Profitable services in an uncertain world

Florentina I. Popovici and John Wilkes
University of Wisconsin-Madison and HP Laboratories
popovici@cs.wisc.edu and john.wilkes@hp.com

In a service-oriented, utility-computing, Grid-like world, service providers will execute jobs on behalf of their clients on systems rented from resource providers. This poses many challenges to the service provider, such as choosing which jobs to admit, when to run them, whether to execute them on one system or many, and how many resources to rent. To complicate matters, the service provider may experience resource uncertainty—an inability to get the resources it needs or expects. The result will be sub-optimal choices of which jobs to accept and when to run them, and the service provider may have to pay penalties to its clients. Using an economics-based approach, we have developed scheduling policies that systematically address these problems. We show that the new policies deliver significantly more profit (or added value) than ones oblivious to such concerns.

1 Introduction

Consider the situation of a *service provider* that offers job-based services to its clients, which can be commercial clients, scientific partners, or other entities. Such service providers are likely to become more prevalent as the Grid matures, and as service-oriented computing becomes more widely deployed in the commercial world.

The results delivered by a job have value to the client. This is simple to describe in an economics-based approach: clients express the value of their jobs as the price they will pay to have them run, and the gap between this price and the cost to run the job is simply the job's *profit* to the service provider. The goal of the service provider becomes to maximize its *profit rate*, its net return per unit time, modulo concerns such as maintaining adequate quality of service, user satisfaction, and other intangibles.[1] It seems self-evident that profit is a valid metric in the commercial world; we also believe that it is useful in academic and scientific circles because it offers a clear, numerical measure for the amount of net value added by a service.

It is helpful to contrast this job-based service with one based on resource rentals, in which the client runs an application themselves, leaving few degrees of freedom to the resource-provider service. The higher-level service we describe here adds value by relieving a client of a great deal of work (e.g., acquiring, setting up, and executing software, establishing a scalable environment, fault-tolerance, and so on), allows the service provider to charge a fair price for its expertise, and leaves it with more freedom in making decisions, such as which jobs to run, when to run them, and whether to run them on one resource or many (the job's *shape*). The result can be economies of scale and expertise, as multiple clients' jobs are mapped onto the same service infrastructure, and costs such as expensive software licences can be amortized across multiple users.

Clients describe how much timely delivery of results matters to them by providing a value (price) profile as a *utility function* of the form shown in Figure 1. A common form is a fixed value (price) up to a specified time, after

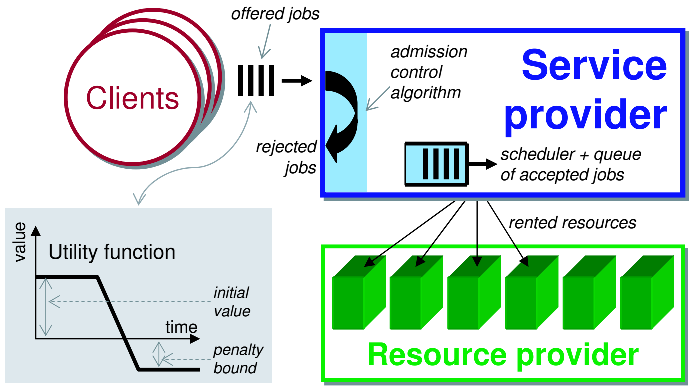

Figure 1. Problem overview. *The relationship between clients, service providers, and resource providers.*

[1] In other work, we are exploring how best to bind additional metrics of this type into the service price; we also defer questions of price-setting in the face of competition between service providers, observing that this only increases the incentive to make efficient use of resources.

which the value declines linearly for late delivery until a fixed, bounded penalty is reached. When a job finishes, and its results are handed back to its client, the price of the job is the value specified at that moment by the job's utility function.

1.1 Resource rental and uncertainty

Executing jobs has costs as well as rewards: processors and other resources cost money, too. Because the amount of work it receives will vary, the service provider can reduce its risk by renting the resources it needs from a *resource provider*, rather than owning them.

