REVENUE MAXIMIZATION IN MARKET-BASED PARALLEL JOB SCHEDULERS

M. ISLAM, G. KHANNA AND P. SADAYAPPAN

Technical Report
Ohio State University (OSU-CISRC-4/08-TR16)

Revenue Maximization in Market-based Parallel Job Schedulers

Mohammad Islam, Gaurav Khanna, P. Sadayappan

Department of Computer Science and Engineering, The Ohio State University

Abstract

This paper addresses the problem of market-based batch scheduling for parallel jobs running on supercomputer centers, with a view to revenue maximization. We adopt a user-centric value based approach, where in, users can specify the maximum price that they are willing to pay and a linearly decaying cost function over time. We propose a new value based scheduling heuristic which is shown to be optimal under restricted versions of the scheduling problem. We evaluate our proposed heuristic with trace-based simulation and compare it with existing heuristics, and show that our heuristic significantly outperforms them in terms of maximizing the revenue, while achieving better performance with respect to standard performance metrics such as slowdown and utilization.

1 Introduction

Recent emerging architectural breakthroughs coupled with the acceptance of parallel concepts in general purpose computing revolutionize the importance of parallel systems in numerous ways. Effective resource management of such huge and demanding systems, as well as satisfying both the resource provider and the user with diversified objectives, is a very challenging problem. Considerable research [12, 18, 17, 20, 22, 15] has already focused on these seemingly orthogonal aspects of parallel job scheduling.

In general, a job scheduler determines when and where to execute a job, given a set of resources and a stream of jobs. In a typical model, when a job arrives in the system, the scheduler tries to allocate the required resources to start the job immediately, if the specified resources are available. Otherwise, the job is queued and scheduled to start at a later time. User satisfaction is often evaluated by response time which is the sum of the waiting time in the job queue (for resources to be available) and the actual runtime after the job starts running. In contrast, a supercomputer center is usually interested in the overall system utilization that determines what fraction of the resources is actually utilized and what fraction remains idle.

While shorter response time and larger utilization are very appealing features from the supercomputing community, overall revenue maximization with the optimal management of resources is the key motivational factor from a resource provider's perspective. Revenue computation primarily requires a suitable charging or cost model. There are two different charging models available in literature. In the provider-centric model, the supercomputer center determines the charge required to execute a job. This widely used model is based on the resources utilized and is usually computed by employing the product, $number\,of\,processors \times runtime$. On the other hand, in the user-centric model, the user, instead of the system provider, offers the price for running a job. According to the adopted model, user specifies the value of a job depending on the importance of timely delivery. Basically, each user defines a piece-wise linear value or utility function whereupon the charge is calculated as function of completion time of a job.

In this paper, we adopt the user-centric approach on the lines of the market-based charging model originally proposed by Culler et. al. [10] and Chase et al [13] for sequential jobs. Both papers proposed various heuristics to maximize the revenue based on expected unit yields and other risk and reward related parameters. However, they do not provide any theoretical backing of the heuristics they employ and their work lacks any impact analysis on widely used performance metrics. In this paper, we propose a new scheduling heuristic which aims at revenue maximization in a online multi-processor system. More significantly, we prove the optimality of our approach in a simplified scenario involving a uni-processor system and an offline batch of jobs. Then, we propose sufficient conditions which when true, guarantee optimality, for an online stream of jobs on a uni-processor system. Finally, we apply our proposed scheduling scheme in a generic multiprocessor system with parallel jobs.

We present the detailed analysis of the schemes with trace-based simulation using different real workloads of jobs from Feitelson's archive [11] . Our results demonstrate that the proposed scheme provides significantly higher revenue as compared to existing schemes while achieving the better performance with respect to standard performance metrics such as slowdown and utilization.

2 Related Work

Several job schedulers such as Portable Batch System (PBS) [6], Moab [4], Load Sharing Facility (LSF) [3], Sun Grid Engine (SGE) [8], etc. have been deployed at shared-resource supercomputer centers to schedule parallel jobs. These schedulers primarily focus on performance by improving utilization, throughput, average turnaround time [12, 9], etc. Even though maximizing revenue is a very desirable feature from a resource provider's perspective, it has received very little attention from both the industry [7, 2, 5] and the research community [16]. In this section, we review some of the existing work pertaining to the concept of revenue in job scheduling.

Typically, the service provider calculates the revenue by using either the provider-selected charging model or the user-specific charging model. Most supercomputer centers [7, 1, 2] adopt the provider-centric approach, in which a user is charged in proportion to its resource usage. In most of the cases, the basic resource is a processor; the charge is proportional to the product of the required number of processors and the run time of a job. Some supercomputer centers [7, 2] that provide multiple queues for different levels of services determine the charges depending on the resource used and quality of services sought. In this model, the charge is generally proportional to ($number\ of\ processors \times runtime \times queuecharge$). In addition, we propose a new charging model for QoS-aware scheduling in our recent work [16]. The proposed charging model is based on the notion that quicker the response time sought, the larger the charge should be. In particular, there are two separate charging components associated with resource usage and QoS guarantees respectively. The resource usage component relies mainly on the amount of resources used and the duration of the service. The QoS component of the charge essentially depends on the flexibility of the requested deadline.

Alternatively, there has been some work done to provide user-centric market-based approaches for revenue-aware scheduling [10, 13, 19]. In the user-centric approach, different users with varied goals and preferences express their desire for service in a common way (e.g., through currency). The most common market-based model follows a auction-based resource allocation mechanism that has three major entities: users or buyers, system providers or sellers and the resources to be sold [23, 21]. A user wants to allocate a processor(s) for a specific duration and is willing to pay a certain value for the execution of the job. The system provider is interested in selling the resources to the user with an intent to maximize its overall profit. The auction process, which is generally proposed by the system provider, considers the $value$ or bid of all contending users and ultimately awards it to the highest bidder.

Wladspurger et. al., [23] proposes a market-based microeconomic approach to batch scheduling. They utilize the auction process to choose the winner from the bids of different users. Stoics et. al. [21] also propose an auction-based microeconomic approach for parallel job scheduling. In this scheme, every user has a savings account where he or she receives funds for buying resources for jobs. Also, the user creates an independent expense account for every job and starts transferring funds from his or her savings to a job's expense account. The rate of this fund transfer determines the priority of the job that ultimately plays a vital role in the auction process.

