ANALYZING BIDDING STRATEGIES FOR SCHEDULERS IN A SIMULATED MULTIPLE-CLUSTER MARKET DRIVEN ENVIRONMENT

BY

SINDHURA BANDHAKAVI

B.E., Osmania University, 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

Demand for high-end compute power has resulted in the emergence of the *Computational Grid* with several research systems facilitating its use. One such system *Faucets*, envisions the realization of *Compute Power as a Utility*, where the consumers can plug into the grid and tap required resources without being concerned about the source. For the efficient mapping of compute power producers and consumers in the system, *Faucets* employs the contract-net economic model. In this model, for each job consumers advertise their quality of service parameters and clusters reply with a bid. Bid value represents the amount a user is entrusted to pay in return for the successful completion of his job. Clusters participating in such a competing market need mechanisms for making profitable admission control decisions.

The thesis presents a conservative deadline driven scheduling algorithm with lookahead for clusters called the *GanttChart Strategy*. This strategy guarantees completion of the accepted jobs within their respective deadlines in addition to ensuring efficient system utilization. In order to generate a bid value in a computational auction, clusters can use the *GanttChart Strategy* to aid in the *private valuation* of jobs. Further, we analyze the effect of dynamically changing the bid value depending on the cluster's utilization and present a model for attaining market equilibrium prices in the contract-net model of *Faucets*.

Table of Contents

List of Figures

Chapter 1

Introduction

A large number of technical and scientific applications in the fields of high energy physics, earth observation, cosmology, molecular dynamics [13, 18] and emerging applications in Engineering Design etc. demand huge amounts of processing cycles and data accesses. Computational Grid has evolved to provide an enormous amount of compute power to solve such problems. Grid Computing refers to the efficient use of distributed computational resources belonging to single or multiple organizations. These independent organizations collaborate with each other forming Virtual Enterprises [7] and share the resources distributed across the world.

Emergence of the Grid has opened up new horizons for research in Computer Science. Several systems are being developed [11, 21] to realize *Compute Power* as a *utility*, just like water or electricity, where the consumer can tap required resources by paying for the used power, without being concerned about the source. In this scenario, producers are defined as the owners of computational resources, storage resources and the like, while consumers are defined as users with computation and/or data intensive jobs and a budget to expend.

These producers and consumers (together referred to as agents) come from different organizations with varied interests and objectives. For example, a cluster's objective function might be to maximize its utilization, while a user's might be to minimize the jobs' execution cost[1]. With every agent trying to optimize its objective function, an efficient resource

[1]User pays a price to the cluster for the computational resources used.

allocation model is needed that enables the jobs to get the best resources available and vice versa.

This scenario can be paralleled with the economic market where buyers and sellers are competing for the best deals for themselves. In order to apply the well-known theories on market economies to the Computational Grid, several issues need to be addressed. What kind of model is best suited for this dynamic environment? How does the quick processing power of computers affect the market decisions? What parameters should the QoS requirements include? How does a cluster decide which job to accept? How does it evaluate the importance of winning the job and consequently calculating the bid value? How can the requirements and the bids be standardised so that two different values can be reasonably compared? What criteria does the consumer use to select the best bid? A variety of researchers have approached these problems from different angles. In this thesis we explore the issue of applying market economy principles to job-resource matching. We also analyze the strategies by which resources can maximize their profit metric and users can minimize their expenses with the best Quality of Service (QoS) contracts.

The principal contributions of this thesis are:

- GanttChartStrategy for Clusters to make effective admission control decisions, with a guaranteed completion of jobs before their deadlines.
- Bid Generation mechanism for clusters in a competing contract/net model economy, enabling them to reach equilibrium price by approximation in the absence of the knowledge of global market conditions.

1.1 Thesis Organization

The thesis is organized as follows:

In Chapter 2, we explore the various economic models that can be applied to the grid environment, with appropriate references to related work. We try to justify why economic models can be used in the computational grid setting in the first place and summarize the previous and ongoing research in this area.

We also present a brief introduction to *Faucets*[11], the resource discovery and account management system with both pricing and bartering capability for resources. *Faucets* system is currently deployed on the clusters at the Department of Computer Science at the University of Illinois at Urbana-Champaign, to facilitate resource allocation and job management. In order to add more capabilities to *Faucets* and research the strategies required to deploy it in the computational economy model , We implemented the simulation environment of *BidSim* that enables modeling of clusters with different scheduling and bidding strategies for admission control. The components of *BidSim* are described in the same chapter.

Chapter 3 covers Bid Generation mechanism for clusters in detail, comparing it to the strategies employed by bidders in traditional closed-bid auctions.

Chapter 4 describes the experiments and the analysis models used to compare the implemented bidding strategies.

Issues that need further research for the deployment of strategies in real world computational markets are presented in Chapter 5.

Chapter 2

Computational Grid as a Market Economy

> *A market is a medium or context in which autonomous agents exchange goods under the guidance of price in order to maximize their own utility.* [20]

In order to apply the term *market* to the *Grid*, the first step is to define the terms that make up a market, i.e., *goods*, *price*, *agents* and *utility*.

- Goods

  In the context of the *Grid*, goods can be defined either as the services offered by clusters and workstations to users or as the resources like cpu time, disk space, memory etc. In order that goods from different producers are interchangeable and comparable, there should be standard accepted units of sale. For the sale of *processing power*, the units could be either *cpu time slots* on a processor or dedicated *processors*[2].

- Price

  Price is the amount paid by the consumers in return for the resources or services of the producers. In the computational grid, it can be interpreted as the currency value a client is entrusted to pay when a job finishes before its deadline. If the cluster cannot meet the deadline of the job, it may either receive a zero amount, a pre-specified amount or even a negative amount (cluster pays the client a penalty) depending on the contract. In a simple model, there is a constant price paid as long as the job finishes before its soft deadline and then the price decreases linearly until it reaches a value of zero at the hard deadline. After the hard deadline, the cluster does not receive any money for finishing the job. In non-commercial markets (universities, scientific research organisations), price is calculated in terms of service units.

More importantly price aids the agents in a market to make several decisions. For example, when users have the availability of multiple clusters providing the same service, they can choose the cheapest one based on the bid returned in an auction. Similarly, when the job's requirements specification contains an initial valuation of the services requested, a cluster can use that value as the job's priority during scheduling. This concept, introduced in [12], is based on the fact that a user would be willing to pay more money for the jobs he considers valuable. (Users are restricted from always providing a high value by their limited budget.)

- Autonomous Agents

  Agents in a market are the *producers* who are interested in selling the goods and the *consumers* who are interested in buying the goods. In the grid scenario, producers are centers operating PCs, clusters, workstations and supercomputers ready to offer the processing power in return for a price; and consumers are the users with computation and/or data intensive jobs and a budget to expend. This budget as discussed earlier could be a monetary unit or just an account unit. The agents are autonomous because they take independent decisions to maximize their own good, without concern for the global good. It should be noted that clusters compete against each other for jobs while jobs compete with each other for the processing power. For this reason, in a healthy economy rational self-interest decisions lead to optimal globally efficient allocations.

- Utility

  Cluster's utility function may be the utilization of its processors. Jobs are scheduled in such a way as to maximize utilization. In the presence of two jobs, cluster might give preference to the longer one. When each job comes with a profit, the objective function could be maximizing the total profit gained. In this case, a job with higher profit is given the higher priority. User might aim either for cost optimization, meaning he wants the cheapest resource, or for time optimization, meaning he wants a resource that completes his request soon.

2.1 Economic Models

In the above section, we defined goods and agents in a Computational Market. Now, how are these goods brought into the market and how do agents interact with each other to facilitate trading? The manner in which this is done depends on what economic model is employed for the system.(For example, who will initiate the negotiation - the buyer or the seller?) The three common economic models used for research in the computational grid context are briefly described below:

- Auctions

  An *Auction* is a commonly employed solution to get the best buyer for the goods in a market. The item that needs to be sold is put up in an *auction* possibly with an intial endowment. In an open-bid English auction [1], buyers compete against each other publicly and everyone is aware of the current bid for the item[1]. A bidder acquires the good either for personal consumption or for resale. Each bidder has his own private valuation of the item. An open-bid auction has the advantage that a bidder knows how much his competitors value the same good. In a closed bid auction, the bids are not disclosed until after the good is sold. In any case, the bidder has sufficient knowledge about the worth of the item. If the item is bought for resale, its bid value also depends on the estimation of future resale value.

