GridIS: an Incentive-based Grid Scheduling

Lijuan Xiao[+,*], Yanmin Zhu[†], Lionel M. Ni[†] and Zhiwei Xu[‡]

[+]*Software Division, Institute of Computing Technology*
Chinese Academy of Sciences, Beijing 100080, China
[*]*Graduate School of the Chinese Academy of Sciences, Beijing 100039, China*
[†]*Department of Computer Science, Hong Kong University of Science and Technology*
Clearwater Bay, Kowloon, Hong Kong, China
[‡]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China*

xiaolijuan@software.ict.ac.cn, {zhuym,ni}@cs.ust.hk and zxu@ict.ac.cn

Abstract

In a grid computing environment, resources are autonomous, wide-area distributed, and what's more, they are usually not free. These unique characteristics make scheduling in a self-sustainable and market-like grid highly challenging. The goal of our work is to build such a global computational grid that every participant has enough incentive to stay and play in it. There are two parties in the grid: resource consumers and resource providers. Thus the performance objective of scheduling is two-fold: for consumers, high successful execution rate of jobs, and for providers, fair allocation of benefits. We propose an incentive-based grid scheduling, GridIS, which is composed of a P2P decentralized scheduling framework and incentive-based scheduling algorithms. Simulation results show that GridIS guarantees the incentive of every participant to a satisfying extent.

1. Introduction

With the rapid development of networking technology, grid computing [1], which enables large-scale resource sharing and collaboration, has emerged as a promising distributed computing paradigm. In a grid environment, resources are dynamic, autonomous, heterogeneous, and wide-area distributed. Due to these unique characteristics, resource scheduling in grid systems is significantly complicated and particularly challenging.

A considerable amount of work has been devoted to tackling the problem of scheduling for grid computing. Unfortunately, the majority of the previous work [2-9] has focused on optimization with respect to system-centric or application-centric performance metrics. Examples of such metrics include system throughput, system resource utilization, and job response time. These optimization objectives can be achieved only under the condition that all the resources are subject to some kind of centralized control. However, in grid systems, resources belong to respective administrative domains and every domain has full control over usage of their own resources. What is more important is that grid resources are usually not free at all. Charging for resource usage is an incentive for resource owners to deploy their resources into grid systems for sharing and is also practical in the real world. Recently a few research projects [10, 11] have taken profit into account and applied economic methods in grid resource scheduling. As viewed from economics, there are two parties in grid systems: resource provider and resource consumer. It is individual economic behavior of all the participants that accomplishes the resource scheduling. Economic methods are desirable for grid resource scheduling owing to the inherent decentralization of a market in which each participant makes decisions on its own behalf. Nonetheless, most related research projects share the common problem that their scheduling only considers the performance objectives for resource consumers, such as shorter response time or less payment, but neglects the performance objectives for the other important party in the market. Actually, resource providers also have their expectation of benefits, and they will never be passive. Once their expectation fails to be realized, they may quit the market. To build a practical grid, it is of great importance to guarantee every participant with enough incentive to stay and play in it. Therefore, two-fold performance objective that considers the two parities is more reasonable for resource scheduling in self-sustainable and market-like grid systems.

In this paper, we propose a novel incentive-based peer-to-peer (P2P) scheduling, GridIS, with the goal of

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

building a global computational grid that every participant has enough incentive to stay and play in it. On one hand, we employ P2P model as the decentralized scheduling infrastructure. A resource provider enters the computational grid via some portal, and the portal will tell it the proper neighbors. Constructing portals is a popular way used in e-commerce nowadays. When a resource consumer has a job to be done, it sends a job announcement via some portal too. The job announcement then is forwarded throughout the P2P network, and resource providers that receive it will bid for the job. On the other hand, to guarantee the incentive of both parties, we design incentive-based scheduling algorithms to model the two parties' behavior, and study the extent to which the conservation degree, a parameter that reflects resource providers' attitude at competing for jobs, impacts the incentive that the two parities obtain. Simulation results show that GridIS successfully guarantees the incentive of each participant and is a promising solution to build a global market-like computational grid.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 gives a formal problem statement. Section 4 introduces the incentive-based P2P scheduling in detail. Section 5 demonstrates our simulation results. Section 6 draws a conclusion and shows some possible future work.

2. Related Work

In this section, we will give an overview of related work. Since considerable work has been done in scheduling for grid computing, the emphasis here will be put on those based on economics.

Enterprise [12] is a market-like task scheduler for distributed computing environments. Each idle computer sends estimated task completion time as a bid to the client that initiates the task request. Then the client chooses the computer claiming the earliest completion time to execute the task. The scheduler is not practical for a grid environment in which resources are autonomous, because resource providers have no incentive to bid without benefits.