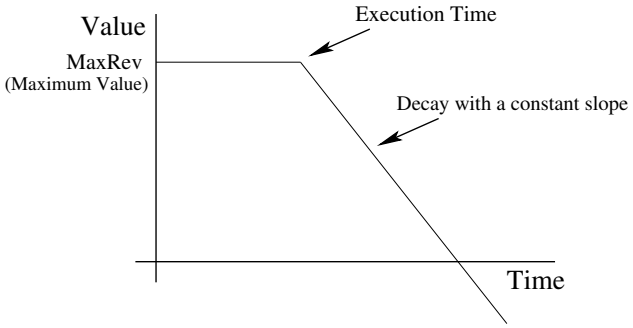


Figure 1. Utility function used in Value-Based Job Scheduling.

Two recent works [10, 13] have also looked at a market-based approach to value-based parallel job scheduling. Both studies rely on a per-job specific utility or value function that provides an explicit mapping of service quality to value. Generally, the value function is a piece-wise linear function that decays as a function of the job completion time. The rate of decay reflects the urgency or sensitivity to delay. The fundamental idea of this model is that the user submits the job with a value function along with other job characteristics. Then, the scheduler decides how to schedule the job using the job information and current state of the system. Culler et. al. [10] adopts user-centric performance metrics instead of system-centric metrics to evaluate the overall system performance. They recommend an aggregate utility function to measure the satisfaction of users with the resource allocation process. For job selection, the proposed scheme implements a heuristic approach where the job with highest value per unit running time gets the preference. Chase et. al. [13] proposes an enhancement to this approach by considering the risk of accepting or rejecting a job due to future uncertainty. Islam at. el. [14] addressed the issue of providing deadline guarantees as well as achieving maximum revenue using a user-centric approach. Their work focused on analyzing the opportunity cost in a QoS-aware scheduler and proposed a history-based predictive technique to estimate the opportunity cost and increase the overall system revenue.

3 Value-based Scheduling

In this section, we examine FirstPrice [10], PresentValue, OpportunityCost, FirstReward approaches [13] pro-

posed in the context of value-based scheduling. The key aspect of the job model is that a user specifies the value of a job depending on the importance of timely delivery. Basically, each user defines a piece-wise linear value or utility function (shown in Figure 1) whereupon the charge is calculated as a function of the completion time of a job. The value function reflects the urgency of the job as a time-dependent function. For simplicity, the value function has two linear pieces. The first part indicates the maximum value that a user is willing to pay if it is completed in its earliest possible time, which in turn expresses the importance of the job. Likewise, the second part denotes decay or down-slope of the value and shows the sensitivity of the job to further delay in the job's completion time. It is expected that users with tight deadline jobs will offer a high initial value and steeper slope for the job. Eventually, the aggregate utility, which is calculated by summing up the individual value earned for all the jobs in the system, can be used to estimate the overall system performance.

Formally, let $S =< j_1, j_2, \ldots, j_n >$ be a set of jobs. Let EST_i be earliest start time of job j_i, P_i be the processing time of job j_i, and $Slope_i$ be the slope of the linearly decreasing value for the job j_i over time. Each job j_i earns a maximum value $MaxRev_i$ if it completes at its minimum completion time $EST_i + P_i$. If the job is delayed, then the value decays linearly at the rate $slope_i$. Since the value starts deteriorating only once the job completion time exceeds the earliest completion time, therefore we focus our analysis on the decreasing portion of the curve. The time varying value $Value_{it}$ in the decreasing portion of the interval is of the form

$$Value_{it} = MaxRev_i - Slope_i * t \tag{1}$$

3.1 FirstPrice.

Jobs are ordered for execution in the decreasing order of their expected yield per unit of resource per unit of processing time. In other words, $FirstPrice$ orders the jobs based on the unit gains they offer.

$$FirstPrice_i = \frac{Value_{it}}{P_i} \tag{2}$$

3.2 PresentValue.

This approach builds upon the idea proposed in the $FirstPrice$ scheduling criterion. It is based on the concept of present value in finance. The key idea is that, if there are two jobs with the same unit gains and slopes, then its preferable to run the shorter job first. This is because a shorter job reduces the risk of delaying a highly urgent or highly valuable job which arrives later. In other words, executing the shorter job first makes the scheduling more risk-aware. The present value of a job PV_i is defined as follows.

$$PV_i = \frac{Value_{it}}{1 + DiscountRate \times P_i} \tag{3}$$

Here, $DiscountRate$ is a parameter which is used to decide the weightage to be given for future gains.

3.3 OpportunityCost.

The $OpportunityCost$ models the loss incurred by a job j_k whose execution is delayed in order to execute another queued job j_i. A job's urgency cost depends upon the urgency of other queued jobs. The opportunity cost OC_i to start a job at some point is equal to the aggregate decline in the yield of all other competing jobs. Formally, it is defined as follows.

$$OC_i = \sum_{\forall k, k \neq i} slope_k \times P_i \tag{4}$$

3.4 FirstReward.

The $FirstReward$ heuristic tries to balance the competing effects of a job's present value as well as its opportunity cost. It does so by introducing a tunable parameter alpha as follows.

$$FirstReward_i = \frac{alpha \times PV_i - (1 - alpha) \times OC_i}{P_i} \tag{5}$$

4 Normalized Urgency

In this section, we present a new approach, namely, $NormalizedUrgency$, to solve the aforesaid problem. The objective function is to maximize the total value earned by the jobs. The problem is how to schedule the jobs so as to maximize the overall value earned. In other words, the goal is to maximize Rev which is defined as follows.

$$Rev = \sum_{i=1}^{i=n} Value_{it} \tag{6}$$

For each job, we define an ordering function $NormalizedUrgency_i$ defined as follows:

$$NormalizedUrgency_i = \frac{slope_i}{P_i} \tag{7}$$

Jobs are ordered for execution in the decreasing order of their $NormalizedUrgency_i$ values. In order to provide an intuition for our approach, we first look at certain restricted versions of the general problem and propose optimal solutions or sufficient conditions to guarantee optimality for them.

