Balancing Risk and Reward in a Market-based Task Service

David E. Irwin, Laura E. Grit, and Jeffrey S. Chase*
Department of Computer Science
Duke University
Box 90129, Durham, NC 27708, U.S.A.
{irwin,grit,chase}@cs.duke.edu

Abstract

This paper investigates the question of scheduling tasks according to a user-centric value metric—called yield or utility. User value is an attractive basis for allocating shared computing resources, and is fundamental to economic approaches to resource management in linked clusters or grids. Even so, commonly used batch schedulers do not yet support value-based scheduling, and there has been little study of its use in a market-based grid setting. In part this is because scheduling to maximize time-varying value is a difficult problem where even simple formulations are intractable.

We present improved heuristics for value-based task scheduling using a simple but rich formulation of value, in which a task's yield decays linearly with its waiting time. We also show the role of value-based scheduling heuristics in a framework for market-based bidding and admission control, in which clients negotiate for task services from multiple grid sites. Our approach follows an investment metaphor: the heuristics balance the risk of future costs against the potential for gains in accepting and scheduling tasks. In particular, we show the importance of opportunity cost, and the impact of risk due to uncertainty in the future job mix.

1 Introduction

The uses of grid computing continue to expand from the core goal of harnessing the power of distributed resources for large-scale computation. Grids are increasingly shared among large numbers of different user groups. Linking clusters together in grids can improve resource efficiency; consolidating small private clusters into cluster utilities can reduce management cost and bring more compute power to each user on demand. Although users benefit from resource sharing, these benefits come at the price of yielding some local control over the resources. It is therefore essential for grids to arbitrate resource usage across competing demands in a way that balances the collective need for effective global use of the resources with each user's need for fairness, predictable performance, and control over the relative priority of their jobs. Economic or market-based approaches are attractive for this purpose.

Today's batch systems address this need with a combination of approaches including user priority, weighted proportional sharing, and service level agreements that set upper and lower bounds on the resources available to each user or group [17, 21]. These scheduling mechanisms have largely been translated, with few adaptations, for use on the grid. Market-based approaches refine this capability by enabling users to expose the relative urgency or cost of their tasks, subject to constraints on their resource usage. In value-based [3, 8] or *user-centric* [10] scheduling, users assign value (also called yield or utility) to their jobs, and the system schedules to maximize aggregate value rather than to meet deadline constraints or a system-wide performance objective such as throughput or mean response time. In economic terms, value corresponds to currency: users "bid" for resources and pay for them according to their user value; the system sells resources to the highest bidder in order to maximize its profit. This approach is a foundation for market-based resource management in federated environments.

This paper focuses on market-based scheduling in grid service sites based on user bids that specify value across a range of service quality levels. Representations of task value create a means to dynamically adjust relative job priority according to the workflow of the organization, e.g., giving jobs higher priority when other activities depend on their completion. For example, the results of a five-hour batch job that is submitted six hours before a deadline are worthless in seven hours. The scheduler must balance the length of a task with both its present value and

*This work is supported by the U.S. National Science Foundation (EIA-9972879, EIA-9870728, ANI-0330658), by IBM, and by HP Labs. Laura Grit is supported by a National Physical Science Consortium (NPSC) Fellowship.

its opportunity cost; in short, it must balance the risk of deferring a task with the reward of scheduling it.

This paper extends the treatment of task scheduling for linearly decaying value functions in the Millennium economic cluster manager at Berkeley [9, 10]. From that starting point, we show how a task service site can schedule a stream of arriving tasks to account for important risk and reward factors. Next, we show how to use value-based scheduling in a framework for market-based bidding and admission control, in which clients negotiate for task services from multiple grid sites. We present a risk/reward heuristic that extends classical scheduling heuristics to a computational economy.

This paper is organized as follows. Section 2 gives an overview of the premises and goals of our work. Section 3 discusses the formulation of value functions, and Section 4 summarizes the resulting scheduling problem and our evaluation methodology. Section 5 develops a configurable scheduling heuristic to balance risk and reward, and Section 6 applies the heuristic to task acceptance and contract negotiation. Section 7 discusses related work, and Section 8 concludes.

2 Overview

We consider a *service market* in which each site sells the service of executing tasks, rather than raw resources. In a service market, clients and servers negotiate contracts that incorporate some measure of service quality and assurance as well as price. For example, clients may pay more for better service, and servers may incur a penalty if they fail to honor the commitments for service quality negotiated in their contracts. This paper considers the mechanisms to assign value to the service, and the policies to negotiate service prices and service quality levels. We focus on the simplest type of service: a batch task service. Our approach is based on three key premises:

- Tasks are batch jobs that consume resources but deliver no value until they complete.
- A job submission specifies a resource request (service demand) to run the job, and correctly specifies the job's duration if its resource request is met.
- Each task is associated with a user-specified *value function* (utility function) that gives the task's value to the user as a function of its completion time. Section 3 discusses these value functions, which are used to specify bids and service contracts.