Spawn [13] is an open, market-based computational system that utilizes idle computational resources in a distributed network of heterogeneous computer workstations. When sellers initiate auctions, buyers bid for CPU slots with given funding. The auctions employed by Spawn are sealed-bid, second-price auctions. "Sealed" means that bidding agents cannot access information about other agents' bids, and "second-price" indicates that the amount paid by the winning agent is the amount offered by the next-highest competitive bidder. The objective of Spawn is the fairness of resource allocation: the number of CPU slots bought is proportional to the amount of funding, which is definitely not sufficient for a market-like grid system. Besides, the seller-initiated auction is suitable only for heavily loaded systems. In a lightly loaded system, the user-initiated auction is proved to outperform the server-initiated one [14].

Computer Power Market (CPM) [15] is a market-based resource management and job scheduling system for grid computing. A CPM comprises markets, resource consumers, and resource providers. A market is the mediator between consumers and providers, which mediates all the information from both consumers and providers. CPM takes advantage of real markets in the human society. However, the centralized market server does introduce many limitations, such as single point of failure and limited scalability. Furthermore, the centralized market server requires additional organizations for regular maintenance.

Buyya et al. [10] presented some economic models which have been used in the human society, such as auction model, commodity market model, contract-net model, bargaining model, and bartering model. They discussed the possible directions how the economic models in the human society can be applied to grid computing. The discussion, however, is at high level and no implementation or performance has been presented. Applying these models to grid computing properly is the major challenge.

Nimrod/G [16] is a resource management and scheduling system based on the parameter sweeping system Nimrod. It is built with the Globus toolkit and targets parameter sweeping applications. Nimrod/G comprises a centralized parameter engineer and dispatcher. Every client submits jobs to the parameter engineer and the engineer is responsible for parsing job parameters. The dispatcher distributes the independent subjobs over resources across the whole grid. Nimrod/G makes an attempt to incorporate the economic idea into scheduling. Resources are associated with prices and jobs are given budgets. But in the paper, the authors do not focus on the economic feature, and give no further explanation and implementation of their economic idea over Nimrod/G.

Our previous study on incentive-based scheduling for grid computing was presented in [17]. In that paper, consumers assign budgets to jobs, and choose providers according to the claimed completion time. No price mechanism is involved. In this paper, we modify our model, and it turns out that price mechanism is very helpful to realize our scheduling objectives.

3. Problem Formulation

There are two parties playing in a global computational grid: resource consumers and resource providers. Resource consumers have jobs to be done and are willing to pay for them. Resources providers have computational

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

resources available and desire to sell them for profit. Scheduling enables the interactions between the two parities and maps jobs to resources properly. In the following subsections, we first characterize the two parties, and then state our performance objectives of scheduling for market-like computational grids.

3.1. Resource Consumers

Resource consumers seek computational resources to execute their jobs and pay for them. Jobs here are computational-intensive and the execution of a job usually lasts several hours or even days. When resource consumers have jobs to be done, they submit job announcements that describe the properties of the jobs to the computational grid. A job announcement includes the information of *job length* and *job deadline*. Job length is an empirical value, assessed as the job execution time on a standard platform. Job deadline is the time instance by which the resource consumer desires the job to be finished. All the jobs are independent from each other.

3.2. Resource Providers

Resource providers execute jobs for resource consumers and charge them for the usage of resources. A resource provider manages the computational resources of one domain with some software such as Condor, LoaderLever, Legion, and Beowulf. These resources are typically interconnected by a high-speed local network and protected by firewalls from the outside world. A resource provider can be modeled with three parameters: *capability*, *unit price* and *job queue*.

A resource provider may possess a collection of computers and each computer may be equipped with multiple processors, which is usually the case in the real environment. To clarify the statement and emphasize our main idea, we simplify the model of resource providers as one computer with a single processor. Actually, our scheduling can be easily extended to accommodate aforementioned complicated cases. We evaluate the capability of a resource provider with a widely accepted method [18]. With the simplification, the ratio of the SPEC benchmark rating of the computer to that of the standard platform, which reflects relative capability compared with the standard platform, is considered as the capability of the provider. Unit price refers the price of unit job length. A provider charges for jobs according to unit price and job length. Price plays an important part in the scheduling that applies economic models. Price adjusting mechanism can further help to accomplish performance objectives greatly. Job queue of a resource provider keeps all jobs not yet executed.

Job execution is not time-shared but dedicated. Thus, we can calculate job execution time by dividing job length with the capability of the resource provider. In fact, the execution time of a job on a computer is very difficult to accurately estimate for it is a comprehensive result of CPU capability, memory size, I/O speed, etc. Several papers [19-21] have attempted to address the problem. In our work, we just apply the simple calculation method.

3.3. Performance Objectives

At a very high level, our goal is to build a global computational grid that every participant has enough incentive to stay and play in it. For resource consumers, they are satisfied with the quality of the job execution service and will continue to buy computational resources in the grid for jobs. For resource providers, they make reasonable amount of profit and are willing to deploy their computational resources into the grid for sale. The performance objective of scheduling is to guarantee the incentive of the two parties, thus two-fold. To resource consumers, the high successful execution rate of jobs can be adopted as a performance metric. A successful execution means that a job is finished without missing its deadline. Consumers evaluate the computational grid with the successful execution rate. If the rate is not satisfactory, they may lose faith in the market and quit it. To resource providers, fair allocation of benefits is a reasonable performance metric. Fair allocation of benefits means that the profit of every provider is proportional to its investment or cost. As the cost of a computer is not convenient to evaluate and is not so significant, we use its capability as the substitute. It is acceptable for the reason that a higher capability usually results from a higher cost. Fair allocation of benefits will be a temptation to all the potential providers including not only those with higher capabilities but also those with lower capabilities.