[1] common for the sale of art

- Tenders/Contract Nets

  This is a widely used model for the sale of services in a distributed environment [16]. Here the service providers are called contractors since they get the contract for the good in return for the promised fulfilment of an agreed-upon quality of service. It starts with the consumer announcing his requirements and inviting bids from contractors. Interested contractors evaluate the announcement and respond by submitting bids. Consumer evaluates the bids and awards the contract to the most appropriate contractor. Consumer and the contractor then communicate privately to complete the contract. [19]

- Commodity Markets

  Goods of the same type are brought to the market by various suppliers and are regarded interchangeable. Market price is publicly agreed upon for each commodity regarded as a whole and all buyers and sellers decide whether (and how much) to buy and sell at this price. When this model is applied to the Computational Grid, the *compute power as a utility* paradigm is strongly supported in the sense that commodities can be interchanged and consumers need not be aware of the source of the commodity. [21] Commodity Market has a flat[2] or supply-and-demand[3] driven pricing model.

For a detailed explanation of the possible economic models and a tabulation of systems using these models for applications in computer science see [15].

In the following section, I give a brief summary of current research in this area. In 2.3 , I explain the *Faucets* system on which my simulator *BidSim* is based.

[2]once pricing is fixed for a certain period, it remains the same irrespective of service quality
[3]prices change very often based on supply and demand changes

2.2 Related Work

Several systems have employed models that differ in their definition of goods, in the selection of the bids and in their intiators of trade. Below I note how some related research systems differ in their computational economy models.

Spawn [3] uses the *Auction* model to support concurrent applications in a heterogeneous system. The seller of the computational resources starts the auction by specifying the goods to be sold - minimum and maximum time slices, with linearly increasing function that relates time-slice length to the cost. The buyer bids for the resources, bid consisting of a length of time, a quantity of funds and a brief task description. The auctions are sealed-bid, second price. The buyer that wins gets the resource.

Rich Wolski et al. investigate G-Commerce[21] - computational economies for controlling resource allocation in Computational Grid settings. There are two types of producers: CPU producers that sell fixed number of CPU slots to the Grid, and Disk producers that sell fixed-sized files that applications can use for storage. A consumer submits its job by specifying the number of slots of each kind that it requires without specifying the amount of time it needs the slots for. When a producer agrees to provide the desired slots, the job is run till completion to the price agreed upon. Consumer's budget is decremented and the producer's revenue is incremented by the agreed upon price every simulated minute. Consumer's budget is refreshed periodically. They employ the commodities market model based on pricing systems and supply/demand measurements that is guaranteed to converge to equilibrium. Price, Supply and Demand are represented as n-vectors; where n is the number of commodities (CPU, disk, network etc.). For a given price vector p, excess demand z is the difference between the demand and supply for this price level. Equilibrium for economy is established when supply is equal to demand, i.e. a price vector p is at an equilibrium price when $z(p)=0$.

The work above focuses more on achieving a stable equilibrium of prices. However, these models require considerable changes to the current resource allocation mechanisms in order to be deployed in real life. Faucets system described in the next section, follows the contract/net model, which is the most in accordance with how users currently submit their jobs to clusters. By employing the market economy model to faucets, we need minimal changes to deploy it in future computational markets.

2.3 Faucets

Faucets [11] is a resource discovery and account management system that aims at the efficient utilization of compute power in the best interest of the user. It provides the infrastructure for clusters to make admission control and scheduling decisions and for users to access various available clusters with a single account. Further *Faucets* system has scheduling strategies that leverage the adaptivity of jobs with respect to the number of processors they can execute on. Such adaptive jobs can be developed using the Charm runtime system [9], that currently supports MPI (Adaptive MPI [6]) and Charm++ adaptive programs.

Resource services in *Faucets* can be provided in two modes:

- *Pricing:* Resources are sold for a price based on the supply and demand at that point of time. Price refers to the amount paid by the consumer for the use of resources. This adapts well to the economy model where the user is not affiliated to any single cluster but is looking for the best available resource in the market. *Faucets* provides the bid generation and resource selection mechanisms for the realization of the *Pricing* model.

- *Bartering:* Consider a situation where a user is desperate to run a job before an imminent deadline but his home cluster is busy and cannot accommodate it. Such a crunch time might vary from organization to organization. If clusters decide to collaborate and contribute their resources to a shared pool when they are relatively free, while drawing from the pool when they need more power, this particular problem can be avoided. The *Cluster Bartering* model of *Faucets* addresses the economic use of collaborating resources by providing the accounting infrastructure.

I discuss the pricing model in detail in section 2.3.1 since it relates directly to *Market Economy Research*, the primary interest of this thesis. Section 2.3.2 introduces *BidSim*, the simulation environment modeled on Faucets to study its applicability in the grid environment. In the next chapter, I discuss the issues when faucets works in a multiple-cluster market economy and propose the bidding model to calculate prices and acheive market equilibrium. The software components existing in the system are shown in the figure 2.2.

2.3.1 The Economy Model of Faucets

Primary agents acting in the *Faucets* system are the *Client*, the *Faucets Server* and the *Clusters*. The interaction of these agents with each other is depicted in Figure 2.1 and described briefly as follows.

1. User submits his job to the Faucets Server with a standardized specification of QoS (Quality of Service) parameters.

2. Faucets Server authenticates the user with a secure username-password mechanism.

3. In the attempt to find the best resource for the submitted job, Faucets Server sends RFBs (Request for Bids) to all the Clusters that are registered with it.

4. The Schedulers on the clusters decide based on the QoS requirements if they can perform the job and return a *bid* accordingly. The bid represents the amount a scheduler would charge in return for completing the job before its deadline.

5. Faucets Server selects the best bid based on the user's objective function, say minimizing the execution cost, and submits the job to the corresponding Cluster.

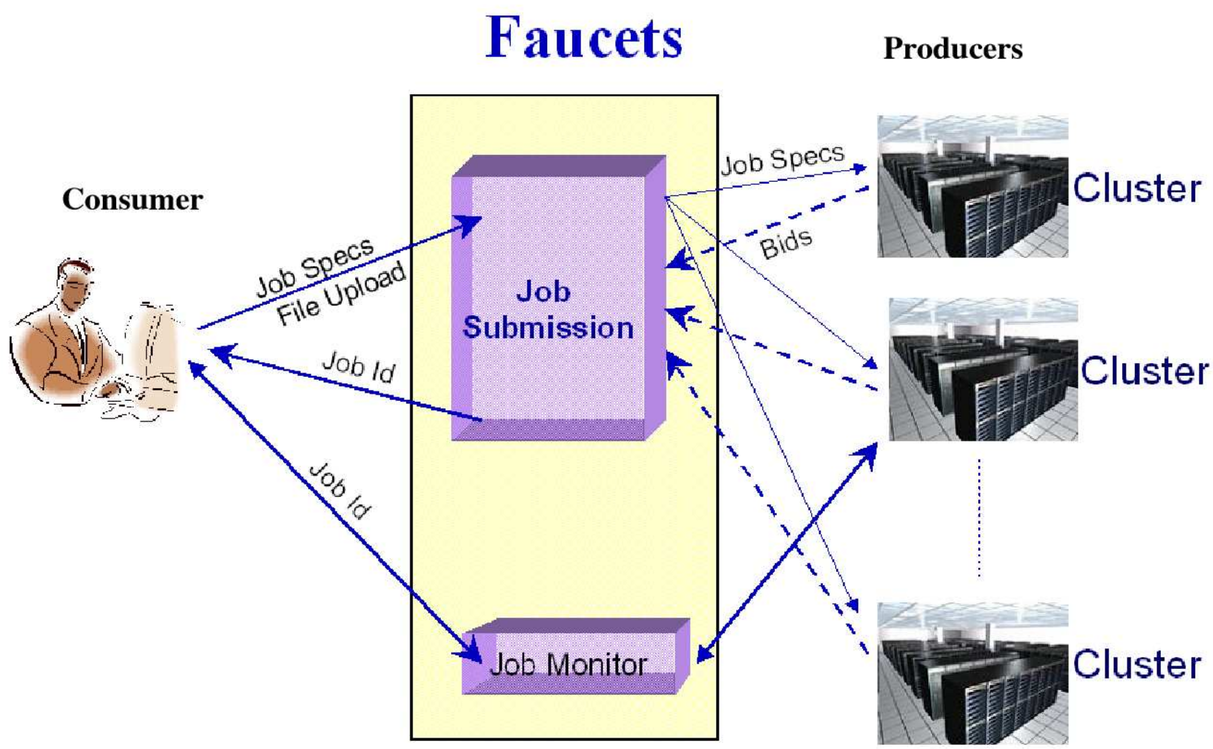

Figure 2.1: Interaction of Agents in *Faucets*

6. Any files that are needed for the successful execution of the job are then transferred from the client to the Cluster's machine.