4.1 Batch of jobs: Offline scenario

In this section, we look at the following restricted version of the problem. All jobs are associated with the same earliest start times. In other words, we are focusing on an offline version of the problem where all jobs are available to be executed at time t=0. *The problem is how to schedule these jobs on a uniprocessor system to achieve our goal function as mentioned in (Eq. 6).*

Theorem 1 *An optimal sequence can be obtained by ordering the jobs in non-increasing order of their respective* $NormalizedUrgency$ *values.*

Proof Consider two jobs j_i and j_k such that the optimal sequence is to execute job j_i before job j_k and let us consider that $NormalizedUrgency_i < NormalizedUrgency_k$. The total revenue attributed to these jobs, Rev_{ik} is:

$$Rev_{ik} = MaxRev_i - slope_i * P_i + MaxRev_k - slope_k * (P_i + P_k) \quad (8)$$

Now swap the two jobs so that job j_k now executes before job j_i. The revenue attributed to the two jobs in this case, $Rev(ki)$ is:

$$Rev_{ki} = MaxRev_k - slope_k * P_k + MaxRev_i - slope_i * (P_k + P_i) \quad (9)$$

The difference in the overall revenue earned is:

$$Rev_{ki} - Rev_{ik} = slope_k * P_i - slope_i * P_k \quad (10)$$

The right hand side in (Eq. 10) is positive since $NormalizedUrgency_i < NormalizedUrgency_k$. Therefore, swapping the two jobs to obey a non-increasing order of the $NormalizedUrgency$ values leads to an increase in the overall revenue earned. Therefore, the former solution is not optimal. A sequence of such interchanges yields a sequence which satisfies the order specified in the claim and is optimal.

4.2 A dynamically arriving stream of jobs (with same slope): Online scenario

In this section, we look at a more relaxed version of the problem discussed in section 4.1. All jobs are associated with the possibly different earliest start times EST. In other words, we are focusing on an online version of the problem where all jobs arrive over time. However, we assume the knowledge of earliest start times for jobs arriving in future. The optimal solution for such a problem can act as an upper bound on the total revenue earned in a truly online scenario where jobs arrive over time and the scheduler does not have any knowledge of the arrival times of future arriving jobs. For simplicity sake, we assume that all the jobs have the same decay rate $slope$.

The earliest completion times ECT_i of each job is calculated dynamically based on the previously scheduled jobs. In the initial state, when no job has been scheduled, the earliest completion time ECT_i of each job is defined as follows:

$$ECT_i = EST_i + P_i \quad (11)$$

Theorem 2 *Two conditions, which when true together, guarantee optimality. Condition 1 states that jobs should be scheduled in a non-decreasing order of their earliest start times. Condition 2 says that the jobs should be scheduled in a non-decreasing order of earliest completion times. If a schedule obeys both these conditions, it is guaranteed to give an optimal solution.*

Proof We first prove this in the context of a two job scenario where in there are only two competing jobs in the system.

Consider two jobs j_i and j_k. Without loss of generality, Let us consider that $ECT_i < ECT_k$. There are three possible scenarios.

1. **Scenario 1:** $EST_i < EST_k < EST_i + P_i < EST_k + P_k$

   If job j_i executes before job j_k, The total revenue earned , Rev_{ik} is computed as follows:

   $$Rev_{ik} = MaxRev_i - slope * (EST_i + P_i) + MaxRev_k - slope(ECT_i + P_k) \quad (12)$$

   If job j_k executes before job j_i, The revenue P_{ki} earned is:

   $$Rev_{ki} = MaxRev_k - slope * (EST_k + P_k) + MaxRev_i - slope * (ECT_k + P_i) \quad (13)$$

   The difference in the overall revenue earned is:

   $$Rev_{ki} - Rev_{ik} = slope * (EST_i + ECT_i - EST_k - ECT_k) \quad (14)$$

   In this scenario, $EST_i < EST_k < ECT_i < ECT_k$, therefore the right hand side in (Eq. 14) is negative. Therefore, the overall value decreases if job j_k executes before job j_i.

2. **Scenario 2:** $EST_k < EST_i < EST_i + P_i < EST_k + P_k$

   If job j_i executes before job j_k, The revenue P_{ik} earned is:

$$Rev_{ik} = MaxRev_i - slope * (EST_i + P_i) + MaxRev_k - slope(ECT_i + P_k) \quad (15)$$

   If job j_k executes before job j_i, The revenue P_{ki} earned is:

$$Rev_{ki} = MaxRev_k - slope * (EST_k + P_k) + MaxRev_i - slope * (ECT_k + P_i) \quad (16)$$

   The difference in the overall revenue earned is:

$$Rev_{ki} - Rev_{ik} = slope * (EST_i + ECT_i - EST_k - ECT_k) \quad (17)$$

   In this scenario, two conditions are sufficient to make the right hand side of the (Eq. 17) negative. The two conditions are: $EST_i < EST_k$ and $ECT_i < ECT_k$. If these two conditions are true, then the order ik is certainly better than the order ki. However, the condition $EST_i < EST_k$ violates the assumptions of Scenario 2 that is $EST_k < EST_i$.

3. **Scenario 3:** $EST_i < EST_i + P_i < EST_k < EST_k + P_k$

   In this scenario, it is quite obvious that job j_i should be executed before job j_k. This is also reflected by the fact that both the two sufficient conditions for the ordering ik to be better than the ordering ki: $EST_i < EST_k$ and $ECT_i < ECT_k$ are true in this scenario.

   From these three possible scenarios, we can safely conclude that there are two conditions which are sufficient to guarantee optimality. The conditions are $EST_i < EST_k$ and $ECT_i < ECT_k$. In other words, a sequence which satisfies a non-decreasing order of earliest start times as well as the earliest completion times is optimal for a 2 job scenario.

4.3 A dynamically arriving stream of jobs (different slopes): Online scenario

In this section, we look at an extended version of the problem discussed in Section 4.2. All jobs are associated with the possibly different earliest start times EST_i as in Section 4.2. However, now we relax the assumption that all jobs have same decay rates. Each job j_i is associated with a decay rate $slope_i$.