In the rest of the paper, we use the terms provider and consumer for short.

4. Incentive-based P2P Scheduling

In this section, we first describe the P2P scheduling framework, and then present the incentive-based scheduling algorithms.

4.1. P2P Scheduling Framework

Our scheduling framework takes advantage of P2P network for it is characterized as decentralized and scalable. First, every participant in the computational grid is autonomous and acts individually. Obviously, a decentralized scheduling infrastructure is more favorable. Second, due to the dynamics of the grid environment, players

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

may enter or leave at their will at any time. P2P network can handle such dynamics fairly well.

The global computational grid has several portals via one of which a provider can join the grid. When entering, the provider gets the information of designated neighbors from the portal and then connects into the P2P network.

As mentioned before, a consumer submits a job announcement to the computational grid via one portal. Then the job announcement spreads throughout the P2P network similar to query broadcast in an unstructured P2P system [26]. The providers that receive a job announcement will bid for the job. We want to realize the complete competition among all the providers based on two considerations. First, jobs are time-consuming and computational-intensive. So the execution time is sufficiently long that the overhead of executing them on remote computers becomes relatively negligible. Thus, all the providers should have an equal chance to compete for any job no matter where their geographical locations are. Second, the number of providers will not be too large and typically no more than several hundreds for a provider represents an administrative domain within which local scheduling policies are employed. In other words, it competes for the benefits of all the computers in its own domain. Therefore, we assume that the competition among all the providers is complete, i.e., every provider can receive any job announcement. It is well known that blind flooding based broadcasting is a fatal weakness of P2P networks. Many researchers [22-27] have studied to build overlay networks whose topology closely matches the topology of physical networks. Once an overlay network with the desirable characteristic is built, an efficient broadcasting mechanism with good performance can be constructed. The work is beyond the scope of this paper. In the scheduling framework, we can just apply one proper method to construct the P2P network and perform broadcasting.

The P2P scheduling infrastructure enables the effective interactions between consumers and providers, and jobs are scheduled as a result. We detail the whole scheduling procedure of a job as follows.

Step 1. A consumer submits a job announcement to the computational grid, and the job announcement is broadcast to all the providers.

Step 2. On receiving a job announcement, a provider estimates whether it is able to meet the deadline of the job. If yes, the provider sends a bid that contains the charge for the job directly to the consumer who initiates the job announcement; or just ignores it.

Step 3. After waiting for certain time, the consumer processes all the bids received, chooses the provider who charges the least and sends the job offer to the selected provider.

Step 4. On receiving a job offer, a provider inserts it into its job queue. When the job is finished, the provider sends the result to the consumer.

During the operation of the computational grid, network or hardware failure may happen, which will bring out unexpected results. Additional solutions are required to deal with the consequent matter, such as how to recover and whose responsibility is. In order not to divert our focus, this problem is not considered in the paper and will be discussed in a separate paper.

From the steps above, we can observe that the behavior of consumers is relatively simple, since they only need to offer every job to the provider who charges the least. However, to providers the decision-making is fairly complicate. Next subsection will describe the behavior of providers in detail.

4.2. Incentive-based Scheduling Algorithms

The behavior of providers significantly influences the extent to which the incentive of the two parties is achieved. We design three incentive-based algorithms to model the behavior of providers. The job competing algorithm describes how a provider bids when receiving a job announcement in Step 2 of job scheduling. The heuristic local scheduling algorithm is responsible for arranging the execution order of jobs in the job queue of a provider. It starts when a provider receives a job offer in Step 3 of job scheduling. The pricing adjusting algorithm helps a provider to dynamically adjust its unit price properly over the period of its participation in the computational grid.

4.2.1. Job Competing Algorithm. Due to the decentralized scheduling framework, providers make decisions based on local, imperfect and delayed information, which often puts them in a dilemma. For example, if a job announcement comes shortly after a provider sends a bid for the previous job, called unconfirmed job, and if whether to bid for the newly coming job announcement depends on whether to take the unconfirmed job into account, the provider must make a decision. If bidding, one job is likely to miss its deadline in case that both jobs are offered; if not, the result may be that though the provider is very advantageous in competing for the newly coming one, it loses both. Things get more complex when more jobs are involved. There are two extreme attitudes for providers to compete for jobs. One is aggressive. It means a provider never considers the unconfirmed jobs when estimating whether it is able to meet job deadline. This is a risky one, but chances often accompany risks. The other is conservative. It means a provider always keeps the unconfirmed jobs in the job queue for consideration for certain time. This one will never lead to deadline missing, but may lose potential chances, thus profit. It is obvious that different competition attitudes will result in different profit allocations. To study the effects of competition attitude, we define a parameter named *conservation degree* whose value is a decimal varying from 0 to 1. For short,

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

we refer to conservation degree as CD. A provider will insert unconfirmed jobs into its job queue at the probability of its CD. Thus, for aggressive providers, CD is 0; for conservative ones, CD is 1.

