovery in grid computing systems. In *Proceedings of the First International Workshop in Grid Computing*, pages 181–190, 2000.

[64] R. Min and M. Maheswaran. Scheduling advance reservations with priorities in grid computing systems. In *In Proceedings of PDCS01*, 2001.

[65] R. Min and M. Maheswaran. Scheduling co-reservations with priorities in grid computing systems. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 266–267. IEEE Computer Society, 2002.

[66] R. A. Moreno and A. B. Alonso-Conde. Job scheduling and resource management techniques in economic grid environments. In *European Across Grids Conference*, pages 25–32, 2003.

[67] J. Nakai. Pricing computing resources: Reading between the lines and beyond. Technical report, National Aeronautics and Space Administration, 2002.

[68] S. Newhouse and J. Darlington. Computational communities: A marketplace for federated resources. In *HPCN Europe*, pages 667–674, 2001.

[69] S. Newhouse, J. MacLaren, and K. Keahey. *Trading Grid services within the UK e-science Grid*. Kluwer Academic Publishers, 2004.

[70] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the popcorn project. In *ICDCS*, pages 592–601, 1998.

[71] D. C. Parkes and L. H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 74–81. AAAI Press / The MIT Press, 2000.

[72] ProModel Corporation. What is simulation? Available at `http://www.promodel.com/simulation.asp`.

[73] S. R. Reddy. Market economy based resource allocation in grids. Master's thesis, Indian Institute of Technology, Kharagpur, India, 2006.

[74] N. Sample, P. Keyani, and G. Wiederhold. Scheduling under uncertainty: Planning for the ubiquitous grid. In *Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 300–316. Springer-Verlag, 2002.

[75] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artif. Intell.*, 135(1-2):1–54, 2002.

[76] T. Sandholm and S. Suri. Optimal clearing of supply/demand curves. In *13th annual International symposium on algorithms and computation*, 2002.

[77] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 69–76. ACM Press, 2002.

[78] J. M. Schopf, J. Nabrzyski, and J. Weglarz. *Grid resource management: State of the art and future trends*. Kluwer Academic Publishers, 2004.

[79] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Softw. Pract. Exper.*, 34(6):573–590, 2004.

[80] S. Song, K. Hwang, and M. Macwan. Fuzzy trust integration for security enforcement in grid computing. In *NPC*, pages 9–21, 2004.

[81] E. Srisan and P. Uthayopas. Heuristic scheduling with partial knowledge under grid environment. In *Second International Symposium on Communications and Information Technology*, 2002.

[82] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Distributed job scheduling on computational grids using multiple simultaneous requests. In *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02)*, pages 354–361. IEEE Computer Society, 2002.

[83] N. Subramaniam. Grid computing: An overview, 2003.

[84] K. Subramoniam, M. Maheswaran, and M. Toulouse. Towards a microeconomic model for resource allocation in grid computing systems. In *IEEE canadian conference on electrical and computer engineering*, 2002.

[85] Sun Microsystems. Interface comparator. Available at `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Comparator.html`.

[86] V. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[87] I. E. Sutherland. A futures market in computer time. *Commun. ACM*, 11(6):449–451, 1968.

[88] A. Takefusa, S. Matsuoka, K. Aida, H. Nakada, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 97–104, Washington, DC, USA, 1999. IEEE Computer Society.

[89] A. Takefusa, S. Matsuoka, H. Casanova, and F. Berman. A study of deadline scheduling for client-server systems on the computational grid. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Perfor-*

mance Distributed Computing (HPDC-10'01), pages 406–415. IEEE Computer Society, 2001.

[90] L. Tesfatsion. Agent-based computational economics: Growing economies from the bottom up. Staff General Research Papers 5075, Iowa State University, Department of Economics, March 2002.

[91] P. Tucker. Market mechanisms in a programmed system, 2002.

[92] A. Turgeon, Q. Snell, and M. Clement. Application placement using performance surfaces. In *9th IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, 2000.

[93] S. Vazhkudai and G. Laszewski. A greedy grid: The grid economic engine directive. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, pages 173–182, Washington, DC, USA, 2001. IEEE Computer Society.

[94] R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. Hey, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[95] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications*, 15(3):258–281, 2001.

[96] Y. Wu, G. Yang, J. Mao, S. Shi, and W. Zheng. Grid computing pool and its framework. In *International conference on parallel processing workshops*, 2003.

[97] L. Xiao, Y. Zhu, L. M. Ni, and Z. Xu. Gridis: An incentive-based grid scheduling. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2005. IEEE Computer Society.

[98] K. Yang, X. Guo, A. Galis, B. Yang, and D. Liu. Network engineering towards efficient resource on-demand in grid computing, 2003.

[99] D. Yu and T. Robertazzi. Divisible load scheduling for grid computing. In *Proc. of the IASTED International Conference on Paralle and Distributed Computing and Systems*, 2003.