Theorem 3 *Two conditions, which when true together, guarantee optimality. Condition 1 states that jobs should be scheduled in a non-decreasing order of their respective $EST \times slope$ values . Condition 2 says that the jobs should be scheduled in a non-decreasing order of their respective $\frac{ECT}{slope}$ ratios. If a schedule obeys both these conditions, it is guaranteed to give an optimal solution.*

Consider two jobs j_i and j_k. Let us consider that $ECT_i < ECT_k$. There are three possible scenarios, of which we look at one in this section.

$EST_i < EST_k < EST_i + P_i < EST_k + P_k$

If job j_i executes before job j_k, The component of the total revenue earned which is attributed to these two jobs, Rev_{ik} is computed as follows:

$$Rev_{ik} = MaxRev_i - slope_i * (EST_i + P_i) + MaxRev_k - slope_k(ECT_i + P_k) \quad (18)$$

If job j_k executes before job j_i, The revenue earned is:

$$Rev_{ki} = MaxRev_k - slope_k * (EST_k + P_k) + MaxRev_i - slope_i * (ECT_k + P_i) \quad (19)$$

The difference in the overall revenue earned is:

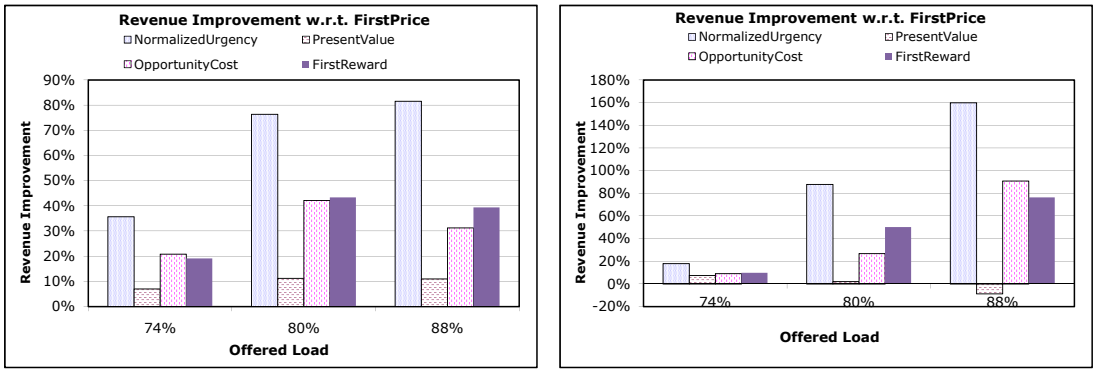
$$Rev_{ki} - Rev_{ik} = slope_i * EST_i - slope_k * EST_k + slope_k * ECT_i - slope_i * ECT_k \quad (20)$$

In order for the right hand side in Equation 20 to be negative, there are two sufficient conditions: $slope_i * EST_i < slope_k * EST_k$ and $slope_k * ECT_i < slope_i * ECT_k$. Therefore, we again have two possibly conflicting conditions. Ordering the jobs according to one condition may violate the another.

5 Experimental Results

In this section, we experimentally compare the performance of our proposed scheme $NormalizedUrgency$ against the existing schemes such as $First\,Price$ [10], $Present\,Value$, $Opportunity\,Cost$ and $First\,Reward$ [13]. We employ an event-based simulator that essentially takes data in the standard workload format version 2.0 [11], simulates the scheduling model and creates an output trace containing data necessary to gather metrics and perform post processing.

We primarily compare the overall revenue gained from different schemes. In addition, other performance metrics such as slowdown, response time and system utilization are also evaluated to analyze the correlated impact of improving the revenue. Moreover, since the parallel system with dynamically arriving parallel jobs could produce varied scenarios, we further study the robustness of the schemes by



(a) 50% Urgency Job

(b) 20% Urgency Job

Figure 2. Revenue improvement relative to FirstPrice assuming exact runtime estimate at different offered loads

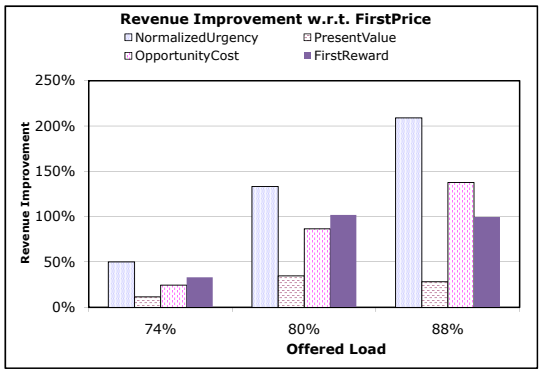


Figure 3. Revenue improvement relative to FirstPrice assuming exact runtime estimate at different offered loads with 80% urgent job

varying the offered loads and also looking at inexact user estimates.

5.1 Evaluation Approach

The strategies evaluated in this paper are simulated using real workload traces, such as those available at the Parallel Workload Archive [11]. These traces include information such as the jobs runtime, the number of nodes each job used, the submission time, and a user estimated runtime limit (wall-clock limit). In this work, we used two such real workload trace of 10,000 jobs subset of the SDSC SP-2 and CTC SP-2 workload trace.

Runtime Estimates: Runtime estimates are critical when evaluating parallel job schedulers. Therefore, two sets of simulations are performed. The first set of simulations takes an idealistic view of the traces and assumes that users are able to perfectly estimate their job's runtime; this allows us to concentrate on the capabilities of our algorithms without being affected by other noise in the traces. The second set of simulations uses the actual runtime estimates given in the workload trace; this allows us to evaluate our algorithms in more realistic environments.

Job Submission Load: As the demand for resources is increasing, we need to investigate the effectiveness of our scheme in various high load environments. This necessitates the expansion of the real traces in a rational way. We mainly use the duplication approach for varying the load on the supercomputer center (number of jobs submitted). Jobs are selected randomly and duplicated with the same arrival time keeping other attributes value unchanged. For example, we start with a trace and call this the base trace (load = 1.0). To generate a new trace with load = 1.2, we randomly pick 20% of the jobs in the base trace and introduce extra duplicate jobs at the same points in the trace. We also pay special attention to maintain the subset relationship of jobs

across the different loads. For example, the jobs selected for load 1.2 are also selected for load 1.3 with additional 10% new jobs. In summary, for CTC trace, we start with base offer load of 74% and increased the load to 80% and 88%. For SDSC trace, the offered loads are 82% (base trace), 90% and 104%.