Every time a provider receives a job announcement, it starts the job competing algorithm. The algorithm is stated as below.

Step 1. The provider estimates whether it is able to meet the job deadline.

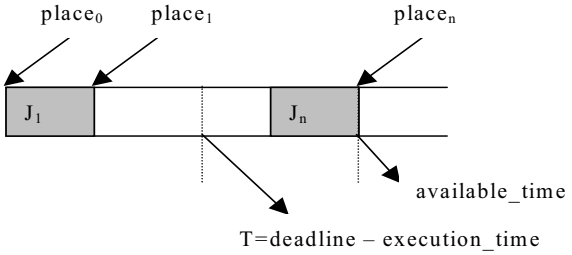

Figure 1. **Job queue of a provider**

As Figure 1 shows, there are *n* jobs in the job queue. If we call the potential new job as J_{n+1}, $place_0$, $place_1$, …, and $place_n$ represent the *n+1* possible places for J_{n+1} to be inserted into. Available time is the time instance when all the jobs in the job queue are completed. T is calculated by subtracting the execution time from the deadline of J_{n+1}. So it is the latest time to begin the execution of J_{n+1} if the provider doesn't want to let J_{n+1} miss its deadline. The estimation is described with the following pseudo-code.

```
If (T > available_time) {
   can_meet = true;
   reordered = false;
   insert_place = placen;
}
Else {
   T is covered by the execution of Ji in the queue
   If (insert Jn+1 at placei-1, none of Ji…Jn will
   miss its deadline) {
         can_meet = true;
         reordered = true;
         insert_place = placei-1;
   }
   Else can_meet = false;
}
```

The estimation algorithm is based on the principle that inserting a new job can never incur new deadline missing. If the variable can_meet is set to true, the algorithm goes on to Step 2.

Step 2. The provider makes a price for the job. The pseudo-code is given below.

```
price = unit_price * job_length;
If (reordered) {
   price = λ * price;
}
```

λ is a decimal larger than 1. When the variable reordered is set to true, price is raised. Generally, jobs are enqueued in the order of their arrival. To meet job deadline, some jobs may be inserted into the job queue ahead of foregoing jobs, which indicates that the deadlines of these jobs are somewhat tight and the jobs need to be given higher priority. Thus it is reasonable to charge more for them. On the other side, a tight deadline also increases the possibility of failing to meet it. So providers raise the price to reduce the chance of being chosen to some extent.

Step 3. The provider sends the price as a bid and inserts the job at the place the variable insert_place indicates at the probability of CD. If the provider chooses to insert and if the job offer doesn't come after certain time, it deletes the job from its job queue. The duration of keeping an unconfirmed job should be as short as possible but long enough to guarantee not to delete offered jobs.

4.2.2. Heuristic Local Scheduling Algorithm. If the competition attitude of providers is not so conservative, jobs may miss their deadlines. To ensure the incentive of consumers to a certain extent, we employ a punishment mechanism that providers are obliged to pay penalty for missing deadline. We define a simple penalty model as shown in Figure 2.

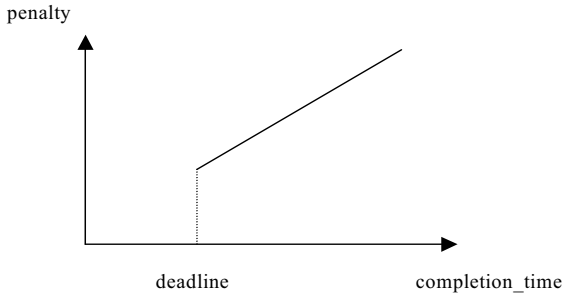

Figure 2. **Penalty model**

Once the penalty model is introduced, providers must take some measures to minimize the loss. What a provider can do is to arrange the execution order of jobs in its job queue. We call it local scheduling. As calculating penalty of all the possible permutations of jobs to find out the one with the least penalty is NP-complete [28], we apply a heuristic approach. The approach is based on the fact that when a job is inserted, the relative order of the jobs in the origin queue is often unchanged. Every time a provider is offered a job that is not kept in the job queue, it starts the heuristic local scheduling algorithm. Obviously, the algorithm is needless for providers whose CD equals 1, because they never let jobs miss their deadlines. The heuristic local scheduling algorithm is described with the following pseudo-code, and Figure 1 is used to illustrate the job queue.

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

```
insert_place = place_n;
//calculate the penalty if insert the job at insert_place
penalty = calculate_penalty(insert_place);
For (place_i = from place_n-1 to place_0) {
   penalty_i = calculate_penalty(place_i);
   If(penalty_i < penalty) {
      penalty = penalty_i;
      insert_place = place_i;
   }
}
insert the job at insert_place
```

As can be seen, the time complexity of the algorithm is linear. When a job is finished, a provider goes on to execute the first job in its job queue.