[100] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov, and A. Shokurov. Comparison of scheduling heuristics for grid resource broker. In *ENC '04: Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)*, pages 388–392, Washington, DC, USA, 2004. IEEE Computer Society.

Appendix A

Output of first verification scenario

```
Job_0 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 27.0 (cpmi= 0.0080)

Broker_0 profit        66.0

Resource_0 profit      264.0

Job_1 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 47.0 (cpmi= 0.0080)

Broker_0 profit        132.0

Resource_0 profit      528.0

Job_0 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 127.0 (cpmi= 0.0080)

Broker_0 profit        198.0

Resource_0 profit      792.0

Job_1 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 147.0 (cpmi= 0.0080)

Broker_0 profit        264.0

Resource_0 profit      1056.0

Job_0 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 227.0 (cpmi= 0.0080)

Broker_0 profit        330.0
```

```
Resource_0 profit      1320.0

Job_1 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 247.0 (cpmi= 0.0080)

Broker_0 profit        396.0

Resource_0 profit      1584.0


Broker_0 profit        263.75

Resource_0 profit      1055.0

number of jobs executed by Resource_0 = 6

number of jobs submitted by Broker_0 = 6

number of jobs of User_0 executed on Resource_0 = 2
number of jobs of User_1 executed on Resource_0 = 2
number of jobs of User_2 executed on Resource_0 = 2

number of jobs of User_0 submitted by Broker_0 = 2
number of jobs of User_1 submitted by Broker_0 = 2
number of jobs of User_2 submitted by Broker_0 = 2

Amount paid by User_0 = 960.0
Amount paid by User_1 = 960.0
Amount paid by User_2 = 960.0

Total MIPS of User_0 = 120000.0
Total MIPS of User_1 = 120000.0
Total MIPS of User_2 = 120000.0

Total MIPS of successful jobs belong to User_0 = 120000.0
Total MIPS of successful jobs belong to User_1 = 120000.0
Total MIPS of successful jobs belong to User_2 = 120000.0

Total number of successful jobs belong to User_0 = 2.0
Total number of successful jobs belong to User_1 = 2.0
Total number of successful jobs belong to User_2 = 2.0

Metric 1 of User_0 = 10.0
Metric 1 of User_1 = 10.0
Metric 1 of User_2 = 10.0

Metric 2 of User_0 = 8.0
Metric 2 of User_1 = 8.0
Metric 2 of User_2 = 8.0

Metric 3 of User_0 = 15.0
Metric 3 of User_1 = -25.5
Metric 3 of User_2 = -36.0

failed= 0
```

Appendix B

Output of second verification scenario

```
Job_0 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 27.0 (cpmi= 0.0080)

Broker_0 profit        66.0

Resource_0 profit      264.0

Job_1 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 47.0 (cpmi= 0.0080)

Broker_0 profit        132.0

Resource_0 profit      528.0

Job_0 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 127.0 (cpmi= 0.0080)

Broker_0 profit        198.0

Resource_0 profit      792.0

Job_1 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 157.0 (cpmi= 0.0090)

Broker_0 profit        276.0

Resource_0 profit      1104.0

Job_0 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 480.0
and the time was 227.0 (cpmi= 0.0080)

Broker_0 profit        342.0
```

```
Resource_0 profit      1368.0

Job_1 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 257.0 (cpmi= 0.0090)

Broker_0 profit        420.0

Resource_0 profit      1680.0


Broker_0 profit        287.75

Resource_0 profit      1151.0

number of jobs executed by Resource_0 = 6

number of jobs submitted by Broker_0 = 6

number of jobs of User_0 executed on Resource_0 = 2
number of jobs of User_1 executed on Resource_0 = 2
number of jobs of User_2 executed on Resource_0 = 2

number of jobs of User_0 submitted by Broker_0 = 2
number of jobs of User_1 submitted by Broker_0 = 2
number of jobs of User_2 submitted by Broker_0 = 2

Amount paid by User_0 = 960.0
Amount paid by User_1 = 1020.0
Amount paid by User_2 = 1020.0

Total MIPS of User_0 = 120000.0
Total MIPS of User_1 = 120000.0
Total MIPS of User_2 = 120000.0

Total MIPS of successful jobs belong to User_0 = 120000.0
Total MIPS of successful jobs belong to User_1 = 120000.0
Total MIPS of successful jobs belong to User_2 = 120000.0

Total number of successful jobs belong to User_0 = 2.0
Total number of successful jobs belong to User_1 = 2.0
Total number of successful jobs belong to User_2 = 2.0

Metric 1 of User_0 = 10.0
Metric 1 of User_1 = 10.0
Metric 1 of User_2 = 10.0

Metric 2 of User_0 = 8.0
Metric 2 of User_1 = 8.5
Metric 2 of User_2 = 8.5

Metric 3 of User_0 = 15.0
Metric 3 of User_1 = -25.5
Metric 3 of User_2 = -36.0

failed= 0
```