7. User receives a job id with which he can monitor the job status, retrieve the result files or even kill the job.

2.3.2 BidSim: A Simulation Environment for Faucets

To study the effectiveness and scalability of new scheduling strategies and to explore the applicability of *Faucets* to the Computational Grid, *BidSim* simulation environment has been developed. BidSim allows the creation of multiple clusters with independent scheduling and bidding strategies. It has the infrastructure to create varying loads and jobs with varying requirements. An instance of the simulation starts with the *JobGenerator* creating jobs

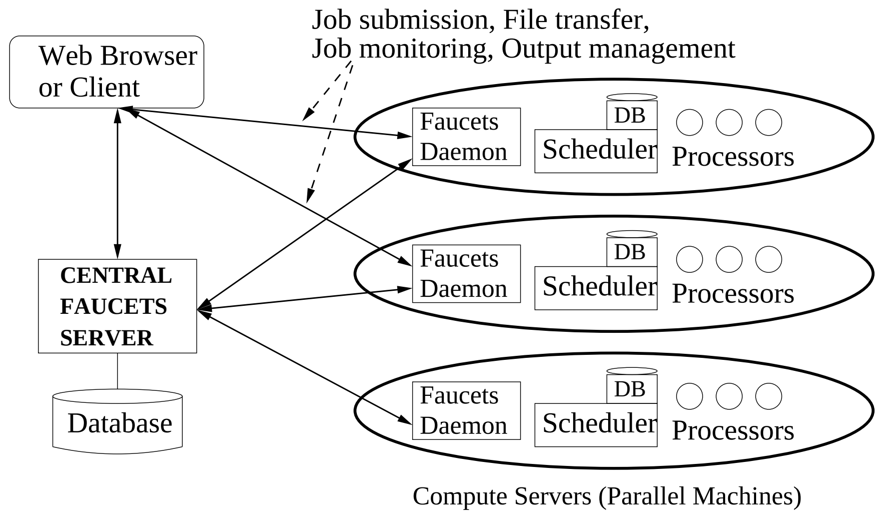

Figure 2.2: *Faucets* System Components

and inserting them into the Event Heap. *Simulator* deletes the events from the heap in the timestamp order and forwards them to the objects they are meant for. New events are added during the course of the simulation. The Simulation ends when all the events in the event heap are processed, which is usually when the last job finishes execution. The components of BidSim that model faucets and that aid in the simulation are described below:

- *Simulator* This class creates the required number of clusters with the specified scheduling and bidding strategies. It also takes care of logging statistical information about the system.

- *JobGenerator* JobGenerator creates jobs with arrival times, deadlines, work and profit metrics following the specified distributions.

- *FaucetServer* This class models the responsibilities of the Faucets Server in the Faucets environment. Jobs are first submitted to this object, which invites bids (RFB events) from the Scheduler objects. Returned bids are evaluated, the suitable cluster aka scheduler is selected and a JOBSUBMIT event is created for the winning scheduler.

- *Scheduler* A scheduler manages both job admission control and allocation of the cluster's processors to the accepted jobs. It is characterized by a scheduling strategy and an agent that aid in taking efficient job management decisions. Scheduling Strategy checks for execution feasibility (defined in section 3.1.1) and Agent calculates the private valuation (defined in section 3.1.2) of a job, to combinedly generate a bidmultiplier to be returned to the FaucetServer.

- *SchedulingStrategy* This class dictates the scheduling policy employed by a cluster. New strategies are implemented as subclasses of the abstract class SchedulingStrategy, that differ in their algorithm to allocate processors to the queued jobs based on individual priorities. For example, non-profit strategies are concerned with maximizing system metrics like utilization while profit strategies emphasize on winning more revenue even if it leads to a slightly lower utilization.

  Faucets uses adaptive job schedulers [10], to leverage the flexibility in the processor allocation of adaptive jobs. Adaptive jobs, as opposed to non-adaptive traditional jobs, can be shrunk to their minimum required processors or expanded to their maximum required processors from the current allocated processors. This allows the scheduler to take scheduling decisions that lead to the efficient utilization of the cluster.

- *Agent* Each scheduler has an Agent associated with it, which dictates the bid value to be returned for a job. It uses *Price Graphs* for mapping the utilization returned by the scheduling strategy to the bid multiplier. It also has the mechanism to dynamically adapt the mean cpu price to the market changes by monitoring the percentage of bids won in a given amount of time.

- *Events and Event Heap* The events existing in the current simulation system are:

  1. ARRIVAL: marks the job arrival event for the *FaucetServer*

  2. RFB: marks the Request for Bid event for a *Scheduler*

  3. BID: returned Bid by the *Scheduler* to the *FaucetServer*

4. JOBSUBMIT: job submit event for a *Scheduler* created by the *FaucetServer*

5. JOBDONE: job completion event for the *Scheduler*

6. SCHEDULEJOB: event that forces the *Scheduler* to reschedule the jobs in the queue

Event heap is a minheap datastructure that stores the events ordered by their timeStamp.

BidSim is an extension to the adaptive job scheduling framework implemented by Sameer Kumar for his Masters thesis. Hence it uses the same API for the scheduling strategies as presented in [10]. The experiments performed using *BidSim* and the analysis of the results are presented in Chapter 4.

Chapter 3

Market Economy Agents

The primary players in a computational market economy are the users and the clusters. Each player has his own set of responsibilities. For example, users are responsible for the accurate advertisement of the job's requirements while clusters are responsible for abiding by the QoS contracts and finishing accepted jobs by their deadlines. Market economy agents require mechanisms by which they can efficiently realize such responsibilities.

Consider the scenario when several clusters are competing against each other for jobs in a contract-net computational economy model. How does a cluster determine what value to bid in order to win the job's contract? We refer to the problem as *Bid Generation* and attempt to find an effective solution through this thesis. Section 3.2 discusses briefly the process of *Bid Selection* that occurs on the user's side in response to *Bid Generation*.

3.1 Bid Generation

Bid Generation in brief is the process by which a cluster evaluates the submitted job's QoS requirements to determine if the job can be accommodated and to calculate the price that gets the worth of the used resources, if not some profit. It basically comprises of the following two steps:

1. Execution Feasibility

   When the RFB (Request for Bid) for a job is submitted to the cluster, it first needs to decide whether the job can be accepted or not. This decision may be based on several factors. Firstly, a cluster needs to have the required hardware (ex: processing power, memory, disk space etc.) and software (ex: operating system environment, compilers etc.) necessary for the successful execution of the job. Even then, the job might have further constraints like deadlines and profit metrics. If the cluster cannot perform the job within its deadline, or if the budget is too low, the job might not be acceptable. (In the model used in this thesis, job's budget is not considered).

Cluster's scheduling policy plays an important role in determining this acceptance criteria. FIFO scheduling policies of the clusters today, that do not consider the Quality of Service (QoS) parameters of the jobs like deadlines and adaptivity[1], are lacking in their ability to do effective admission control and efficient resource allocation. In section 3.1.1, we present one such scheduling strategy, namely the *GanttChartStrategy*, that leverages the job's QoS parameters for maximizing system utilization.

2. Private Valuation

When objects are acquired for personal consumption (as opposed to resale) in an auction, bid value is predicated by its private valuation[2] to the bidder. Once the cluster is sure of the execution feasibility, it needs to calculate the private valuation of the job which indicates how important it is for the cluster to win the bid. This might again depend on various factors, such as the current utilization of the cluster. Such a valuation would aid the cluster to generate a bid and is discussed in section 3.1.2.