Figure 1 illustrates the context for our work. Each server site selects buyers for resources based on client bids for submitted tasks, which are given as value functions. If the server agrees to accept a client's bid, it proposes an expected completion time and price derived from

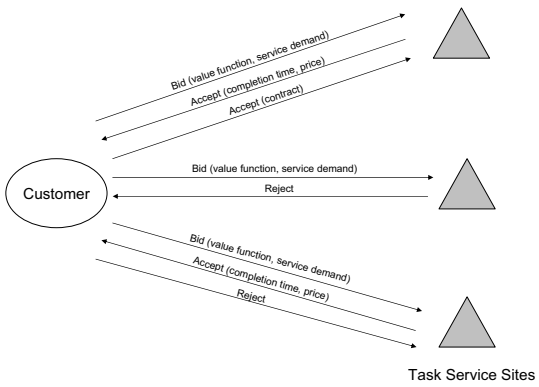

Figure 1. A client or broker negotiating with three task service sites to determine where to run a task.

the value function. The client then selects a server to run the task by choosing among the set of grid sites that responded to its bid. Since the bid specifies value across a range of service quality levels (completion times), the negotiation protocol between the buyer and the seller may consist of just this one pair of exchanges. A broker could coordinate this negotiation process, as in Mariposa [20].

Once the customer and the site agree on the expected completion time and value, a contract is formed. If the site delays the task beyond the negotiated completion time, then the value function associated with the contract determines the reduced price or penalty. As a result, server sites must consider the risks inherent in each contract and scheduling choice.

Our work focuses on the policy choices made by grid sites to maximize their gains in a computational economy. The contribution of this work is a framework and heuristic for task bidding, negotiation, acceptance, and scheduling at a grid site. We show how each site can use the value measures to balance risk and reward in bidding for tasks and scheduling tasks accepted into its task mix. Other aspects of market dynamics—such as pricing systems, incentive mechanism design, and client bidding strategies—are beyond the scope of this paper. In particular:

- We propose a means to negotiate and formulate contracts, but we do not specify mechanisms for payment or contract enforcement. In particular, we assume that buyers have an external supply of currency in arbitrary units, but we do not define how the currency is replenished or recycled through the economy. We envision that each user or group is assigned a *budget* to spend on computing service over each time interval, as in previous economic resource

managers (e.g., [4, 19, 20, 22, 24]). Currency may be distributed and delegated in a decentralized or hierarchical fashion to meet global goals [22, 23].

- We focus on the policies to determine the winning bids and to schedule server resources, but we do not specify pricing mechanisms. In practice, it may be useful to charge prices below the bid price to provide incentives for buyers to bid truthfully. For example, single-commodity Vickrey auctions charge the winner a price derived from the second-highest bid for this purpose, and are used in Spawn [22] and others.
- Our bidding and negotiation protocol uses sealed bids, and we do not include mechanisms to advertise price signals to buyers in order to balance supply and demand. Given sufficient market volume, it may be sufficient to publish summaries of recent contracts as a basis for competitive bidding.

Although we focus on a service market, our approach shows how a task service provider can use risk and reward measures to quantify its gains from the underlying computational resources that host the task service. We suggest that a service provider may use these measures as the basis for a bidding strategy for raw resources in a computational resource market (e.g., [4, 5, 13, 22, 24]). Thus, our approach is complementary to previous work in market-based grids that sell resources as opposed to application service (see Section 7).

3 Value Functions

Each server site makes admission control and scheduling decisions based on the task value functions. These functions give an explicit mapping of service quality to value, exposing information that allows each site to prioritize tasks more effectively. Value-based scheduling is an alternative to scheduling for deadline constraints, which give the system little guidance on how to proceed if there is no feasible schedule that meets the constraints, e.g., due to unexpected demand surges or resource failures.

A value function specifies the value of the service to the user for a range of service quality levels—in this case, expected task completion times. The key drawback of the user-centric approach is that it places a burden on users to value their requests accurately and precisely. The more precisely users can specify the value of their jobs, the more effectively the system can schedule their tasks.

The formulation of value functions must be simple, rich, and tractable. We adopt a generalization of the *linear decay* value functions used in Millennium [9, 10], as illustrated in Figure 2. Each task i earns a maximum value $value_i$ if it completes at its minimum run time $runtime_i$; if the job is delayed, then the value decays linearly at some

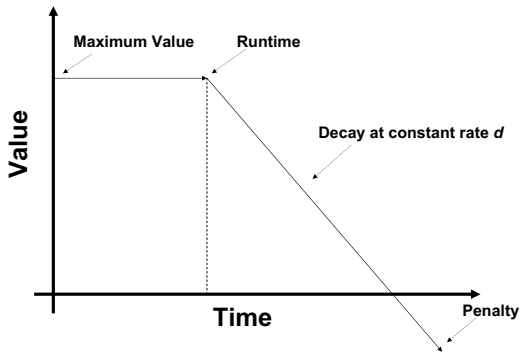

Figure 2. An example value function. The job earns a maximum value if it executes immediately and completes within its minimum run time. The value decays linearly with queuing delay. The value may decay to a negative number, indicating a penalty. The penalty may or may not be bounded.

constant *decay* rate $decay_i$ (or d_i for short). Thus if task i is delayed for $delay_i$ time units in some schedule, then its value or yield is given by:

$$yield_i = value_i - (delay_i * decay_i) \quad (1)$$

These value functions create a rich space of policy choices by capturing the importance of a task (its maximum value) and its urgency (decay) as separate measures. Jobs are more urgent if expensive resources (such as people) are waiting for them to complete, or if they must be finished before real-world deadlines. The framework can generalize to value functions that decay at variable rates, but these complicate the problem significantly.