Urgency and Job Cost: None of the available workload traces contains any information about urgency requirements for the job or the amount the user is willing to pay for the job, an aspect which is essential to our model. In our model, we assume that each job specifies a maximum cost the user is willing to pay for the job, and a linearly decaying cost function (as shown in figure 1). Hence, for our evaluations, we randomly select a fraction *U* of the jobs as *urgent*. The cost of non-urgent jobs is fixed at 0.1 units per processor-second of the job. The cost of urgent jobs is set to be higher than that of non-urgent jobs by a factor *C*. In our experiments, we used values of 20%, 50% and 80% for *U* and value of 100 for *C*.

5.2 Revenue Improvement for Different Schemes

Revenue is an important metric to Supercomputer center to evaluate the performance of the center. In this section, we measure the revenue achieved using each scheme for the same workload. Overall revenue of a system of n jobs can be estimated using the Eq. 6 in the user-centric model (Figure 1).

Offered Load	20% Urgent Job	50% Urgent Job
59%	7.5	7.8
65%	12.7	13.9
72%	35.9	19.4
78%	40.2	42.7

Table 1. Percentage Improvement of $NormalizedUrgency$ over $FirstReward$ for sequential job traces.

Although our main focus is parallel jobs in a multiprocessor system, we initially study the schemes in the context of only sequential jobs . In Table 1, we present the revenue improvement of our proposed scheme $NormalizedUrgency$ with respect to the best previously proposed scheme $FirstReward$. For this experiment, the sequential job trace is synthetically generated using the utility from [11]. The results are presented for various offered loads and for two different job-mixes. The data displays that the new scheme $NormalizedUrgency$ earns as high as 40% more revenue than $FirstReward$. More detailed results are shown for different schemes in Table 2 and Table 3.

Figure 2 shows the revenue improvements for different schemes with respect to $FirstPrice$ [10] for a multiprocessor system. Figure 2(a) displays the revenue where urgent and non-urgent jobs are equal in load and Figure 2(b) shows the revenue where the number of urgent jobs are fewer (20% urgent jobs). In both graphs, we vary the system load from the actual load (74%) of CTC trace to higher loads upto 88%. The graphs clearly demonstrate that our proposed scheme $NormalizedUrgency$ achieves the highest revenue among all the schemes (nearly 45% improvement over the second best scheme). This revenue improvement is expected because the $NormalizedUrgency$, unlike other schemes, is based on the job's urgency relative to its length. It does not look at the maximum revenue offered by a job. Our optimality proofs and sufficiency criteria for the restricted versions of the problem validate the idea of normalized urgency. In addition, we observe that the revenue improvements are much higher at high load as compared to the original load (74%) scenario. This is because, though the job mix is the same as the original load case, the absolute number of urgent jobs is higher in the high-load case allowing more opportunity to start more high revenue jobs earlier. The results for traces with mostly urgency jobs (80%) also exhibits the same trends as shown in Figure 3.

Offered Load	First Price	Normalized Urgency	Present Value	Opportunity Cost
59%	-79	7	-81	-11
65%	-140	13	-139	-20
72%	-314	36	-309	-24
78%	-452	40	-445	-53

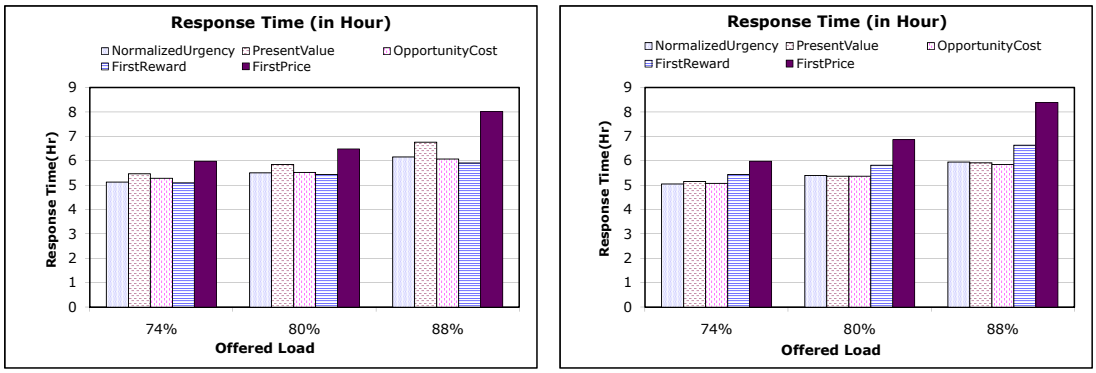
Table 2. Percentage Revenue Improvement of different schemes over $FirstReward$ for sequential jobs with 20% urgent job

Offered Load	First Price	Normalized Urgency	Present Value	Opportunity Cost
59%	-89	8	-87	-11
65%	-137	14	-136	-20
72%	-280	19	-276	-39
78%	-476	43	-469	-56

Table 3. Percentage Revenue Improvement of different schemes over $FirstReward$ for sequential jobs with 50% urgent job

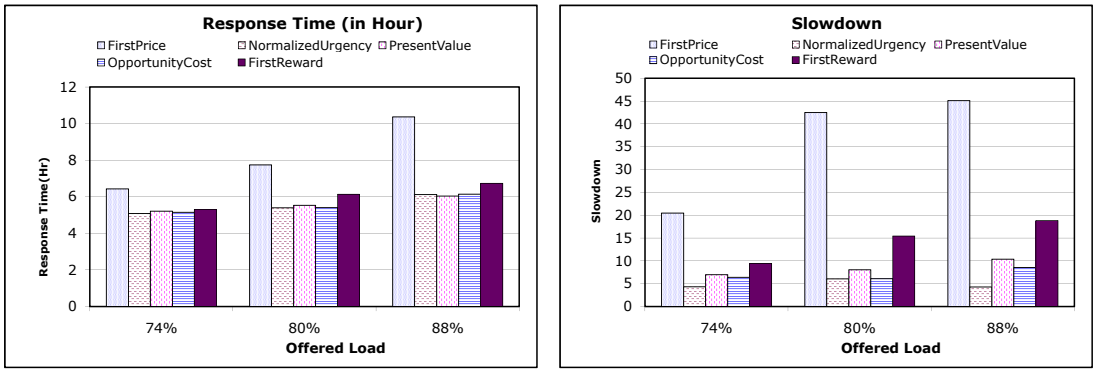
5.3 Impact on System Performance

Although revenue maximization is an important aspect for a supercomputer center, it is significant to study the impact on other widely used performance metrics such as slowdown, response time and utilization. In this section, we study the response time and slowdown metrics for various schemes at different load. We did not include the utilization