3.1.1 Execution Feasibility

Scheduling Strategies in vogue accept all the jobs that are submitted to them. These jobs wait in the queues until they are scheduled in the order of either arrival, priority, deadline

[1] minimum and maximum processors

[2] As opposed to common value which depends not only on the private value but also on the valuation of the prospective buyers

or some other parameter depending on the strategy used. Such strategies neither guarantee the efficient utilization of the available resources, nor do they promise the completion of jobs within their deadlines. In order to use clusters in a market economy, some kind of QoS contracts should be committed to the user. One way to do that would be to accept a job only if it can be scheduled immediately, as in the BestFit Strategy presented in [10]. Another is to accept a job if it can be executed some time in the future before its deadline. This requires an estimation of the work in the current queues and the amount of free resources that would be available in the future.

GanttChart Strategy is a deadline driven lookahead scheduling strategy that accepts a job if it can be accommodated into the job queue without violating deadlines of the running and the previously accepted jobs, while completing the new job before its deadline. A ganttchart datastructure that stores the number of free processors in the window of time from now to the latest deadline of the queued jobs is used to make such a decision.

When an RFB event comes, the scheduler creates a gantt chart representing fixed size time intervals and uses it to calculate the number of processors available in each interval. If the job can be executed before its deadline, with the current load, the estimated utilization value of the system from now to the new job's deadline is returned to the Scheduler class. The Scheduler, with the help of the Agent, converts that value to the bid value for the job. GanttChart Strategy employs a similar scheme while scheduling the queued jobs.

A Comparison of the GanttChart Strategy with the BestFit Strategy with respect to system utilization, rejected jobs and earned revenue is presented in the section 4.2.

3.1.2 Private Valuation

When a cluster determines that it can finish a job before its deadline, it needs to decide what to bid in order to obtain the job. This bid generation may be dependent on several factors as described in the following subsections. Before that, certain terms that dictate the meaning of revenues and bids need to be defined.

A *Service Unit (SU)* defines one CPU hour on a cluster. A *Standardized Service Unit (SSU)* refers to this value normalized across all the clusters participating in the system.

Jobs' execution units in *BidSim* are specified in terms of *SSU*s. For example, a *job length* of 200 means that it takes 200 standardized CPU hours for the job to complete its execution.

Each cluster has a *median charge* associated with it, which determines the normal value a cluster desires to charge per *SSU* used.

Bid multiplier is a variable value, typically between 0 and 2, that is multiplied with the *median charge* of a cluster to get its current charge per *SSU*. Thus, when the *bid multiplier* is low, the cluster gets a lower *revenue per job* than when it is high and vice versa. Varying the *bid multiplier* with respect to cluster utilization and system loads, has been discussed in section 3.1.2. It has been shown in section 4.3 that basing multiplier variations on these factors increases a cluster's overall revenue.

Revenue gained by a cluster per job is defined by the following equation:

*revenuePerJob = bidMultiplier * jobLength * medianCharge*

So a job with execution time of *100 SSU*s when submitted to a cluster with a *median charge* of *$2 per SSU* provides a total *revenue* of *$100* to the cluster on the its successful completion before deadline, if the *bid multiplier* at the time of submission is *0.5*.

With these definitions, we proceed to the next sections for the discussion of how utilization of the cluster and the system load affect the private valuation of a job and hence the *bid multiplier*.

Utilization

It is intuitive that a cluster that has idle time and is in want of jobs values a job higher than when it is busy. In other words it might be ready to do the job for lower value than its *median charge*. On the other hand, whether the cluster should raise its price to more than the *median charge*, if it is busy is still a question. Intuitively if the cluster can do the job, it should maximize its chances of winning the bid by keeping the price as low as possible.

However we investigate this question in section **??** and discover that increasing the charge at higher utilizations indeed increases the revenue of a cluster.

This concept of varying the *bid multiplier* and consequently the *revenuePerJob* can be represented as *Price Graphs* shown in figure 3.1. X-axis shows projected cluster utilization for a period between the current time and the new job's (job that requested for bids) deadline. Y-axis shows the bid multiplier between 0 and 2.

Price Graphs *1*,*2* and *3* shown in the figure 3.1 start from a low bid multiplier under low utilizations, linearly increase the value until it gets to a steady utilization and at higher utilizations, increase again to take advantage of the heavy loads. Price Graph *4*, uses the same strategy as others in the initial phases utilization, but even in the higher loads it keeps a constant bid multiplier without increasing it. How these strategies behave when competing in a market with varying loads is shown in the section 4.3.

System load

A cluster, as explained above, uses the *bid multiplier* parameter to adapt to the local system changes, thereby gaining increased revenues. However this method of private valuation would be incomplete if the cluster does not adapt to the global system state. Such an adaptation can be achieved by the corresponding changes in the parameter *median charge*. In the absence of a global system state advertisement, this would be possible by learning from the percentage of bids won by the cluster with respect to the total number of favorable bids that it sent. For example, if a cluster is losing most of the bids, it is likely that its charges are too high for the current market condition. Experimenting with the variations in the *median charge* and analyzing its effect on price stabilization is the proposed extension of this thesis, and is discussed briefly in section 5.2.1.

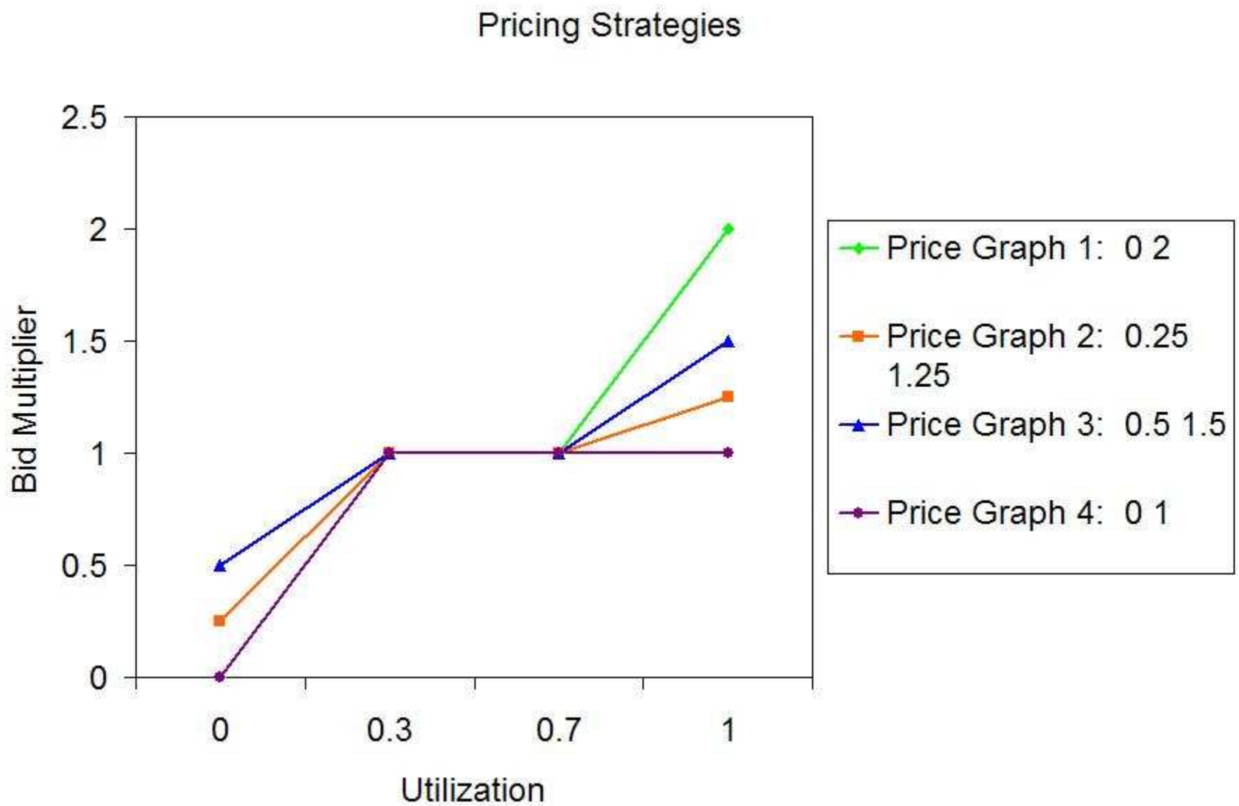

Figure 3.1: *Pricing Strategies* mapping cluster utilization to *Bid Multiplier*

User's Asking Price for the Job

Most of the time users do not have any idea of how much they should be paying for the used resource. But when they do, they can have the option of providing each job with a *budget*, the maximum value they are willing to pay for the job. In such a case, clusters have one more value on which they can base their bids. They can make bid multiplier adjustments to meet the job's *asking price* or choose not to participate in the auction if they consider the *asking price* too low. If no cluster is ready to accept the job below the *asking price* the user needs to increase the budget or the deadline before re-submitting the job to the system [17]. In our current experiments and evaluations, we do not use this parameter for the auction model.