Our framework for a market-based grid allows for a *penalty* if a task is delayed beyond the point at which its value goes to zero. Penalties provide a disincentive for a site to accept too much work or violate its contractual obligations by discarding an accepted task if circumstances prevent the site from completing the task in a timely fashion. The value function may specify a *bound* on the penalty. If a task i's penalty is bounded, then $expire_i$ is its *expiration time*—the time when i's value function stops decaying. Value functions are bounded at zero in Millennium; the system incurs no cost even if it discards an expired task and never completes it.

4 Value-based Scheduling

A scheduler is driven by a sequence of task arrival times (*release times*) $[arrive_0, ..., arrive_i]$ and task completion/departures. It maintains a queue of tasks awaiting dispatch, and selects from them according to some scheduling heuristic. Once the system starts a task, it runs to completion unless preemption is enabled and a higher-priority task arrives to preempt it. RPT_i represents task i's expected Remaining Processing Time—initially its $runtime_i$.

Two common scheduling algorithms are First Come First Served (FCFS), which orders tasks by $arrival_i$, and Shortest Remaining Processing Time (SRPT), which orders by RPT_i. These baseline algorithms do not consider user-centric measures of value. We consider the value-based scheduling problem under the following simplifying assumptions:

- The processors or nodes within each grid site are interchangeable. If preemption is enabled, then a suspended task may be resumed on any other processor. Context switch times are negligible.
- The system never schedules a job with less than its full resource request, e.g., jobs are always gang-scheduled using common backfilling algorithms with the requested number of processors. For simplicity we assume that the resource request is a single processor or node.
- The predicted run times $runtime_i$ are accurate. There is no interference among running tasks, e.g., due to contention on the network, memory, or storage. We do not consider I/O costs [2] or exceedance penalties for underestimated runtimes at this stage in our research.

A candidate schedule determines the expected next start time $start_i$ and completion time for each task i. The completion time is given as $start_i + RPT_i$ assuming the task is not preempted. Thus the expected value $yield_i$ for $task_i$ in a candidate schedule is given by Equation 1 and the delay for the expected completion time:

$$delay_i = start_i + RPT_i - (arrival_i + runtime_i) \quad (2)$$

Scheduling based on linearly decaying value functions is related to well-known scheduling problems. Total Weighted Tardiness (TWT) seeks to minimize $\sum_i d_i T_i$ where d_i is the weight (or decay) and T_i is the tardiness of job i. A job i is *tardy* if it finishes after a specified deadline. We focus on the variant in which each job's deadline is equal to its minimum run time, so that any delay incurs some cost. If penalties are unbounded, then this problem reduces to Total Weighted Completion Time (TWCT), which seeks to minimize $\sum_i d_i C_i$ where d_i is a job's weight (equivalent to decay) and C_i is its completion time in the schedule [1].

The off-line instances of TWT and TWCT are both NP-hard. This paper considers on-line heuristics and their use by server sites for value-based scheduling [3, 8] in a computational economy. A key difference in this context is that each task has a value as well as a weight. Tardiness problems ignore value on the premise that the scheduler is constrained to execute every arriving task; in this case the values are not significant, and the only goal is to minimize loss of value as given by the weights. For example, the best known heuristic for TWCT is Shortest Weighted Processing Time (SWPT). SWPT prioritizes tasks according to the task's d_j/RPT_j, and is optimal for TWCT if all tasks arrive at the same time.

However, with admission control (as in a grid economy) or bounded penalties (as in Millennium), server sites have the option to reject or abandon any given job. In these cases the scheduler must also consider reward or the value to be earned by executing the job. For example, the Millennium *FirstPrice* heuristic prioritizes tasks greedily according to the expected yield per unit of resource per unit of processing time ($yield_i/RPT_i$). We refer this value as *unit gain* or *utility* since it is a measure of yield per unit of resource per unit of time. The Millennium study refers to it as the task's *price* in the schedule, although the actual price charged in an economy may be lower.

Note that it is also important to consider risk, weight, or decay. For example, if a task is preempted and delayed further, its yield declines according to $decay_i$, and hence its unit gain or price drops. Importantly, the system earns the lower unit gain even on the resources already invested in the task. Thus running a task is a risky investment: if the task is delayed at some later time then the gain is lower than expected. Our approach follows this investment metaphor: the heuristics discount future gains and consider opportunity cost to obtain a configurable balance of risk and reward.