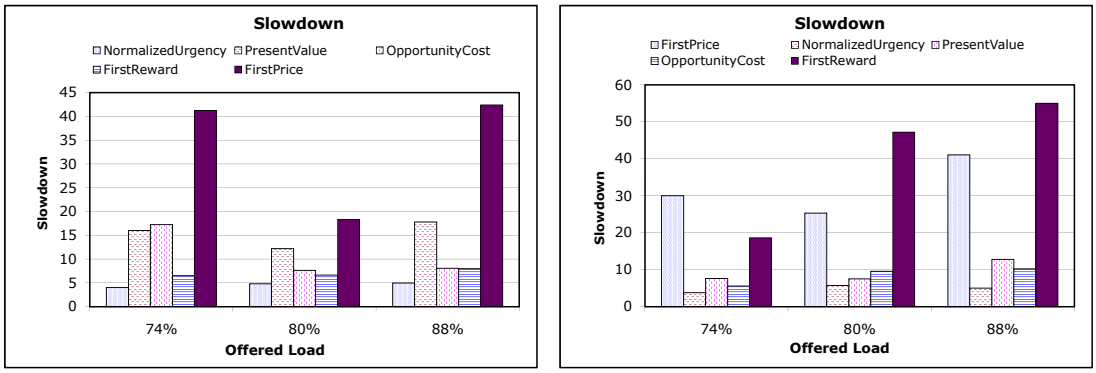
(a) 50% Urgency Job (b) 20% Urgency Job

Figure 4. Response time for different schemes assuming exact runtime estimate at different offered loads



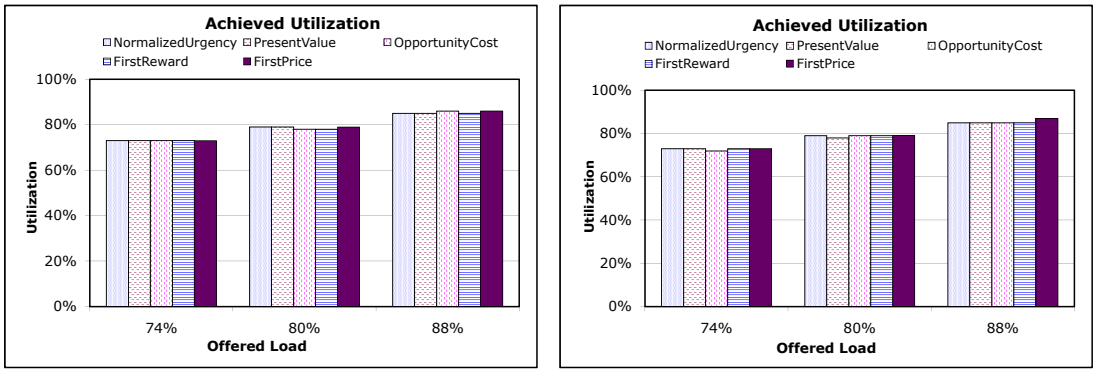
(a) Response Time (b) Slowdown

Figure 5. Response time and Slowdown for different schemes assuming exact runtime estimate at different offered loads with 80% urgent job



(a) 50% Urgency Job (b) 20% Urgency Job

Figure 6. Slowdown for different schemes assuming exact runtime estimate at different offered loads



(a) 50% Urgency Job

(b) 20% Urgency Job

Figure 7. Achieved utilization for different schemes assuming exact runtime estimate at different offered loads

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	137.8	61.28	91.47	267.16
10m-1hr	3.09	7.02	17.37	36.36
1hr-8hr	1.8	1.98	4.99	5.59
> 8hr	1.17	1.29	1.84	2.04

Table 4. Slowdown (First Price) for offered load 88%

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	0.96	1.63	4.13	15.91
10m-1hr	1.80	2.80	6.14	12.87
1hr-8hr	6.79	6.44	12.40	16.06
> 8hr	15.72	17.68	22.68	22.95

Table 5. Response Time in hour (First Price) for offered load 88%

results due to the space constraints.

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	14.47	23.85	55.92	234.58
10m-1hr	1.42	2.7	8.13	35.54
1hr-8hr	1.27	1.45	3.36	4.78
> 8hr	1.11	1.15	1.92	2.2

Table 6. Slowdown (Present Value) for offered load 88%.

Response Time: The response time of a job is the sum of the time for which it has to wait in the job queue (for resources to be available) and the actual runtime after the job starts running. Figure 4 shows the graphs for response time as the load varies for two different job mixes. The graphs demonstrate that $FirstPrice$ performs worst whereas the proposed $NormalizedUrgency$ performs the best or comparable to the second best. Figure 5(a) displays the response time where most of the jobs are urgent (80%).

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	0.28	0.67	2.51	14.27
10m-1hr	0.89	1.18	2.93	12.28
1hr-8hr	5.23	5.01	9.44	15.17
> 8hr	15.04	15.99	23.73	25.50

Table 7. Response Time in hour (Present Value) for offered load 88%.

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	3.76	6.99	17.33	30.86
10m-1hr	1.39	1.92	5.64	8.2
1hr-8hr	1.26	1.38	3.16	3.44
> 8hr	1.14	1.14	2.19	2.59

Table 8. Slowdown (Normalized Urgency) for offered load 88%.