4.2.3. Price Adjusting Algorithm. Because a consumer chooses the cheapest provider, it is the price mechanism that directs the whole scheduling. That's why we say price mechanism plays an important role. Some research projects try to apply economics theory to study price mechanism in building computational economy, and price equilibrium theory is a hot research topic. Nakai [29] investigated many of the projects and came to an conclusion that none of them have genuinely applied economics to scheduling. After all, computer systems are very different from human economies in many aspects [13]. First, human decision-making is difficult to model and human beings are diverse in their methods for decision-making. Second, human beings make decisions based on the information from the whole society via various mediums. However, it is not the case for computer systems. In addition, decisions made by computers can take place much faster than human decisions. Therefore, to enable computer systems to exhibit meaningful market-like behavior is still an open problem. To emphasize our main idea, we don't take great pain trying to model the price mechanism of the real market. Instead we design a simple and intuitive price adjusting algorithm to achieve our performance objectives.

As our performance objective for providers is fair allocation of benefits, it involves all the providers. It is almost impossible to be realized if every provider just behaves based on the local information. Inevitably, all the providers need to know some global information. In our algorithm, we assume every provider informed with the aggregated capability of all the providers in the computational grid. The information can be attained when a provider enters the grid via a portal, and updated in the same way a job announcement is forwarded.

In a certain period of time, every commodity has a predominant price in the market. For a commodity like CPU cycles, such a price is easier to determine because commodities of this kind have no great difference in quality. We call the price as *market price*, and it acts as a directive. When entering the grid, a provider gets the market price via a portal and sets it as the initial unit price. Then every time a provider is offered with a job or deletes an unconfirmed job, it starts the price adjusting algorithm. The algorithm is stated as the following pseudo-code.

```
r1 = offered_job_length / total_job_length;
r2 = capability / total_capability;
If (offered a job) {
   If ((r1 > r2)) && (unit_price <= market_price)) {
      unit_price = α * unit_price;
   }
}
Else { //delete an unconfirmed job
   If ((r1 < r2) && (unit_price >= market_price)) {
      unit_price = β * unit_price;
   }
}
```

α is a decimal above 1 and β is a positive decimal under 1. Offered job length is the aggregated length of jobs offered to the provider. Total job length is the aggregated length of jobs whose announcements are received by the provider. Total capability refers the aggregated capability of all the providers. Offered job length and total job length rewind when total capability is updated. Our price adjusting mechanism is simple and intuitive -- just to make prices different to differentiate the chances of providers to be chosen and eventually realize the fair allocation of benefits. Furthermore, the algorithm skillfully avoids endless increase or decrease of unit price. Thus the price will fluctuate around the market price, which is acceptable for both consumers and providers.

Providers can choose not to adjust price every time one job is offered or not, but start the algorithm every several jobs. However, if so, the providers are slow to react to the market. The fairness will be degraded consequently.

5. Simulation Settings and Results

We develop a discrete event driven simulator with Java programming language to simulate the computational grid. Consumers and providers are modeled as two kinds of entity in the simulation system. The communications between consumers and providers are performed by event delivery. As for the advance of simulation time, there are mainly two drives: one is the network delay of communications and the other is job execution. In reality, the network delay usually counts in milliseconds, and it is far shorter than job execution time. Therefore, the network delay barely takes effect in the simulation. We don't simulate the network delay in the simulator but focus on implementing the incentive-based scheduling algorithms. The main purpose of the simulation is to evaluate the algorithms. In the following subsections, we describe simu-

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

lation settings and demonstrate simulation results in detail.

5.1. Simulation Settings

5.1.1. System Load. As viewed from mathematics, system load can be defined as a ratio of aggregated load to aggregated capability in a period of time. The following equation gives a formal explanation.

$$system\_load = \frac{a\_a\_r * a\_j\_l * n\_c}{a\_c * n\_p}$$

a_a_r : average arrival rate of jobs

a_j_l: average length of jobs

a_c: average capability of providers

n_c: number of consumers

n_p : number of providers

The value of system load expresses the extent to which the whole system is busy. If in a certain period of time the number of jobs submitted to the system is small and the lengths of jobs are short, then the system load is light; otherwise, the system load is heavy. We study system load because it influences the performance of scheduling inherently. When system load gets heavier, deadline missing rate tends to get greater accordingly. Considering that, we do experiments under different system loads to investigate the performance of our algorithms. Here the system load means an average over the period of a whole simulation. We design a system load model as illustrated in Figure 3.