4.1 Experimental Methodology

To explore the effectiveness of the scheduling heuristics, we built a simulator for a bidding and task service economy with linear value functions. The simulator includes a scheduling module that could run directly in a grid-based batch engine.

In our experiments we use synthetic traces consisting of a mix of single-processor compute jobs. The traces are representative of real batch workloads as characterized in previous trace studies [12, 16, 18]. These studies show that exponentially distributed inter-arrival times are

common in batch workloads. While job durations are not always exponential, recent research indicates that durations rarely affect the relative ranking of scheduling algorithms [16]. Most experiments use exponentially distributed inter-arrival times and job durations, though in some cases we use normal distributions to reproduce and compare to results from the Millennium study.

Previous studies give no guidance on how users value their jobs, since no traces from deployed user-centric batch scheduling systems are available. This is unfortunate because the distribution of maximum values and penalties has a significant impact on the results. We chose the same bimodal distributions for value as the Millennium study. The value assignments are normally distributed within *high* and *low* classes: unless otherwise specified, 20% of jobs have a high $value_i/runtime_i$ and 80% have a low $value_i/runtime_i$. The ratio of the means for high-value and low-value job classes is the *value skew ratio*. The Millennium study did not investigate the behavior of job mixes with varying decay rates, but we chose to adopt a similar bimodal distribution for the decay rates. The decay rates are assigned in a similar fashion parameterized by a *decay skew ratio*.

We limit our investigation to relative performance and basic insights about scheduling using value and decay. Many interacting characteristics of the job mixes play key roles in determining the results. For example, the magnitude of all results is dependent on the *load factor*, i.e., the total requested work over any interval, divided by total capacity. Higher load factors increase the importance of effective scheduling. Most experiments hold load factors constant and close to saturation (load factor one), forcing the system to delay or reject some tasks but allowing it to complete and earn value on a majority of tasks. Other trace properties that affect results include the distributions of value, decay, job duration, and inter-arrival times.

5 Risk/Reward Heuristics

This section develops a parameterized heuristic called *FirstReward* for value scheduling with linear value functions. We first consider the problem of scheduling a sequence of arriving tasks to maximize value. Section 6 shows how to use the heuristic as a basis for negotiation and admission control (task rejection).

For experiments in this section, the scheduler receives a trace of 5000 jobs representative of the workload characteristics, and the experiment runs until the system has completed all jobs. These experiments are conservative in that the trace parameters are stable through the experiment and the scheduler must run all tasks, rather than using its heuristics to select tasks from the mix.

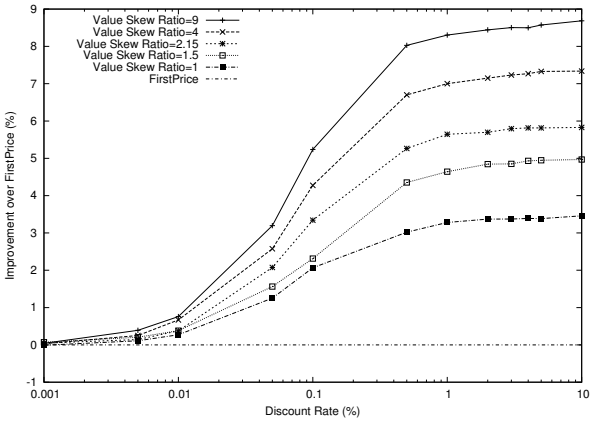

Figure 3. Yield improvement for Present Value (PV) relative to FirstPrice for variants of a task mix used in the Millennium study, with load factor 1. At discount rate 0 PV is equivalent to FirstPrice. Yield improves for modest increases in the discount rate along the x-axis. The improvement is larger for workloads with a higher variance in task value.

5.1 Discounting Future Gains

We first extend the notion of yield and unit gain to account for the risk of gains deferred to the future. For example, given two tasks with the same unit gain and urgency, it may be preferable to run the shorter task first, since it carries a lower risk of preemption by a newly arriving task before the investment pays off. Even without preemption, the shorter task carries a lower risk of delaying a valuable or urgent task that arrives later. In either case, a risk-averse scheduler might even choose to run a lower-yield task if it can realize its gains quickly.

Our approach is based on the notion of *present value* in finance. The present value of a task i is defined as:

$$PV_i = yield_i/(1 + (discount\_rate * RPT_i)) \qquad (3)$$

This formula is standard for the present value of an investment instrument with face value $yield_i$ that matures in time RPT_i: an asset with value PV_i earning simple interest at $discount\_rate$ matures to yield a payoff $yield_i$ after a period RPT_i. The $discount\_rate$ is a tunable parameter: higher discount rates cause the system to discount future gains more aggressively, making the system more risk-averse. We introduce a heuristic called Present Value or *PV* that selects jobs in order of discounted unit gain PV_i/RPT_i.