Slowdown: Slowdown of a job measures how much slower the system appears to the user compared to a dedicated machine. It is calculated as the ratio of the response time to the runtime of a job. Figure 6 shows the average slowdown for various schemes at different loads. In addition, Figure 5(b) displays the slowdown where most of the jobs are urgent (80%). The graphs demonstrate that our proposed scheme $NormalizedUrgency$ significantly outper-

	1 Proc	2-8 Procs	9-32 Procs	>32 Procs
0-10min	0.09	0.24	0.81	1.95
10m-1hr	0.87	0.98	2.11	3.06
1hr-8hr	5.21	4.89	9.70	11.69
> 8hr	15.56	15.78	27.16	29.74

Table 9. Response Time in hour (Normalized Urgency) for offered load 88%.

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	4.83	9.83	28.31	85.18
10m-1hr	1.42	2.05	6.53	14.21
1hr-8hr	1.23	1.33	3.55	5.79
> 8hr	1.07	1.11	1.73	2.5

Table 10. Slowdown (Opportunity Cost) for offered load 88%.

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	0.12	0.30	1.37	5.89
10m-1hr	0.89	0.95	2.32	4.89
1hr-8hr	4.96	4.66	9.83	16.58
> 8hr	14.48	15.35	21.42	28.57

Table 11. Response Time in hour (Opportunity Cost) for offered load 88%.

forms the other schemes in terms of slowdown. In general, the shorter jobs contribute most to the average slowdown metric. In other words, the scheme, which prefers the short jobs to long jobs, exhibits a lower slowdown value. Since the proposed scheme also favors the short jobs, the improvement is anticipated. This hypothesis is further supported by the size-wise results in the table 4 to table 13. We categorize the jobs into sixteen categories based on the runtime and number of processors requested. The tables present the slowdown and response time for each category of job for various schemes. The first row of table 8 and 12 show that $NormalizedUrgency$ schemes serves the short jobs (runtime between 0 to 10 minutes) better as compared to $FirstReward$, resulting in a lower slowdown. However, the slowdown value for longer jobs are similar for both the schemes, as expected.

Utilization : Utilization is the ratio of resources used by the jobs to the resources offered. The utilization metric is very important to the supercomputer centers. Figure 7 shows the achieved utilization for different schemes. The achieved utilization for the schemes are nearly identical, and are a function of the scheduler's ability to tightly pack the submitted jobs in the 2D chart.

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	5.82	9.73	29.64	72.57
10m-1hr	1.39	2.24	6.88	16.17
1hr-8hr	1.21	1.24	3.33	3.97
> 8hr	1.06	1.11	1.7	2.54

Table 12. Slowdown (First Reward) for offered load 88%.

	1 Proc	2-8 Procs	9-32 Procs	> 32 Procs
0-10min	0.14	0.31	1.39	4.77
10m-1hr	0.87	1.03	2.52	5.76
1hr-8hr	4.82	4.51	9.27	12.42
> 8hr	14.39	15.35	20.74	28.96

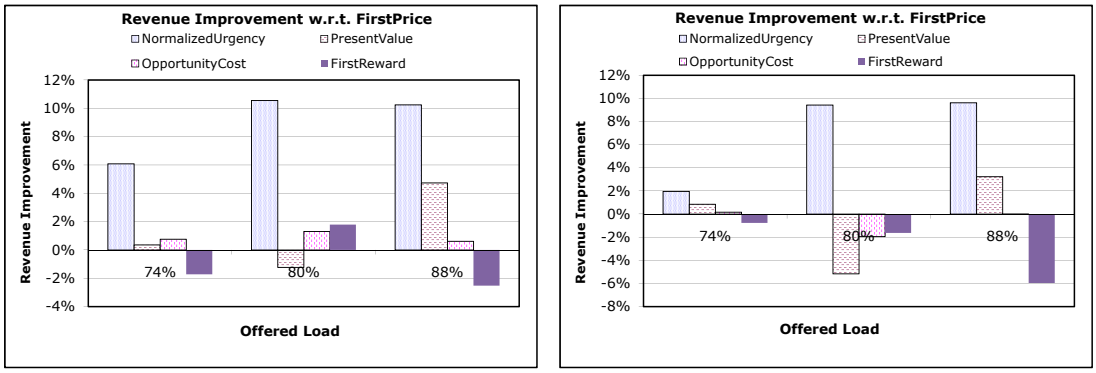
Table 13. Response Time in hour (First Reward) for offered load 88%.

5.4 Impact of User Runtime Inaccuracy

In our previous experiments, we assumed that the user accurately estimated runtime of a job at submission. The assumption was made to reduce the number of variables and thereby aid the process of understanding the behavior of schemes in a more predictable scenario. However, the user runtime estimates are inherently inaccurate. Most of the time, the user over-estimates the runtime of a job. Therefore, in this section, we investigate the impact of such inaccuracy in user runtime estimation for different schemes.

Figure 8 shows the revenue improvement in a real setting where the inaccurate estimated runtime from an actual trace is utilized in scheduling decision. The results show similar trends to the ones shown in Figure 2 where exact runtime estimation was assumed. However, the absolute revenue improvements are lower in all cases (compared to Figure 2) and this disparity is highest in the case where we employ the lowest percentage of urgent jobs (Figure 8(b)). The main reason for this is the over-estimation of a job's runtime and the usage of overestimated runtime in defining value function (Figure 1). As previously described, our value function has two region: a flat region where revenue is constant and a sloped region where the revenue drops as the time increases. Since the runtime is usually over-estimated, the flat region becomes longer and most of the jobs get completed within the flat region resulting in no differences in revenue. Figure 9 displays the same overall trend where most of the jobs (80%) are urgent. In summary, although the absolute revenue gains of the schemes are not very good, our proposed scheme still outperforms all other schemes at all loads and at all job mixes exhibiting the improved adaptability of our scheme.