3.2 Bid Selection

Selection of a bid depends upon the information made available through the bid. If the bid is a simple number representing cost as in the case of BidSim, the user or the faucet server can choose the cheapest bid to submit the job. If the user wants an earlier finish time, he can change the deadline of the job to a lower value and be guaranteed that his job would complete before that time if a cluster returned a favorable bid.

However if the bid were complex, probably specifying a range of finish times with their corresponding costs, the user would have more flexibility in the process of bid selection. The Nimrod/G resource broker [5, 14] runs in two user modes,*Optimize for Cost* or *Optimize for Time*. As the names suggest, the broker attempts to discover resources suitable for the user's set of jobs, trying to minimize either the cost or the overall finish times respectively.

As is evident from the above discussions, the processes of Bid Generation and Bid Selection are tightly tied with each other and with the QoS parameters that are exchanged between the user and the cluster.The more the research in simplifying and standardizing extensible QoS specifications, the easier it would be to deploy market economy on the computational grid.

Chapter 4

Analysis of Strategies

Analysis of the Bid Generation techniques proposed in the chapter 3 in the context of system utilization and revenue gained by the clusters is presented in the following sections. First, the GanttChart scheduling strategy is compared with the BestFit strategy of [10] in a single cluster scenario i.e., there is just one cluster in the system to which all the jobs are submitted. Then we analyze the bidding mechanism by comparing the performance of clusters using various price graphs in a system with 3 clusters competing against each other. We approach the experiments with the following questions in mind:

How does GanttChart Strategy perform w.r.t the BestFit strategy on a single cluster?

How do using lower bid values affect the cluster revenue and market economy?

How does revenue vary for different price graphs used by the bid generation strategies w.r.t varied loads?

4.1 Parameters in the Experiments

Before getting to the experiments and the results, here is a brief explanation of the parameters varied in the experiments.

- Job Arrival Rate

  I perform the experiments with jobs having exponentially distributed mean arrival times. By increasing the load factor, I vary the rate at which jobs enter the system. For example, a load factor of 1 means that a new job will enter the system approximately as soon as the existing job finishes execution. When the load factor is 2, a new job enters the system when the existing job is half done i.e. jobs arrive twice at the rate that the system can handle them.

- Job Execution Time

  Job execution time is the product of the number of iterations of the job and the time for each iteration. Number of iterations per job is an exponentially distibuted random variable with a mean value of 100 [1]. Iteration time is taken as 0.60 units of simulation time.

- Job Deadlines

  Each job comes with a deadline within which it needs to be finished. This is implemented as a scalar value added to the job's submission time to get the absolute value in time.

- Job Processor Requirements

  Sequential jobs run on one processor, hence their processor requirement is 1. Parallel jobs can be run on more than one processor, hence they have two parameters: minproc[2] and maxproc[3]. Parallel jobs can be traditional or adaptive. Once a traditional job starts executing, the number of processors assigned to it cannot be changed. Whereas the number of processors assigned to an adaptive job can be varied during its execution as long as it does not go below the minproc requirement of the job.

  In the experiments, minproc of the jobs is a random variable with a mean of 10. maxproc for traditional jobs is set to the same value as their minproc and that of adaptive jobs is set to 64.

[1]modeled similar to the experiments of [10]
[2]absolute minimum processors required to execute the job
[3]the extreme number of processors that a job can scale up to

- Cluster's revenue per Job

  Revenue gained by a cluster for one job is calculated using the equation below as explained in the section 3.1.2.

  *revenuePerJob = medianCharge * bidMultiplier * jobLength*

  In the experiments, *medianCharge* of all clusters is assumed to be the same and *bidMultiplier* varies depending on the pricing strategy used by the cluster.

- Number of Processors per Cluster Number of processors per cluster can be varied in the experiments. But since in my comparison I wanted all the clusters to be of the same processing power, I used the value of 64 processors per cluster.

4.2 Performance of GanttChart Strategy

First I would compare the performance of the presented GanttChart scheduling strategy with the BestFit strategy of [10] w.r.t to traditional and adaptive jobs. Both the GanttChart and the BestFit strategies make use of the adaptivity of the jobs (Adaptive Job Schedulers [10]).

The experiments consist of a single cluster with 64 processors, being submitted 10000 jobs at a load factor varying from 0.05 to 1.1. In one run, the BestFit Strategy is used and its results are labelled BS in the graphs. Another run has the same parameters except that the cluster uses GanttChart Strategy. It can be seen in the figures 4.1 and 4.2 that GanttChart strategy has better utilization and acquires more profit when compared to the BestFit strategy especially at higher system loads. At low loads the amount of work entering the system is less than the processing power available. In such cases, the scheduler does not need to look into the future to decide whether it can execute the job or not. In most cases, the job can be accepted and executed at the same time as its submission. However, at higher loads, when the cluster cannot perform the newly arrived job at that point of time, using a ganttchart for making the decision helps in acquiring higher utilization and revenues.

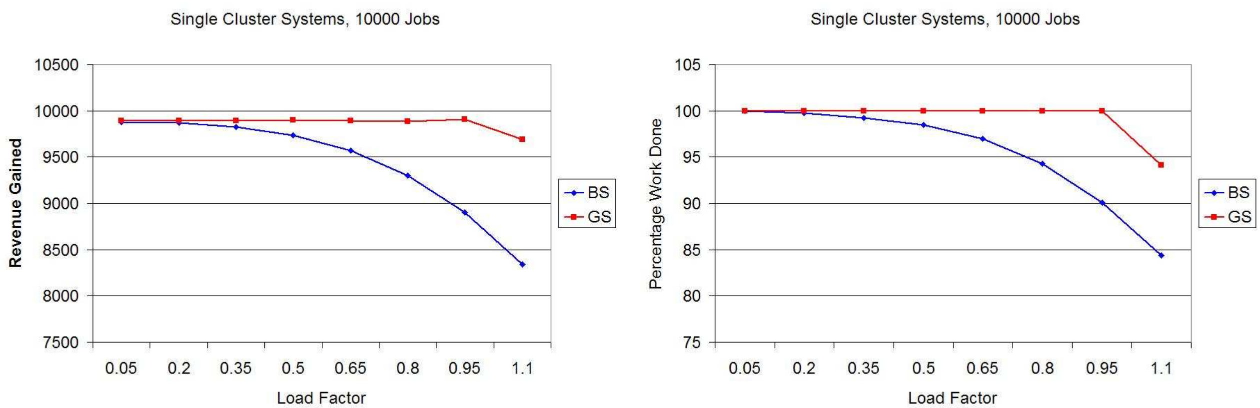

Figure 4.1: Total revenue gained and the amount of work done for the same period of time and same set of jobs is higher in case of *GanttChartStrategy* when compared to *BestFitStrategy*

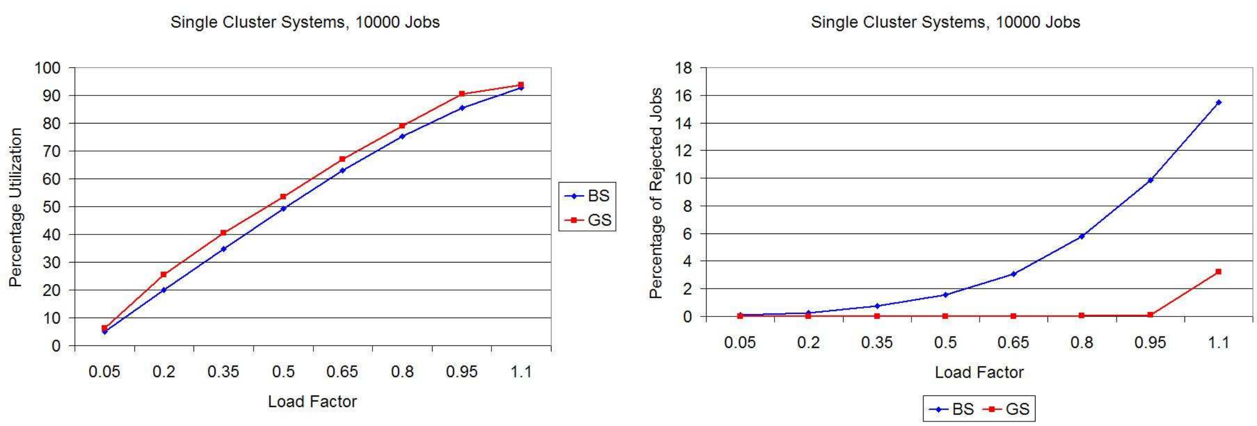

Figure 4.2: Comparison of *GanttChartStrategy(GS)* with *BestFitStrategy (BS)*. Utilization of the system is higher and percentage of rejected jobs is lower in case of GS, especially at higher load factors

Figure 4.3 shows similar results for non-adaptive jobs.