Figure 3 shows the yield improvement of *PV* relative to *FirstPrice* for a standard task mix from the Millennium study, with varying value skew ratio. The inter-arrival

times and job durations are normally distributed, with 16 jobs submitted in a batch on each arrival. The decay rates are the same across all tasks in each mix, and penalties are bounded at zero. Preemption is enabled.

As we increase the discount rate, Figure 3 shows that the *PV* heuristic improves yield modestly even for these stable workloads. Discounting future gains is more important with higher value skew ratios, which imply a higher risk that higher-value jobs arrive after resources are committed to a long job but before the job completes.

5.2 Opportunity Cost

Next, we extend the reward heuristic to consider losses from the *opportunity cost* to select a given task i instead another queued task j, causing j's yield to decay. If task j is more urgent than i, i.e., its value decays more rapidly, then it may be better to prefer task j even if it has a lower unit gain. A task's opportunity cost depends only on the urgency of competing tasks. If the scheduler must eventually complete all tasks, then it should consider only cost and ignore gains completely: it will obtain the full yield for each task, minus the cost for any delay it incurs in the schedule. More generally, the system can safely defer a less urgent task with a low decay rate d, even if it has a high value or a high unit gain. This is the essence of the well-known SWPT heuristic discussed in Section 4.

The opportunity cost $loss_i$ to start a candidate task i at some point in the schedule is given by the aggregate decline in yield of all competing tasks over time RPT_i, starting from that point:

$$cost_i = \sum_{j=0;i\neq i}^{n} d_j * MIN(RPT_i, expire_j) \qquad (4)$$

If penalties are bounded and tasks expire, then it takes $O(n)$ time to compute the cost for any task in a set of n tasks, and the scheduler can find the least-cost task in at most $O(n^2)$ time. If penalties are unbounded, then we can simplify the formula to compute a per-unit cost:

$$cost_i/RPT_i = \sum_{j=0;i\neq i}^{n} d_j = \sum_{j=0}^{n} d_j - d_i \qquad (5)$$

Since the first term is a constant for all tasks in the mix, each scheduling step becomes $O(n)$ at worst, and it can take $O(\log n)$ time if the current schedule is represented as a heap. Scheduling to minimize this per-unit cost is equivalent to SWPT.

5.3 Balancing Gains and Opportunity Cost

It is risky to defer gains from a high-value task on the basis of opportunity cost alone. *FirstReward* uses a reward metric that combines each task's expected gain with its opportunity cost per unit of resource per unit of time, weighted by a tunable parameter α:

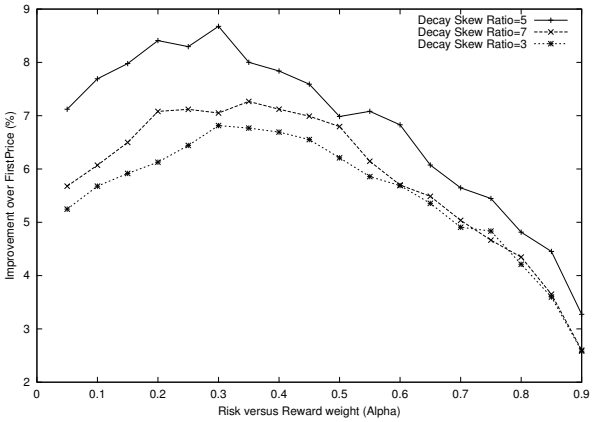

Figure 4. Improvement of FirstReward over FirstPrice as the α parameter varies, for job mixes with bounded penalties and varying decay skew ratios. It is more important to consider cost (low α) than gains in this case, and the importance of cost increases with the variability of the decay rates. The hybrid heuristic works best overall, and is most effective around $\alpha = 0.3$.

$$reward_i = ((\alpha)PV_i - (1-\alpha)cost_i)/RPT_i \qquad (6)$$

The α parameter controls the degree to which the system considers expected gains. With $\alpha = 1$ and $discount\_rate = 0$, the heuristic reduces to *PV*. With $\alpha = 0$ it reduces to a variant of SWPT. The general form considers discounted future gains and opportunity cost to varying degrees to balance risk and reward, given appropriate settings for $discount\_rate$ and α.

Figure 4 shows the effect of varying α for three traces with bounded penalties and different decay skew ratios. We hold the value skew ratio constant at 2; the discount rate is 1%. Although the best α depends on the properties of the task mix, other experiments have shown that generally the ideal is $\alpha < 0.5$. Interestingly, SWPT ($\alpha = 0$) is a good approximation to minimize cost even with bounded penalties. However, gain is also a good predictor of cost when penalties are bounded: with high α the heuristic biases against low-value jobs, which have the least value to lose. Since decay rates are not correlated with value, the low-value jobs tend to reach their bounds and expire faster; once a task has expired it may be deferred to the end of the schedule with no further cost.