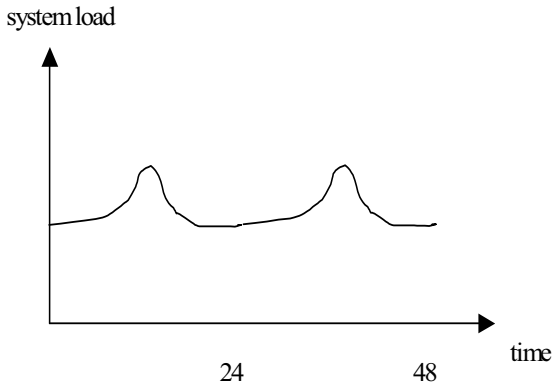

Figure 3. **System load model**

First, system load changes periodically with a period of 24 hours, i.e., one day. Second, a burst of system load happens once a day. Our simulator can accept a given average system load as a parameter and generate the system load of every hour in a day according to the model.

5.1.2. Consumer and Provider. Consumers independently generate jobs from time to time. For each consumer, job generation is modeled as a Poisson process. Thus the interval between two job generations is exponentially distributed. The mean is the reciprocal of average arrival rate of jobs and can be calculated with the aforementioned system load definition equation. The job length is uniformly distributed, and the duration from the time instance of job generation to the deadline is uniformly distributed as well.

The capability of providers is normally distributed due to the observation that during a certain time, computers of some capability predominate in the computer market.

To study the impact of CD on scheduling performance, we devise the simulator so that one simulation can work out results of different CD configurations, while other configurations are the same, including lengths, deadlines, order and arrival intervals of jobs, and capabilities of providers.

5.2. Simulation Results

In our simulation experiments, there are in total 20 consumers and 80 providers. Lengths of jobs average 100, and capabilities of providers average 10. λ is assigned as 1.05, α as 1.1, and β as 0.9. Market price is 1. System load of simulations varies from 0.1 to 0.7 with a step of 0.1. Every simulation runs as long as 100 days in simulation time, working out results of three different CD configurations: 0, 0.5, and 1.

First, we study the incentive of consumers. Our performance objective for consumers is high successful execution rate of jobs. There are two related metrics: failure rate and deadline missing rate. A job fails because all the providers think that they cannot meet the deadline and decide not to bid. The definition of the two metrics is listed below.

$$failure\_rate = \frac{n\_j\_fail}{n\_j\_submitted}$$

$$deadline\_missing\_rate = \frac{n\_j\_miss}{n\_j\_finished}$$

n_j_fail : number of jobs that fail

$n\_j\_submitted$: number of jobs submitted

n_j_miss: number of jobs that miss deadline

$n\_j\_finished$: number of jobs finished

Table 1 and Table 2 show simulation results of the two metrics.

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

Table 1. **Failure rate of jobs**

system load	CD=0	CD=0.5	CD=1
0.1	0.00%	0.00%	0.18%
0.2	0.00%	0.00%	2.16%
0.3	0.00%	0.01%	5.09%
0.4	0.00%	0.02%	7.12%
0.5	0.00%	0.05%	8.91%
0.6	0.00%	0.11%	10.27%
0.7	0.00%	0.27%	12.12%

Table 2. **Deadline missing rate of jobs**

system load	CD=0	CD=0.5	CD=1
0.1	0.0000%	0.0000%	0.0000%
0.2	0.0002%	0.0000%	0.0000%
0.3	0.0004%	0.0001%	0.0000%
0.4	0.0022%	0.0003%	0.0000%
0.5	0.0078%	0.0038%	0.0000%
0.6	0.0431%	0.0079%	0.0000%
0.7	0.3631%	0.1290%	0.0000%

From Table 1, we can see that within the tested system load, if providers are aggressive, i.e., CD equals 0, all the jobs submitted by consumers can be executed. If providers are not so aggressive but a little conservative, job failure happens for providers reserve time for unconfirmed jobs and tend to fail to meet the deadlines of jobs coming later. As the results show, if the conservation comes to an extreme, i.e., CD equals 1, failure rate increases greatly when system load gets heavier. When system load is 0.7, the failure rate is above 10%. Table 2 is about deadline missing rate. When CD equals 1, jobs executed never miss their deadlines. As providers get more aggressive and system load is heavier, deadline missing rate increases. However, even when system load is 0.7 and CD equals 0, the deadline missing rate is under 1%. We define successful execution rate with the equation below.

$$successful\_execution\_rate = (1 - f\_r) * (1 - d\_m\_r)$$

f_r : *failure rate*

d_m_r : *deadline missing rate*

With the definition and the two tables above, we get Figure 4. Because deadline missing rate is much smaller than failure rate in most cases, it has less impact on successful execution rate. As Figure 4 shows, the conservative attitude at competing for jobs is not a desirable one when considering the successful execution rate.

For providers, first we study how CD impacts the total penalty, sum of penalty paid by all the providers. Table 3 shows the results.

Apparently, as CD decreases and system load increases, total penalty gets bigger, which is consistent with the observation about deadline missing rate. The more meaningful metric is net profit, and Figure 5 shows the simulation results of it.