To allow service providers to decide whether to accept work, a resource provider will offer predictions of how many resources it will have available when, using a set of tuples of the form <*start-time, duration, resources, price*>.

Because resource providers also need to make money, they will try to maximize the return they receive on the fixed assets they own. For example, a resource provider might over-book resources if demand is uncertain; or a service provider might make it an offer it cannot refuse for resources that have already been promised to another; or it might suffer equipment failures, and have to prioritize its service-provider clients' demands.

One result is uncertainty in resource availability for the service provider, which may expose it to the risk of accepting more work than can be handled, and having to pay out penalties for delay or non-performance. What is a service provider to do? We believe that taking account of the potential risk and uncertainty is better than pretending it does not exist.

Assuming that sufficient history has built up to allow the accuracy of the resource provider's estimates to be characterized and modeled, we show how the service provider can make use of this information in its scheduling process to make informed tradeoffs between risk and return.

1.2 Contributions

The main contributions of this work are the description and evaluation of a new family of profit-based scheduling and admission control algorithms for higher-level service providers that (1) explicitly address the cost of renting resources; (2) handle variable-shaped jobs (ones that can be run on one or more processors) that scale imperfectly; and (3) explicitly address resource-availability uncertainty.

The new algorithms are compared with previously-proposed approaches, and evaluated across a range of operating conditions: load, resource price and quantity, utility function shape (client impatience), and resource uncertainty level. Our experiments show that the new algorithms can extract higher profit rates than the previous ones, and we provide data to help explain why this is so.

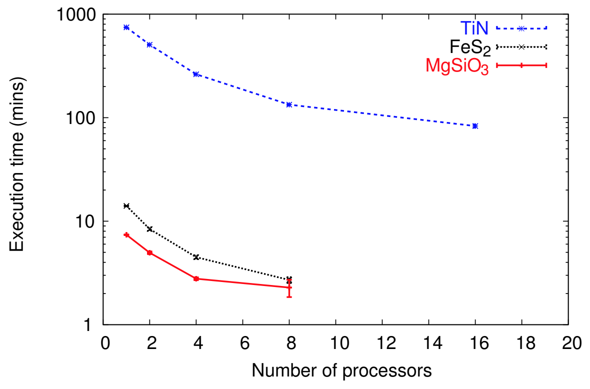

Figure 2. CASTEP job scale-up behavior. *How job-execution time varies as a function of molecule type and number of processors. These experiments were performed on a 500-node cluster of HP DL360 processors.*

1.3 CASTEP

Because there is much potential variability in work such as this, we chose to adopt the same values used by prior studies for factors such as job mix and utility-function settings. But rather than just inventing or postulating the effects of changing job shapes on running times, we eliminated one source of load-based uncertainty by driving our simulations with measurements taken from a real parallel application – CASTEP.

CASTEP [Segall2002] is a quantum-mechanics package that calculates bond lengths and energy levels for molecular structures. The execution times of a single CASTEP job can vary from a few minutes to many hours (see Figure 2); typically, each job is part of a larger set that explores a molecular structure of interest. A single CASTEP job can be scaled to run in parallel on many machines, trading off shorter elapsed time for greater total resource usage, because of startup and communication overheads. The number of machines has to be a power of 2 for CASTEP, which in practice bounds the number of alternate shapes that need to be considered during admission control and scheduling.

We assume that service providers use their domain expertise to estimate the running times of jobs in advance (e.g., from a historical database of prior runs), which in turn helps them to predict the cost of running jobs. This is not unreasonable: there is a great deal of literature in the field of workload characterization; [Chiang2001, Squillante1999] are just two examples.

Although, for simplicity, we assume a single CASTEP service provider and a single resource provider that rents out physical machines, our results can easily be generalized, and we believe that our experiments are useful predictors across a range of situations. In particular, CASTEP stands in for a wider range than normal of such loads, because it can be tuned to behave like many of them.

2 Related work

Our work builds most closely on two previous utility-computing studies: *Millenium* [Chun2002] and *RiskReward* [Irwin2004]. They both considered a similar setup to ours, except that the service provider and resource provider roles were merged, with a fixed-sized pool of nodes on which to run jobs; they did not explicitly consider resource costs, reshapable jobs, or resource uncertainty.