In contrast, Figure 5 shows that there is little benefit to considering gain when the penalties are unbounded. In

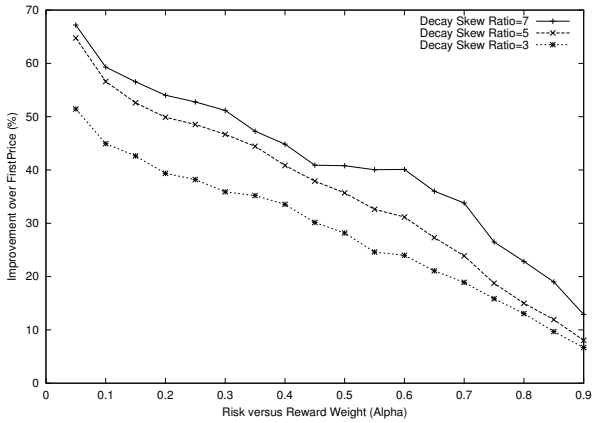

Figure 5. This experiment is identical to Figure 4, but the penalties are unbounded. In this case, where the system must accept and complete all jobs, it is never useful to consider gains, only cost. Note that the magnitude of the improvement relative to FirstPrice is much larger with unbounded penalties.

these cases, the relative advantage of a cost-based heuristic can be an order of magnitude or more depending on the urgency and decay skew in the task mix. Again, in these experiments the scheduler is constrained to execute all submitted tasks. The next section considers use of these heuristics for server bidding and admission control, where it is always important to balance risk and reward.

6 Negotiation and Admission Control

We propose a market-based task service in which clients submit task bids to server sites or brokers, specifying each task i's expected run time and its value function as a tuple $(runtime_i, value_i, decay_i, bound_i)$. Each site may choose to accept or reject any submitted task. If it accepts the task, it negotiates to establish a price and expected completion time.

Each site maintains a pool of tasks it is contracted to run, and a candidate schedule for its n pending tasks. It executes the following procedure for each proposed task:

- Integrate the task into the current candidate schedule according to the heuristic (e.g., *FirstReward*), as described in Section 5.
- Determine the expected yield for the task if it is accepted, by evaluating its value function for its expected completion time in the candidate schedule.
- Apply an *acceptance heuristic* to determine if it is worthwhile to accept the task into the current task mix, as described below.
- If the task is worthwhile, then accept the client's bid and issue a server bid to the client. The server bid represents the expected completion time for the task in the candidate schedule, and the expected price to run the task. Our site policies act as if the price is derived directly from the original value function, i.e., client bid value and price are equivalent, although a pricing strategy may propose a different price as discussed in Section 2.
- If the contract is accepted by the client, then incorporate the task into the schedule and execute it. It is possible that later arrivals will delay its completion time, in which case the site receives a lower price or pays a penalty as described in Section 3.

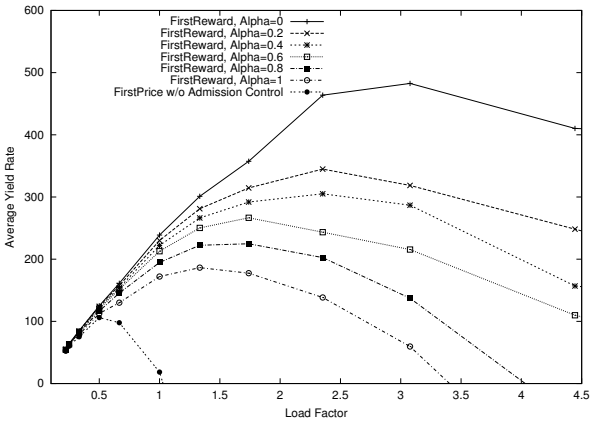

Figure 6. Admission control allows sites to select tasks with high reward and low risk in the current candidate schedule. The graph gives the yield per unit of time for task streams with increasing loads along the x-axis, and different values of α in the FirstReward heuristic.

The acceptance heuristic should consider both the potential reward and the degree to which the task would constrain acceptance of future tasks into the mix. For example, urgent tasks incur more risk because they increase the opportunity cost to run any new high-yield tasks that arrive later, and so decrease their reward. We propose a simple acceptance heuristic based on the *slack* for a submitted task i—the amount of additional delay (beyond its place in the candidate schedule) that the task can incur before its reward falls below some yield threshold. Without loss of generality, we take the yield threshold equal to zero, i.e., the point at which the task loses the system money, either by incurring a penalty or by delaying other accepted tasks. The slack is computed from the current point in the candidate schedule:

$$slack_i = \frac{PV_i - cost_i}{decay_i} \quad (7)$$

The cost of running the task is an estimate of the impact on those tasks j that are behind i in the candidate schedule, i.e., those tasks that will be delayed more than expected by accepting this new task i:

$$cost_i = \sum_j decay_j * runtime_i \quad (8)$$

The acceptance policy rejects tasks whose slack falls below some *slack threshold*. Under higher load, with more tasks queued, new tasks will tend to be placed further back in the schedule, and will also tend to have more tasks behind them. Thus high load tends to both reduce the potential yield and increase the cost of each new task, both of which reduce its slack. The slack also captures the risk of accepting the task, as determined by its decay rate and its position in the schedule. Accepting a low-slack task constrains the site's flexibility to accept other higher-value tasks in the future, since the gains for those tasks would be partially offset by the additional opportunity cost to run them, i.e., the yield loss and potential penalty for the low-slack task. This opportunity cost may force the site to reject other profitable tasks later because the cost of delaying an urgent task is too high.