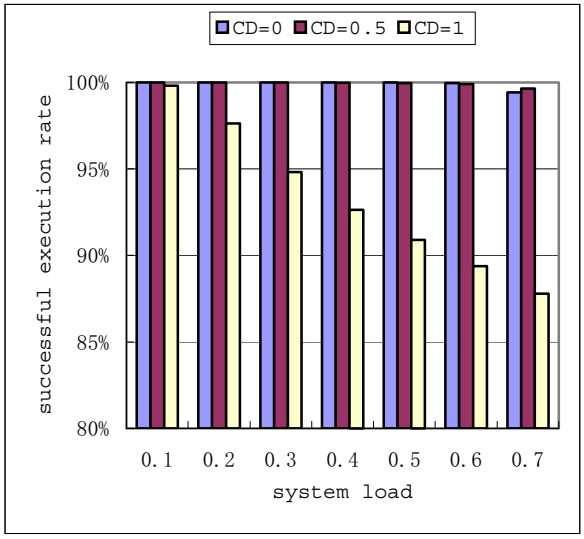

Figure 4. **Successful execution rate of jobs**

Table 3. **Total penalty of providers**

system load	CD=0	CD=0.5	CD=1
0.1	0.00	0.00	0.00
0.2	0.18	0.00	0.00
0.3	2.83	0.97	0.00
0.4	19.09	9.75	0.00
0.5	86.40	50.00	0.00
0.6	625.00	375.08	0.00
0.7	6425.89	2970.00	0.00

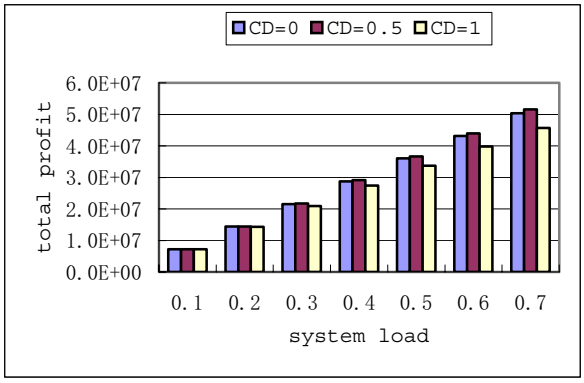

Figure 5. **Total profit of providers**

Conservative providers never need pay penalty, but they lose potential jobs because of the conservative attitude. Thus the total profit is not very satisfactory. The results show that a trade-off attitude, i.e., CD equals 0.5, is superior to the extreme two.

Our performance objective for providers is fair allocation of benefits by which we mean higher capability, more profit. To evaluate it, we first define a metric named *fair-*

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

ness scale. The metric expresses the extent to which a provider fairly benefits. Its definition is as following.

$$for\ provider_i:$$

$$fairness\_scale_i = \frac{profit_i / capability_i}{total\_profit / total\_capability}$$

The ideal case is that fairness scale of every provider is 1. So we hope in simulation results they are as close to 1 as possible. The standard deviation (SD) of fairness scale of all the providers is calculated to evaluate the overall fairness among providers. Figure 6 shows the simulation results.

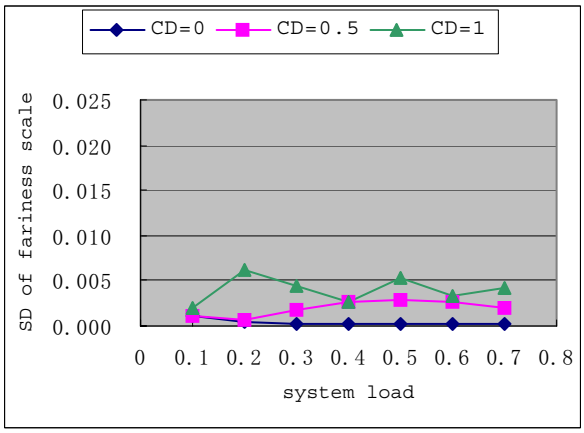

Figure 6. **Standard deviation of fairness scale**

Figure 6 indicates that aggressive competing can achieve the most desirable fairness. Our price adjusting algorithm tries to reasonably allocate job length among all the providers. However, profit is not only related to length of jobs offered but also related to price. Intuitively, when CD equals 0, providers compete aggressively with each other and resources are always sold at a low base price. But in the case that providers are more conservative, price tends to be more diverse. Altogether, within the tested system load, the SD of fairness scale is under 0.01, which proves the effectiveness of our algorithms in guaranteeing the incentive of providers.

6. Conclusion and Future Work

In this paper, we proposed an incentive-based grid scheduling, GridIS, which is composed of a P2P decentralized scheduling framework and incentive-based scheduling algorithms. Our goal is to build a market-like computational grid that every participant has enough incentive to stay and play in it. Since there are two parties in the grid: consumers and providers, the meaning of incentive is two-fold. For consumers, their incentive is ensured by high successful execution rate of jobs. For providers, their incentive is achieved by fair allocation of benefits. We have conducted simulations to assess our incentive-based algorithms. Results show that they are successful in guaranteeing the incentive of the two parties.