The earlier of the two, Millenium, selected jobs to run by calculating the value $yield_i/RPT_i$ for each job i, where RPT is the remaining processing time and *yield* the value (price) obtained when the job finishes. This heuristic gives priority to jobs that generate high value per unit running time.

RiskReward extended Millenium by introducing a discount rate for future rewards, reflecting the increasing risk of making decisions that bind the service provider into the future. With this, *PV* (Potential Value) becomes an estimate of potential future income: $PV_i = yield_i/(1+discount\_rate \cdot RPT_i)$. Similarly, an estimate was added of the penalty caused to other jobs j by delaying them to run new job i. The resulting $delay_{i,j}$ is the smaller of RPT_i and the time for the affected job to hit its lower penalty-bound. The $decay_j$ term is the slope of the penalty portion of the job's utility function: $cost_i = \sum_{j=0,j\neq i}^{n}(decay_j \cdot delay_{i,j})$. The final *FirstReward* heuristic balanced potential future rewards against potential opportunity-cost penalties by means of a simple weighting function: $reward_i = (\alpha \cdot PV_i - (1-\alpha) \cdot cost_i)/RPT_i$.

The utility functions used by FirstReward assume that the job's value stays constant over its normal running time, and starts to decay immediately thereafter. Although we implemented support for arbitrary straight-line-segment, monotonic-decreasing utility-functions, we chose to model clients that defined the utility function independently of the job's running time, on the grounds that value to the clients is probably more affected by what they will do with the results than it is by the running time – which they might not even know in advance. This independence allowed our scheduler to take advantage of the job's scale-up function, and run it on more processors in parallel if that made sense. This is an extension of prior work: [Irwin2004] looked only at jobs that ran on a single processor.

Both Millenium and RiskReward allowed jobs to be preempted, although Millenium only allowed each job to be preempted once, and found little benefit unless the late-delivery penalty increased rapidly. We chose not to allow preemption in our experiments, because of how hard it is to implement in a gang-scheduled multi-processor context without application-level modifications.

Space precludes more than a cursory mention of some other related topics:

- Alpha OS [Clark1993] handled time-varying utility functions; [Chen1996e] discussed how to do processor scheduling for them; [Lee1999b] looked at tradeoffs between multiple applications with multiple utility-dimensions; and [Petrou2004] described using utility functions to allow speculative execution of tasks.
- Libra [Sherwani2004] is a scheduler built on top of a proportional-share load balancer that required users to estimate running time, a target deadline, and the value for a job; it did not support time-varying utility functions.
- The MUSCLE scheduler [He2004] handles variable-shape ("moldable") jobs with soft deadlines, and emphasizes efficient bin-packing and load-balancing across multiple resource clusters rather than value, risk, or uncertainty management. The techniques it uses complement ours, and might improve our job-placement heuristics.
- Scheduling is a well-developed field of study, both inside and outside computer science, and we don't have space here to do it justice. One relevant, representative sample may help to give a flavor of the state of the art: [Sun2004] looks at scheduling work (jobs) in a bounded-capacity factory (service provider) that is dependent on multiple suppliers for its parts (resources).

[Irwin2004] has a good survey of other relevant work.

3 Our approach

As described above, we model a service provider that runs a stream of jobs on behalf of a set of clients. Each job has associated with it a utility function; we assume that the utility value represents the price that the client is willing to pay. Resources to run the jobs are obtained from a separate resource provider.

3.1 Jobs

When it is offered a job, the service provider first executes an *admission control algorithm*, which decides whether it should accept the job; if it decides that the likely profit will decrease, it rejects the job, and no further action is taken. If the admission control algorithm accepts the job, it is placed into a work queue, from which it is selected at some future time by a *job scheduler*.

The normal way for an accepted job to leave the system is for it to be run to completion, at which point the service provider is credited with the current value of the job's utility function. We chose to insist that all accepted jobs be completed, regardless of the cost; other schemes are possible.