Figure 6 shows the effect of admission control on the value earned per unit time. Each point is for a representative trace of 5000 submitted jobs with exponentially distributed task durations and inter-arrival times, and unbounded penalties. The value skew ratio is 3, and the decay skew ratio is 5. The jobs arrive over shorter time intervals as load factor increases along the x-axis. The system runs until the scheduler completes all accepted jobs; we then obtain the average yield per unit time over the active interval. The lines on the graph show the effect of different settings for α, which controls the degree to which *FirstReward* considers gains (high α) vs. cost (low α). The discount rate is 1%, and the slack threshold is 180.

We can see from Figure 6 that admission control is critical to maintain the yield rate under heavy load. Without admission control, the system is forced to accept too many tasks, and delays and penalties eat away at gains. With admission control we see that increasing load factor initially increases the yield per unit time, since the scheduler has more tasks to choose from and is free to reject the tasks that are least worthwhile—relative to its current mix—without penalty. Note that the admission control heuristic considers both gains and decay rate in its computation of slack time. Even so, in this unbounded penalty case it is still more important to order submitted tasks in the candidate schedule to minimize their costs before computing their slack time.

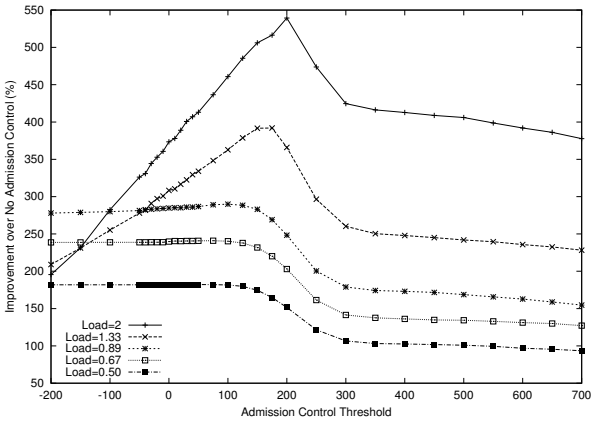

Figure 7. The admission control (slack) threshold has a peak that balances risk and reward for a given load factor. It is more important to set the slack threshold correctly at higher load levels.

Our last experiment, Figure 7, shows how the slack threshold affects value earned by a schedule. The task mixes are similar to Figure 6, but we vary the admission control (slack) threshold along the x-axis, for selected load factors. Higher load offers the potential for higher yields, since the server can "cherry pick" the task mix; however, higher load also presents higher risk if the admission control heuristic is ineffective. The graph shows that the ideal slack threshold changes depending on the load factor: higher load requires a more risk-averse admission control policy that applies a higher slack threshold. Setting the threshold is a balance of risk and reward; a high threshold prevents commitments that could incur penalties while a low threshold commits to potentially costly tasks.

7 Related Work

Cluster batch schedulers. Common batch schedulers such as GridEngine [21] and LSF [17] support variants of priority FCFS scheduling. LSF, as an example, provides hierarchical grouping and proportional share mechanisms. These weighting and priority mechanisms may be viewed as coarse-grained assignments of value to jobs. User-centric scheduling enables fine-grained assignments of value and urgency for customers bidding for task service in a computational economy. Millenium [9, 10] applied this idea to a cluster batch scheduler, in which the scheduler is constrained to accept all jobs from its user community. Stoica et. al. [19] considered economic scheduling for parallel jobs.

Job scheduling. The literature on scheduling algorithms is too large to survey here. Karp first proposed

the Job Scheduling with Deadlines (JSD) problem in 1972 [14]. JSD was proved to be NP-complete in the number of jobs by reduction from the $0/1$-Knapsack problem. The problem we consider has no deadlines, but imposes variable costs for any delay, which is amenable to heuristic solutions. As discussed in Section 4, the problem may be viewed as an extension of Total Weighted Completion Time [1] incorporating value as well as decay weights. Related forms of value-based scheduling have also appeared in real-time systems [3, 8]. We apply simple heuristics based on a financial metaphor of risk and reward, and show how to apply them in a grid economy.

Grid scheduling. Many systems schedule tasks across sites in a federated grid. For example, the Superscheduler Architecture [18] selects a server site for each task based on estimates of wait time, e.g., under FCFS schedules at each site. Our approach enables the client or broker to select a server site that balances wait time and value, and reorders tasks at each site based on value and urgency. It is an open question how to extend our approach to consider I/O costs and task placement for I/O, as in BAD-FS [2].

Federated computational economies. Previous works on computational economies for federated resource allocation include Spawn [22], Mariposa [20], the G-commerce framework [24], and Nimrod-G [4]. Our proposal is similar in that it is based on two-phase negotiations between server sites and buyers endowed with budgets and knowledge of their resource demands.