4.3 Bidding Strategies

As noted earlier, the next set of experiments aims at comparing the various price graphs presented in section 3.1.2. System has 3 clusters competing with each other for 30000 jobs at loads varying from 0.05 to 1.4. Graphs are labelled as *0 2, 0.25 1.25, 0.5 1.5, 0 1* representing the clusters using pricegraphs *1,2,3* and *4* shown in the figure 3.1.

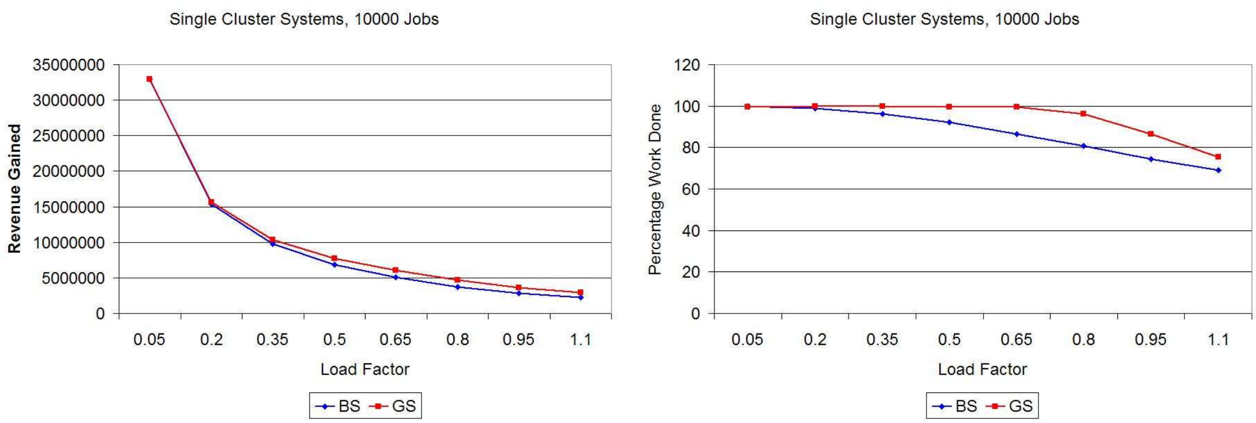

Figure 4.3: Total revenue gained and the amount of work done by using *GanttChartStrategy* and *BestFitStrategy* for traditional, non-adaptive jobs

In the first set of experiments(case *lowhi*), system consists of clusters using pricegraphs *1*, *2* and *3*, while the second set of experiments (case *flatend*) has clusters using pricegraphs *2*,*3* and *4*. Pricegraph *4* differs from pricegraph *1* in the higher utilization phase, where instead of linearly increasing as in *1*, the price remains constant.

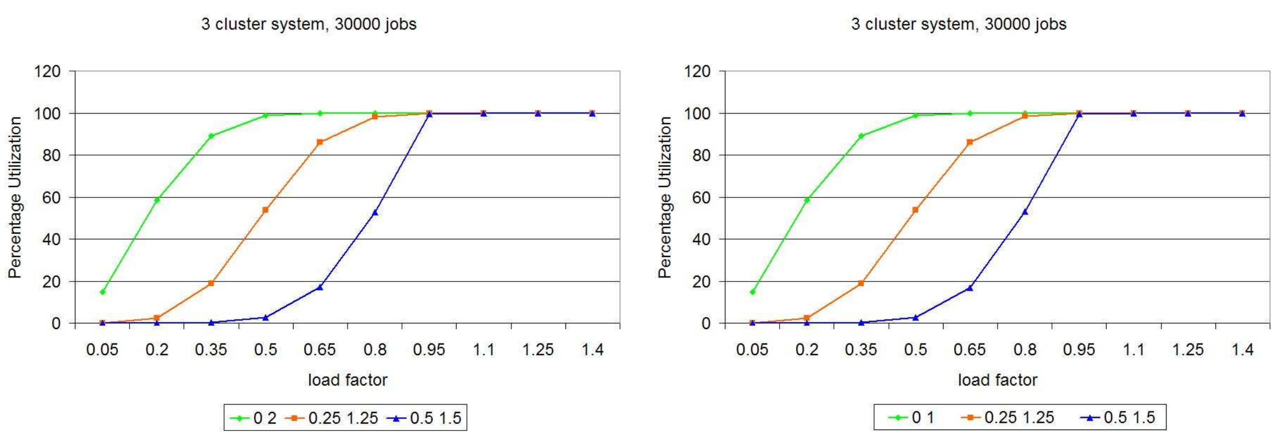

Figure 4.4: Percentage Utilization for clusters with different price graphs in a 3 cluster system