3.2 Resources

Resources to run the jobs are obtained from a separate resource provider, who offers predictions of the number of resources likely to be available at different times in the future, and their price. We assume the availability of an *uncertainty estimator*, expressed as a probability distribution of actual resource availability associated with each prediction from the resource provider. This estimator could be supplied by the resource provider, or learned (the hard way) by a service provider.

3.3 Scheduling

The scheduling problem a service provider faces is NP-hard, and only heuristic solutions are practical for large-scale systems. Such solutions (including ours) typically have two parts: an *admission policy*, which selects which jobs to accept, and a *scheduling policy*, which determines the order those jobs will be executed in.

The job scheduler maintains a preferred schedule, based on its estimates of resource availability, and attempts to execute that. The scheduler is invoked whenever a job arrives, a job completes, or the number of available resources changes; it may choose to run a job, or decide that it cannot do so yet. Preferred-schedule design is complicated by the opportunity to reshape a job, trading off more processors in order to get faster job completion at higher cost.

When the scheduler does decide to run a job, it requests resources from the resource provider; if they are available, the job is started and run to completion – resources are not preempted. If sufficient resources are not available, the scheduler can see if the preferred job can be reshaped (e.g., by running it on smaller number of machines), or it can see if it has another, smaller, job ready to run, or it may do these in the reverse order. The process repeats until the scheduler has no more jobs that can be run.

In this paper we examine the behavior of several schedulers:

1. *longest-job first (LJF)* always runs jobs on one processor, and sorts the jobs in order of decreasing predicted run time, in an attempt to reduce the number of missed deadlines.

2. *shortest-job first (SJF)* always uses a shape with as many processors as it can (up to the maximum it can scale-up to), and then sorts jobs in order of increasing predicted run time, in order to maximize job throughput.

3. *FirstPrice-reshape* is our shape-adjusting refinement of FirstPrice [Chun2002]. It considers all possible job shapes, not just one, and sorts jobs based on their predicted value $\div$ running time (i.e., ignoring resource-rental costs).

4. *FirstReward-reshape* is our shape-adjusting refinement of FirstReward [Irwin2004].

5. *FirstProfit*: a new scheduler that selects the shape of jobs in order to maximize the per-job profit for each job independently, and sorts jobs by profit alone.

6. *FirstOpportunity*: a new scheduler that examines the effect of running a job on the others in the queue. It builds a new schedule for the entire workload by trying each possible job in turn (picking only the most profitable standalone shape), and then using FirstProfit to generate a schedule for the remaining jobs. It then selects the job that generates the schedule with the highest total profit.

7. *FirstOpportunityRate*: like *FirstOpportunity*, but it selects the job that would produce the highest aggregate profit $\div$ the total schedule length.

3.4 Admission control

The admission control algorithm is responsible for determining whether it will be profitable for the service provider to accept a job. Essentially, it is trying to decide if the net increase in profit is (probably going to be) positive, at an appropriate level of risk.

Our early experiments showed that simply accepting all jobs did poorly under overload: admission control is a necessity. We then evaluated schedulers and admission control algorithms independently, fixing the latter as we varied the former, but found that the decisions made by admission control and the scheduler sometimes conflicted, so we switched to using the same scheduler in the admission-control algorithm as we used for job sequencing.

The most thorough way of doing admission control is to compute new job schedules that include the new job (one for each of its possible shapes), add in the existing schedule (without the new job), and then select the most rewarding schedule from the resulting set. Only if the new job is in that schedule should the job be admitted; this takes into account any effects it may have on the already-accepted jobs. This is exactly the approach taken by our *PositiveOpportunity* admission-control algorithm, which we used for all the experiments we report on here.

Compared to the cost of running the jobs, the execution time for the admission control algorithm is small, even with the shape variants, with the exception of the Opportunity-based schedulers when the queue lengths get large.

The metric used for "rewarding" varies by scheduler. For FirstPrice and FirstReward, it is yield (value)-rate and $reward_i$ respectively; for SJF, LJF, FirstProfit, and FirstOpportunity it is profit; for FirstOpportunityRate it is profit divided by schedule duration.