Several of these works have examined dynamics of computational markets based on various pricing and auction systems. For example, Spawn [22] was a landmark system based on second-price Vickrey auctions, an approach that many other systems have followed. Pricing and market dynamics are beyond the scope of this paper, although our heuristics are compatible with other pricing schemes. A key focus of our work is to extend these economic approaches to incorporate penalties for violating contractual obligations, and to develop strategies that balance risk and reward with respect to current commitments when accepting and scheduling tasks.

Service markets and service quality. In contrast to most of these systems, the sites in our system sell a *service* rather than raw resources such as blocks of CPU time. Service markets introduce a new dimension: the service may be provided at a range of quality levels. Mariposa [20] is similar in that it uses economic bidding to distribute queries in a database system, in which the location of stored data is significant. Our approach follows Mariposa in bidding for service based on user value that decays with service delays. Contractual agreements for service quality are essential for computing utilities and the next-generation grid [11].

One goal of our work is to create a foundation for service providers to buy or sell raw resources in an underlying resource market, based on current demand for the service they provide. Our proposed task service may use its internal measures of per-unit gain and risk as a basis for its own pricing and bidding strategy in a resource market. That is, the task service may act as a reseller of resources acquired from a shared resource pool as envisioned in our previous work on SHARP [13], Muse [6], and Cluster-on-Demand [7]. Ultimately, the service provider must establish a utility function capturing the value it assigns to the resources it might acquire, based on the improvements in service quality and yield that would result. Resource allocation may then be viewed as a bidding problem [5] or an optimization problem [15].

8 Conclusion

This paper develops heuristics for task scheduling and admission control in a market-based grid task service, and illustrates their behavior under representative conditions. The foundation for the market-based approach is user-centric or value-based scheduling, which prioritizes tasks according to a user-specified measure of value. Since user value corresponds to what the user will pay for the task service, scheduling to maximize user value allows a market-based task service to maximize its yield. The urgency of a task is captured by the rate at which its value decays with increasing wait times.

Our approach is based on a financial metaphor for reasoning about scheduling choices in a service economy. We propose a scheduling heuristic called *FirstReward* that is parameterized to balance the risk or cost of a task with the reward of completing it. One conclusion of our study is that cost and risk are often more important than gains in determining scheduling effectiveness, although heuristics that consider gains to some degree are more effective in some cases, particularly when penalties are bounded.

This paper is a step towards understanding the balance of risk and reward for scheduling based on fine-grained expressions of value. The contributions include: detailing the different areas of scheduling risk, defining heuristics to mitigate those risks, exploring the parameter space for a general scheduling heuristic, and showing how value-based schedulers can drive server bidding and admission control in a computational economy.

Acknowledgments

We thank Brent Chun for discussions about user-centric scheduling in Millennium, and the anonymous reviewers for their comments, which helped to improve the paper. Terence Kelly helped us to understand related scheduling work.

References

[1] I. Baev, W. Meleis, and A. Eichenberger. Algorithms for total weighted completion time scheduling. *Algorithmica*, 33(1):34–51, May 2002.

[2] J. Bent, D. Thain, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Explicit control in the batch-aware distributed file system. In *First Symposium on Networked Systems Design and Implementation*, March 2004.

[3] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46(4):305–325, January 2000.

[4] R. Buyya, D. Abramson, J. Giddy, and K. 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)*, 14(13-15), Nov.-Dec. 2002.

[5] A. Byde, M. Salle, and C. Bartolini. Market-based resource allocation for utility data centers. Technical Report HPL-2003-188, HP Laboratories Bristol, 2003.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, pages 103–116, October 2001.

[7] J. S. Chase, L. E. Grit, D. E. Irwin, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[8] K. Chen and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-Time Systems*, 10(3):293–312, 1996.

[9] B. Chun. *Market-based Cluster Resource Management*. PhD thesis, University of California at Berkeley, November 2001.

[10] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *2nd IEEE International Symposium on Cluster Computing and the Grid*, May 2002.

[11] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002.

[12] A. Downey and D. Feitelson. The elusive goal of workload characterization. In *Performance Evaluation Review*, pages 14–29, March 1999.

[13] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[14] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, March 1972.

[15] T. Kelly. Utility-directed allocation. In *First Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.

[16] V. Lo, J. Mache, and K. Windisch. A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing*, March 1998.

[17] LSF. http://www.platform.com.

[18] H. Shan, L. Oliker, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *Proceedings of Supercomputing (SC2003)*, November 2003.

[19] I. Stoica, H. Abdel-Wahab, and A. Pothen. A microeconomic scheduler for parallel computers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS)*, pages 122–135, April 1995.

[20] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *3rd International Conference on Parallel and Distributed Information Systems*, September 1994.

[21] Sun's GridEngine. http://gridengine.sunsource.net/.

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

[23] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *First Symposium on Operating Systems Design and Implementation*, November 1994.

[24] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational Grid. *The International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.