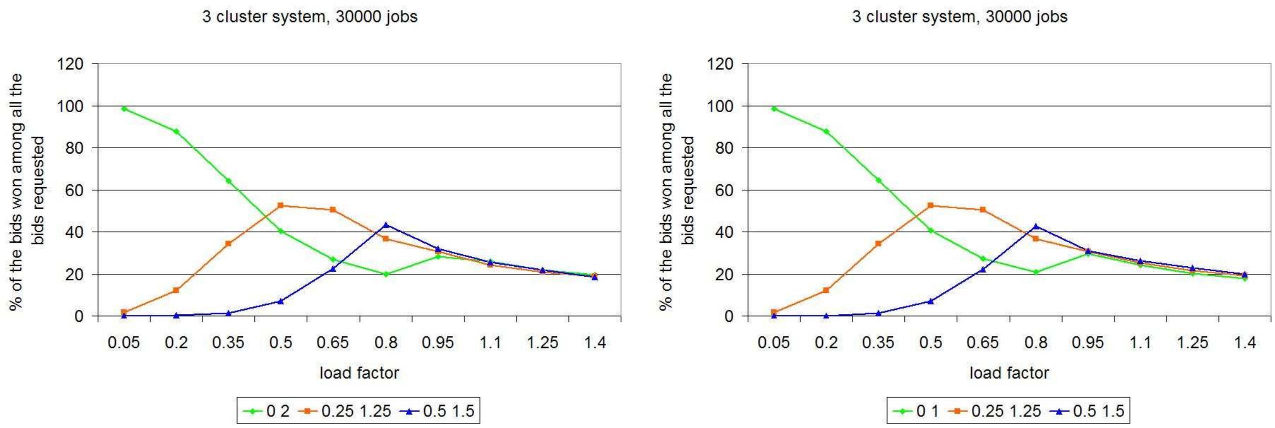

Figure 4.5: Percentage bids won out of the total *Requests For Bids*

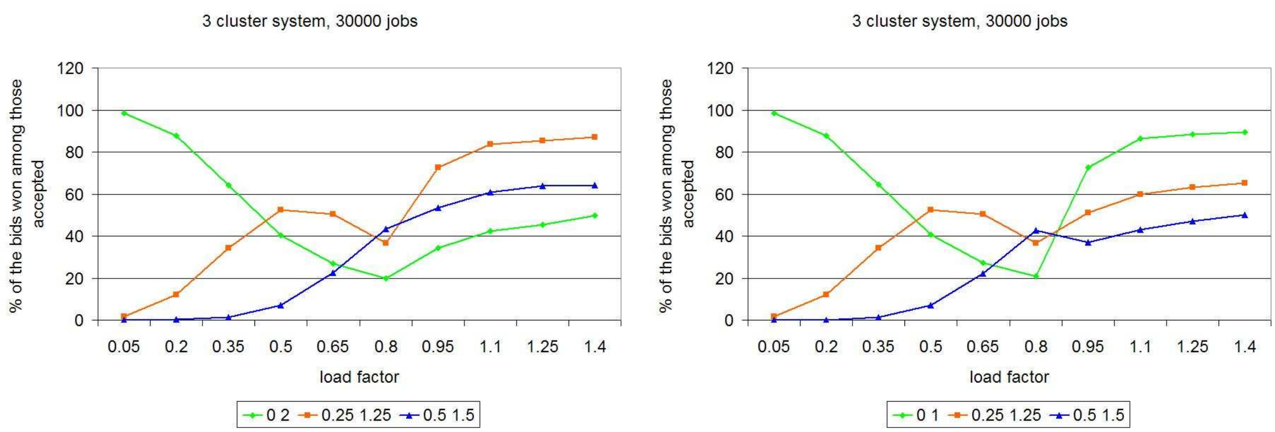

Figure 4.6: Percentage bids won out of the favorable bids sent

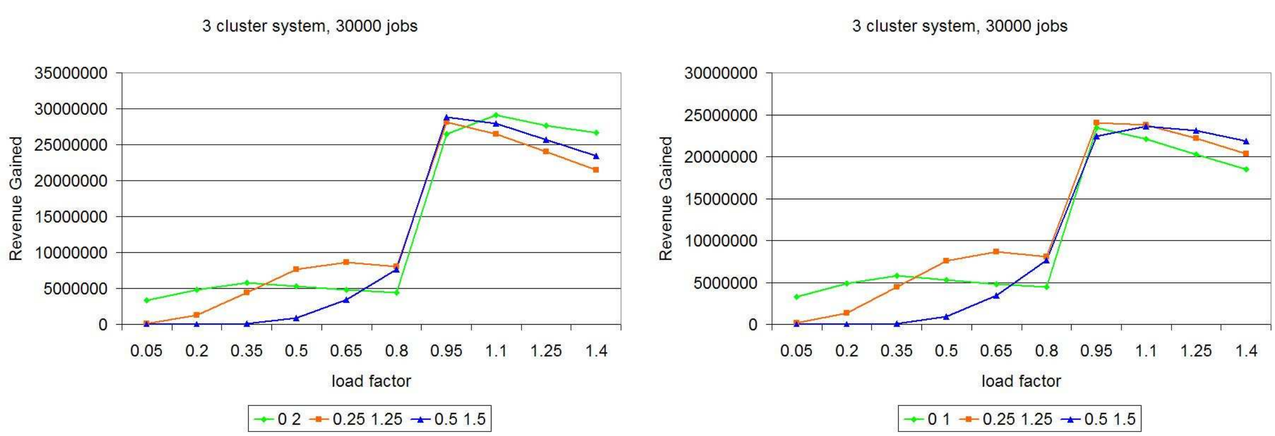

Figure 4.7: Gross revenue gained by each cluster in the system

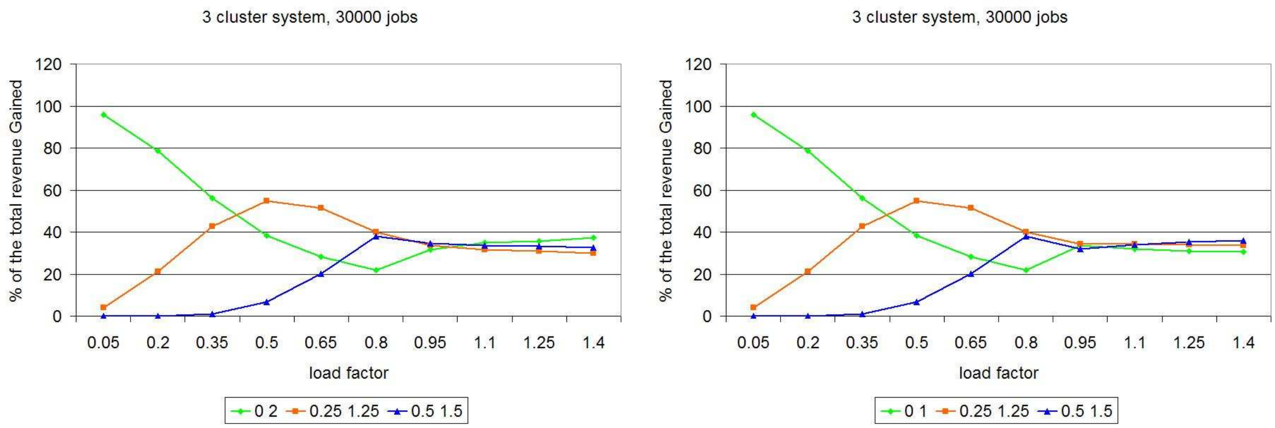

Figure 4.8: Percentage revenue gained by each cluster with respect to the total revenue of the system

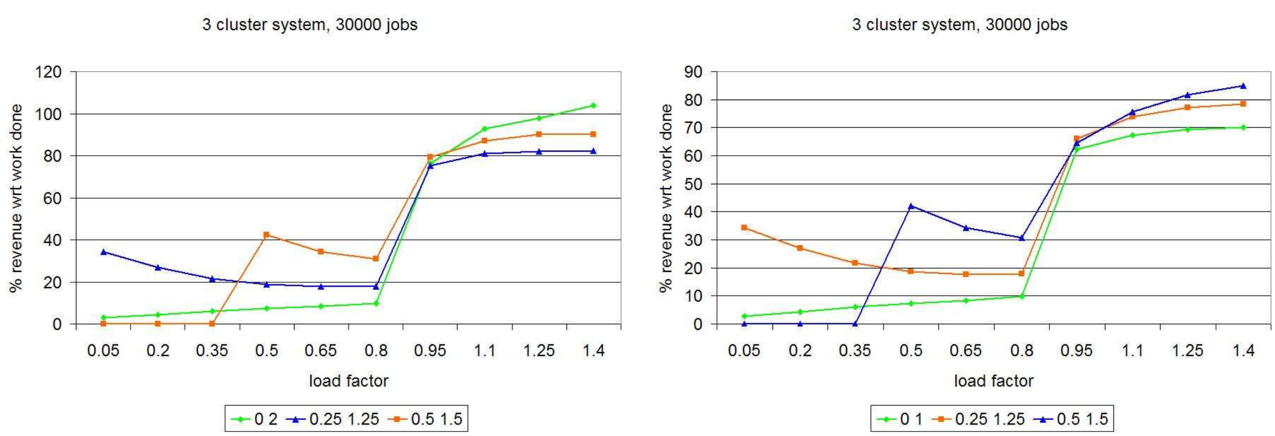

Figure 4.9: Percentage revenue per amount of work done by each cluster

Chapter 5

Conclusion and Future Work

5.1 Summary

This thesis presented the simulation environment of *BidSim*, which is modeled on the *Faucets* system architecture. *BidSim* allows the creation of multiple clusters each running their own scheduling and bidding strategies. Using *BidSim*, I created market scenarios in a contract-net model and analysed the proposed strategies in multi-cluster environments. Results show that GanttChart strategy that uses a lookahead for scheduling jobs gains better performance with respect to the BestFit strategy, that controls the admission of jobs based on the current system state. In the market environment, where clusters are bidding against each other for jobs, it is seen that the one bidding lower values wins more jobs and consequently more revenue at low system loads, while the one bidding higher values gains more revenue at higher system loads. BidSim architecture can be further explored to research the desired characteristics for a bid generation algorithm.

5.2 Future Work

Certain issues that were not addressed in the experiments and analysis, where further research would be required is described in brief in the next few sections.

5.2.1 Adaptive Bidding and Price Stabilization

In Chapter 4, it has been shown that a cluster that bids the lowest in a market wins the most bids when the system is at low load. Under higher load factors however, clusters bidding higher would in effect gain more revenue than those bidding low values. So how can a cluster in the market make sure that it is in the lower bid range under low system loads and in the higher bid range when the system is heavily loaded? In the absence of a global system state advertisement, this would be possible by learning from the percentage of bids won by the cluster with respect to the total number of bids that it sent.