Appendix C

Output of third verification scenario

```
Job_0 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 37.0 (cpmi= 0.0090)

Broker_0 profit        78.0

Resource_0 profit      312.0

Job_1 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 57.0 (cpmi= 0.0090)

Broker_0 profit        156.0

Resource_0 profit      624.0

Job_0 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 137.0 (cpmi= 0.0090)

Broker_0 profit        234.0

Resource_0 profit      936.0

Job_0 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 237.0 (cpmi= 0.0090)

Broker_0 profit        312.0

Resource_0 profit      1248.0

Job_1 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 257.0 (cpmi= 0.0090)

Broker_0 profit        390.0
```

```
Resource_0 profit      1560.0


Broker_0 profit        275.25

Resource_0 profit      1101.0

number of jobs executed by Resource_0 = 5

number of jobs submitted by Broker_0 = 5

number of jobs of User_0 executed on Resource_0 = 2
number of jobs of User_1 executed on Resource_0 = 1
number of jobs of User_2 executed on Resource_0 = 2

number of jobs of User_0 submitted by Broker_0 = 2
number of jobs of User_1 submitted by Broker_0 = 1
number of jobs of User_2 submitted by Broker_0 = 2

Amount paid by User_0 = 1080.0
Amount paid by User_1 = 540.0
Amount paid by User_2 = 1080.0

Total MIPS of User_0 = 120000.0
Total MIPS of User_1 = 120000.0
Total MIPS of User_2 = 120000.0

Total MIPS of successful jobs belong to User_0 = 120000.0
Total MIPS of successful jobs belong to User_1 = 60000.0
Total MIPS of successful jobs belong to User_2 = 120000.0

Total number of successful jobs belong to User_0 = 2.0
Total number of successful jobs belong to User_1 = 1.0
Total number of successful jobs belong to User_2 = 2.0

Metric 1 of User_0 = 10.0
Metric 1 of User_1 = 5.0
Metric 1 of User_2 = 10.0

Metric 2 of User_0 = 9.0
Metric 2 of User_1 = 9.0
Metric 2 of User_2 = 9.0

Metric 3 of User_0 = 5.0
Metric 3 of User_1 = 5.0
Metric 3 of User_2 = 4.0

failed= 1
```

Appendix D

Output of fourth verification scenario

```
Job_0 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 37.0 (cpmi= 0.0090)

Broker_0 profit        78.0

Resource_0 profit      312.0

Job_1 of User_0 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 57.0 (cpmi= 0.0090)

Broker_0 profit        156.0

Resource_0 profit      624.0

Job_0 of User_1 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 137.0 (cpmi= 0.0090)

Broker_0 profit        234.0

Resource_0 profit      936.0

Job_0 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 237.0 (cpmi= 0.0090)

Broker_0 profit        312.0

Resource_0 profit      1248.0

Job_1 of User_2 was submitted by Broker_0
and was executed on Resource_0
The price paid by user was 540.0
and the time was 257.0 (cpmi= 0.0090)

Broker_0 profit        390.0
```

```
Resource_0 profit      1560.0


Broker_0 profit        275.25

Resource_0 profit      1101.0

number of jobs executed by Resource_0 = 5

number of jobs submitted by Broker_0 = 5

number of jobs of User_0 executed on Resource_0 = 2
number of jobs of User_1 executed on Resource_0 = 1
number of jobs of User_2 executed on Resource_0 = 2

number of jobs of User_0 submitted by Broker_0 = 2
number of jobs of User_1 submitted by Broker_0 = 1
number of jobs of User_2 submitted by Broker_0 = 2

Amount paid by User_0 = 1080.0
Amount paid by User_1 = 540.0
Amount paid by User_2 = 1080.0

Total MIPS of User_0 = 120000.0
Total MIPS of User_1 = 120000.0
Total MIPS of User_2 = 120000.0

Total MIPS of successful jobs belong to User_0 = 120000.0
Total MIPS of successful jobs belong to User_1 = 60000.0
Total MIPS of successful jobs belong to User_2 = 120000.0

Total number of successful jobs belong to User_0 = 2.0
Total number of successful jobs belong to User_1 = 1.0
Total number of successful jobs belong to User_2 = 2.0

Metric 1 of User_0 = 10.0
Metric 1 of User_1 = 5.0
Metric 1 of User_2 = 10.0

Metric 2 of User_0 = 9.0
Metric 2 of User_1 = 9.0
Metric 2 of User_2 = 9.0

Metric 3 of User_0 = 5.0
Metric 3 of User_1 = 5.0
Metric 3 of User_2 = 4.0

failed= 1
```