So far, in our experiments, we used the CTC trace as the base trace. To demonstrate the robustness of our schemes



(a) 50% Urgency Job (b) 20% Urgency Job

Figure 8. Revenue improvement realtive to FirstPrice assuming inexact runtime estimate at different offered loads

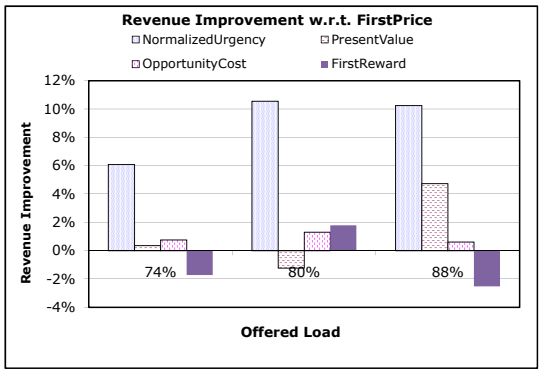


Figure 9. Revenue improvement realtive to FirstPrice assuming inexact runtime estimate at different offered loads with 80% urgent job

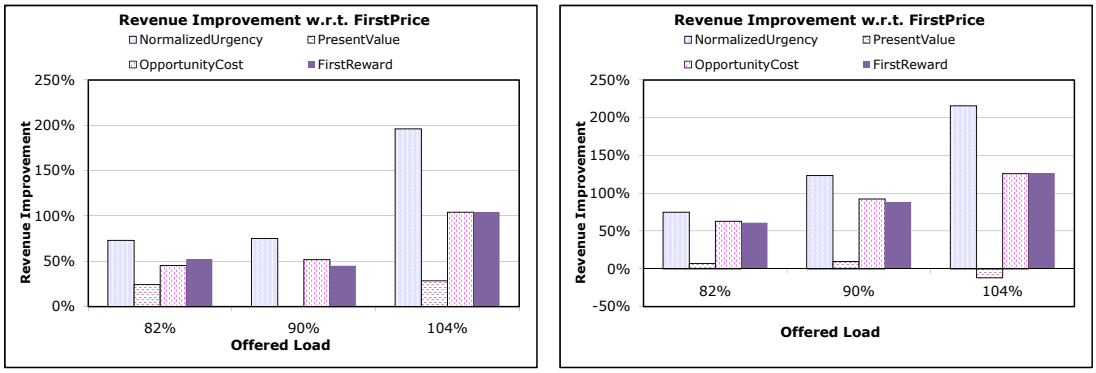
across different workloads, we illustrate the revenue improvement using the SDSC workload. Figure 10 shows the revenue improvement for various loads and two different job mixes where user estimated runtime is actual but inaccurate. Although the trends are consistent with the CTC workload, the improvement is far better for SDSC workload. This is attributed to high original offered load (82%) for SDSC compared to low original offered load (74%) in CTC. At high load, there are more urgent jobs which the schemes can leverage to improve the overall revenue.

6 Conclusions

In this paper, we explored the problem of market-based batch scheduling for parallel jobs running on supercomputer centers, with a view to maximize the overall revenue earned by the resource provider. We proposed a new scheduling heuristic called $NormalizedUrgency$ which is based on the notion of prioritizing jobs based on their urgency and their processing times. We prove the optimality of our approach for certain restricted versions of the problem. Finally, we experimentally compare our schemes against the existing schemes like $FirstPrice$, $FirstReward$ etc. using real supercomputer center workloads. Our results demonstrate that the proposed scheme leads to significantly higher revenue as compared to existing schemes while achieving better performance with respect to standard performance metrics such as slowdown and utilization.

References

[1] Charging for Academic use in OSC. http://www.osc.edu/hpc/software/general.shtml#charging.



(a) 50% Urgency Job

(b) 20% Urgency Job

Figure 10. Revenue improvement realtive to FirstPrice assuming inexact runtime estimate at different offered loads in SDSC trace

[2] Internal Charging Facilities for Moab. http://www.clusterresources.com/products/mwm/docs/6.5internalcharging.shtml.

[3] Load Sharing Facility(LSF). http://www.platform.com/productsv/LSF/.

[4] Moab. http://www.supercluster.org/moab.

[5] National Energy Research Scientific Computing Center. http://www.nersc.gov/nusers/accounts/charging/sp-charging.php.

[6] OpenPBS. http://openpbs.org.

[7] Queues and charging for resource usage on bluevista. http://www.cisl.ucar.edu/computers/bluevista/queue.charge.html.

[8] Sun Grid Engine. http://gridengine.sunsource.net/.

[9] S. H. Chiang and M. K. Vernon. Production Job Scheduling for Parallel Shared Memory Systems. In *IEEE International Parallel and Distributed Processing Symposium*, 2001.

[10] B. Chun and D. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers . In *IEEE International Symposium on Cluster Computing and the Grid*, 2002.

[11] D. G. Feitelson. Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/.

[12] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer Verlag, 1997.

[13] D. Irwin, L. Grit, and J. Chase. Balancing risk and reward in a market-based task service. In *IEEE International Symposium on High-Performance Distributed Computing*, 2004.

[14] M. Islam, P. Balaji, G. Sabin, P. Sadayappan, and D. K. Panda. Analyzing and Minimizing the Impact of Opportunity Cost in QoS-aware Job Scheduling. In *International Conference on Parallel Processing*, Oct 2007.

[15] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for Parallel Job Scheduling. In *Workshops on Job Scheduling Strategies for Parallel Processing*, 2003.

[16] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. Towards Provision of Quality of Service Guarantees in Job Scheduling. In *IEEE International Conference on Cluster Computing*, 2004.

[17] P. Keleher, D. Zotkin, and D. Perkovic. Attacking the Bottlenecks in Backfilling Schedulers. In *Cluster Computing: The Journal of Networks, Software Tools and Applications*, March 2000.

[18] A. W. Mualem and D. G. Feitelson. Utilization, Predictability, Workloads and User Estimated Runtime Esti mates in Scheduling the IBM SP2 with Backfilling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 12, pages 529–543, 2001.

[19] F. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Proceedings of Supercomputing*, 2005.

[20] G. Sabin and P. Sadayappan. Analysis of unfairness metrics for space sharing parallel job schedulers. In *In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing. Springer-Verlag*, 2005.

[21] I. Stoica, H. Abdel-Wahab, and A. Pothen. A Microeconomic Scheduler for Parallel Computers. In *Proceedings of the IPPS '95 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–135, April 1995.

[22] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP2 scheduler using Slack Based Backfilling. In *International Parallel Processing Symposium*, pages 513–517, April 1997.

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