If the cluster is losing most of the bids, it is likely that its bid is too high for the system's load factor. Similarly if it is winning most of the bids, its bid value is too low. In either case, the cluster can then adjust its price to a lower or a higher value respectively until it wins a considerable percentage of bids. This strategy would make the clusters bid lower and lower until supply equals demand. If supply is much greater than demand, the prices will stabilize at a value of zero. At equilibrium, when supply equals demand, the system should reach a steady state and prices should be stabilized. When all the clusters sell their resources at the same price, the jobs will be distributed equally among all of them, as shown in the figure fig:equalbids. In this graph, the two competing clusters send a constant bid multiplier of 1 for any job that they can accept, thus leading to an equal distribution of load and profit between them. The idea explained above can be experimented using BidSim for various loads and if it can be proven that this strategy indeed leads to market equilibrium and price stabilization, then complex algorithms for price determination can be avoided.

5.2.2 Uncertainty in the Real World Scenarios

In the simulation environment, jobs are always processed sequentially in the order they arrive. This means that if a cluster bids for a job, it is guaranteed to finish execution before its deadline when submitted. But in a real environment, there might be a delay between the

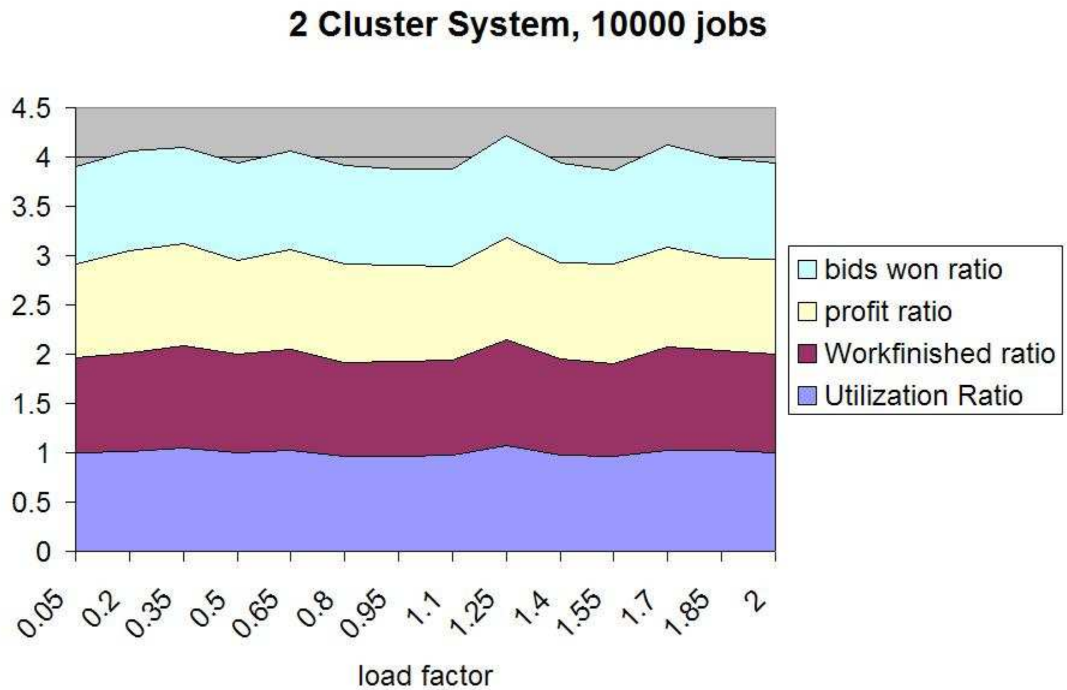

Figure 5.1: Equal Distribution of Jobs and Profit among Clusters bidding constant values.

request for bid and the actual submission of the job.

For example, consider the following scenario: *Job A* sends an RFB to *cluster 1* and *cluster 1* replies with a favorable bid; *Job B* then sends an RFB to *cluster 1*, which makes its admission control decision and sends a favorable bid to *Job B*. This decision does not include *Job A* in the queue because it has not been submitted yet to the cluster. Imagine now that both A and B are submitted to *cluster 1* and they cannot both be finished before their deadlines (either A or B or both miss their deadlines). In such a case though *cluster 1* promised to do the jobs before the deadlines, it will fail to do so.

One way to avoid this situation is to use a two-phase commit protocol that checks with the cluster before job submission even though a favorable bid was requested.

5.2.3 Improved Negotiations

As discussed in the section 3.2, the more the information presented in the bids, the more options a user has in bid selection. Instead of returning one fixed value, there should be a standardized mechanism with which clusters can present their status so that better negotiations are possible. Research in the area of specifying Quality of Service (QoS) parameters [4], could probably be utilized in expanding the definition of bids in *BidSim*.

Similar is the case with user's profit metric. Instead of a fixed value, user might be able to present a time-utility graph which maps the importance of the job with the time it gets done. A simple implementation of this idea with linearly decreasing profit metric from the soft deadline to hard deadline of the job was incorporated and presented in *BidSim*. Further, user's utility/profit graph and cluster's price graph could be used to produce a combined graph as the resultant bid.

5.2.4 Auction/Selection on the Cluster's Side

If the cluster gets more than one job while it is making its private valuations, it has one more degree of freedom to base its decision on i.e. which job is more profitable. Rather than evaluating the job against the current state of the cluster, in this case evaluation can also be done against several competing jobs. This kind of approach leads again to the something similar to the commodity markets model, where both the jobs and the clusters submit bids and there is a central or distributed clearing agent that maps the jobs to clusters from time to time.

5.2.5 Job Length and Bid Value

Section 3.1 discusses how cluster utilization and job's profit metric affects the bid value generated. But how should the length of the job affect the private valuation? The profit obtained by the cluster by finishing a job is directly proportional to the length of the job.

Apart from this respect, should the length of the job be considered during private valuation? In other words, should the cluster favor longer jobs to shorter jobs? This again depends on the cluster's current position. If the cluster is relatively idle it may have the same valuation for both long and short jobs. But when the cluster is relatively busy, it can afford to exercise its preference [8]. Job length can be an added parameter in the bidgeneration algorithm and might lead to interesting results in terms of the amount of revenue earned by the cluster.

References

[1] www.agoricsi.com library: Auctions.

[2] N. R. Bogan. Economic allocation of computation time with computation markets. Technical Report MIT-LCS//MIT/LCS/TR-633, 1994.

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

[4] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459:62–82, 1998.

[5] D.Abramson, J.Giddy, and L.Kotler. High performance parametric modeling with nimrod/G: Killer application for the global grid? In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 520–528, 2000.

[6] Chao Huang, Orion Lawlor, and L. V. Kalé. Adaptive MPI. In *The 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 03)*, College Station, Texas, October 2003.

[7] I.Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[8] J.Doyle. A reasoning economy for planning and replanning, 1994.

[9] L. V. Kalé, B. Ramkumar, A. B. Sinha, and V. A. Saletore. The CHARM Parallel Programming Language and System: Part II – The Runtime system. *IEEE Transactions on Parallel and Distributed Systems*, 1994.

[10] Sameer Kumar. An adaptive job scheduler for timeshared parallel machines. Master's thesis, University of Illinois at Urbana-Champaign, 2001.

[11] L.V.Kale, S.Kumar, J.DeSouza, M.Potnuru, and S.Bandhakavi. Faucets: Efficient resource allocation on the computational grid. Technical Report Parallel Programming Laboratory 03-01, University of Illinois at Urbana-Champaign, 2003.

[12] Brent N.Chun and David E.Culler. User-centric performance analysis of market-based cluster batch schedulers. In *Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGRID2002)*, 2002.

[13] James C. Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant V. Kalé. Namd: Biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*, Baltimore, MD, September 2002.

[14] R.Buyya, D.Abramson, and J.Giddy. Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid. In *HPC Asia 2000*, pages 283–289, 2000.

[15] R.Buyya, D.Abramson, J.Giddy, and H.Stockinger. Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press*, 2002.

[16] R.G.Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computer*, C-29:p. 1104–1113, December 1980.

[17] Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra: An economy driven job scheduling system for clusters. In *6th International Conference on High Performance Computing in Asia-Pacific Region(HPC Asia 2002)*, December 2002.

[18] W.T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major seti project based on project serendip data and 100,000 personal computers. In *Proceedings of the 5th International Conference on Bioastronomy*, 1997.

[19] T.Malone et al. Enterprise: A market-like task scheduller for distributed computing environments. *In The Ecology of Computation, B.A. Huberman Ed., 40 Amsterdam, North-Holland*, pages 177–205, 1988.

[20] Paul Tucker. Market mechanisms in a programmed system. Technical report, Department of Computer Science and Engineering, University of California, 1998.

[21] Rich Wolski, James Plank, John Brevik, and Todd Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. pages 46–46.