We truly believe that the future success of grid computing does require a sound business model. As an attempt to develop such a model, this paper has two major contributions. First, we have defined incentive as scheduling objective for a decentralized market-like computational grid, and the incentive is two-fold, not only for consumers but also for providers. Second, we have studied how the competition attitude of providers influences the performance of scheduling.

Our research in this area is still in its beginning stage and there is much work worthy of further study. Here we list some for consideration. First, we don't take network and hardware failure into account in this study. A failure model may be employed to study the influence. Besides, as it is hard to accurately estimate job length, corresponding mechanisms should be proposed to reduce the possible impact. Finally, the price adjusting algorithm is simple and intuitive. More rational algorithms can be studied. With further work, we envision that a practical market-like computational grid can be successfully built.

Acknowledgements

This work is supported in part by National Natural Science Foundation of China (Grant No. 90412010), China Ministry of Science and Technology (Grant No. 2002AA104310), Chinese Academy of Sciences (Grant No. 20044040), and Hong Kong RGC Grant HKUST6264/04E.

References

[1] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, pp. 37-46, 2002.

[2] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104-111, 1988.

[3] J. Gehring and A. Reinefeld, "MARS - a framework for minimizing the job execution time computing environment," Paderborn Center for Parallel Computing, Technical report, 1995.

[4] F. Berman and R. Wolski, "Scheduling From the Perspective of the Application," *Proceedings of Fifth IEEE Symposium on High performance Distributed Computing HPDC9*, pp. 100-111, 1996.

[5] F. Berman and R. Wolski, "The AppLeS Project: A Status Report," *the 8th NEC Research Symposium*, 1997.

[6] H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *Intl. Journal of Supercomputing Applications and HPC*, vol. 11, 1997.

[7] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "Resource management in Legion," *Workshop on Job Scheduling Strategies for Parallel Processing, in conjunction with the International Parallel and Distributed Processing Symposium*, 1999.

[8] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 59, pp. 107-131, 1999.

[9] H. Chen and M. Maheswaran, "Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems," *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.

[10] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing," *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, pp. 1507-1542, 2002.

[11] S. Shetty, P. Padala, and M. Frank, "A Survey of Maket Based Appraoched in Distributed Computing," Technical Report TR03-13, 2003.

[12] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard, "Enterprise: A market-like task scheduler for distributed computing environments," in *The Ecology of Computation*, B. A. Huberman, Amsterdam: north-Holland, 1988, pp. 177-205.

[13] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. kephart, and S. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions on Software Engineering*, vol. 18, pp. 103-177, 1992.

[14] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A comparison of receiver-initiated and sender-initiated adaptive load sharing," *Performance Evaluation*, vol. 6, pp. 53-68, 1986.

[15] R. Buyya and S. Vazhkudai, "Compute Power Market: Towards a Market-Oriented Grid," *Proceedings of 1st International Symposium on Cluster Computing and the Grid*, p. 574, 2001.

[16] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: an architecture of a resource management and scheduling system in a global computational grid," *HPC Asia 2000*, 2000.

[17] Y. Zhu, L. Xiao, L. M. Ni, and Z. Xu, "Incentive-Based P2P Scheduling in Grid Computing," *Proceedings of Grid and Cooperative Computing*, vol. 3251 / 2004, p. 209, 2004.

[18] The Standard Performance Evaluation Corporation (SPEC), http://www.specbench.org/.

[19] R. Wolski, N. Spring, and J. Hayes, "The network weather service: A distributed resource performance forecasting service for metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, pp. 757-768, 1999.

[20] L. Gong, X. H. Sun, and E. Waston, "Performance modeling and prediction of non-dedicated network computing," *IEEE Trans. on Computer*, vol. 51, pp. 1041-1055, 2002.

[21] S. Xian-He and W. Ming, "GHS: A performance prediction and task scheduling system for Grid computing," *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.

[22] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," *Proceedings of 23rd IEEE International Conference on Distributed Computing Systems*, pp. 500-508, 2003.

[23] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," *Proceedings of ACM SIGCOMM 2003: Conference on Computer Communications*, vol. 33, pp. 407-418, 2003.

[24] Y. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," *Proceedings of ACM SIGMETRICS 2000*, vol. 28, pp. 1-12, 2000.

[25] B. Krishnamurthy and J. Wang, "Topology Modeling via Cluster Graphs," *Proceedings of the First ACM SIGCOMM Internet Measurement Workshop: IMW 2001*, pp. 19-23, 2001.

[26] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "A distributed approach to solving overlay mismatching problem," *Proceedings of 24th International Conference on Distributed Computing Systems*, vol. 24, pp. 132-139, 2004.

[27] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," *Proceedings of ACM SIGCOMM 2001- Applications, Technologies, Architectures, and Protocols for Computers Communications-*, vol. 31, pp. 173-185, 2001.

[28] M. J. Gonzalez, "Deterministic Processor Scheduling," *ACM Computing Surveys*, vol. 9, pp. 173-204, 1997.

[29] J. Nakai, "Pricing Computing Resources: Reading Between the Lines and Beyond," Technical Report NAS-01-010, 2002.