3.5 Factoring in resource-uncertainty

The resource provider offers the service provider predictions of when resources will be available and their price. As mentioned above, these predictions may be inaccurate, and so we assume that an estimate of their accuracy can be built up (e.g., by aggregating historical information). For simplicity, and without losing generality, we used a normal distribution to model the estimator, with mean equal to the resource provider's prediction; supporting other models is a trivial change.

There are two parameters to consider: (1) the amount of uncertainty (how inaccurate are the resource-provider's estimates); and (2) how much risk a service provider is willing to take, which we express as a bound on the expected probability that a bad outcome might happen. For example, 10% risk factor means a desire that a positive outcome is expected to occur at least 90% of the time.

Our scheduling and admission control policies are extended to factor in risk and resource-uncertainty. Whenever a job is considered for admission, the service provider computes a risk assessment by using the predicted resource availability, and a job is admitted only if the risk of the schedule associated with it being profitable is lower than a set threshold. A similar approach is used for scheduling. When building a schedule, the service provider will not assume it can use resources that will generate a higher risk than it is willing to tolerate.

The result is that a service provider can select a maximum risk it is willing to take, and make decisions that accord with this. This approach allows the service provider to couple its admission-control process to its likely access to resources, even in the face of uncertainty. This simple policy is both easy to apply and quite powerful, as we show below.

4 Experimental setting

We studied the different schedulers and admission control policies under different workloads and resource availability profiles by building a simulator. This section describes the experimental setup we used, and the default parameters used in our experiments. The next section presents the results we obtained.

To allow comparison with prior work, we modeled our test workloads on the ones used by Millenium [Chun2002] and RiskReward [Irwin2004]: a mixture of high-value and low-value jobs, each with a mixture of steep and shallow late-delivery penalties expressed as linear decay rates in value to fixed lower bounds. Not all offered jobs could be executed profitably with the cost, computation time, and value settings used, even if the service provider was otherwise idle.

Inter-arrival time for jobs followed an exponential distribution. Jobs were split into two utility classes, with 20% having high value, and the rest low value; the value associated with each class was normally distributed around the mean with standard deviation 0.2 times the mean. Following [Irwin2004], the ratio of these means is called *value skew*. Our utility functions have set value-decay rates, which for convenience are given positive values to represent the value lost per unit time. Ratios of decay rates between two job populations are called *decay skews*, again following [Irwin2004]. Unlike [Chun2002] and [Irwin2004], our decays started immediately, at submission time, to better model the expected behavior of clients who do not have good estimates of job running times. [Irwin2004] explains how these synthetic traces correlate to real workloads.

Table 1. Simulator parameter-settings. *Our currency units are called florins. With these settings, a mean inter-arrival time of 0.15 hours generates a potentially-profitable load for approximately twice the available resources.*

Parameter	*Default value*
Simulation length	200 jobs
Mean job inter-arrival time	0.15 hours, exponential
Mean job size	7 processor-hours
Job shape	1–5 processors
Resources available	10 processors
Resource uncertainty	none (in processors)
Mean low value job	48 florins
Mean shallow decay rate	3.7 florins/hour
Penalty value bound	–48 florins
Value skew (low:high)	3 (ratio)
Decay skew (shallow:steep)	5 (ratio)
Resource price	3 florins/hour

To handle variable job shapes, we modeled the job scale-up behavior using Amdahl's law [Amdahl1967, Kleinrock1992].[2] While our simulator can handle workloads composed of jobs that have different scale-ups, we chose to use the serial fraction derived from our experimental measurements of CASTEP jobs on our cluster: 80% of the computation was parallelizable, 20% serial.

Execution time in our simulator setup for the several hundred runs we needed imposed a few restrictions on us. To fit within our own resource envelope, we modeled a 10-processor resource pool, and 200 jobs per run. We took care to look only at steady-state performance: e.g., percentage utilization data excludes the "emptying" phase, where no

[2]Amdahl's law states that if β is the serial fraction, and N the number of parallel nodes, then $speedup = 1/(\beta + (1-\beta)/N)$.

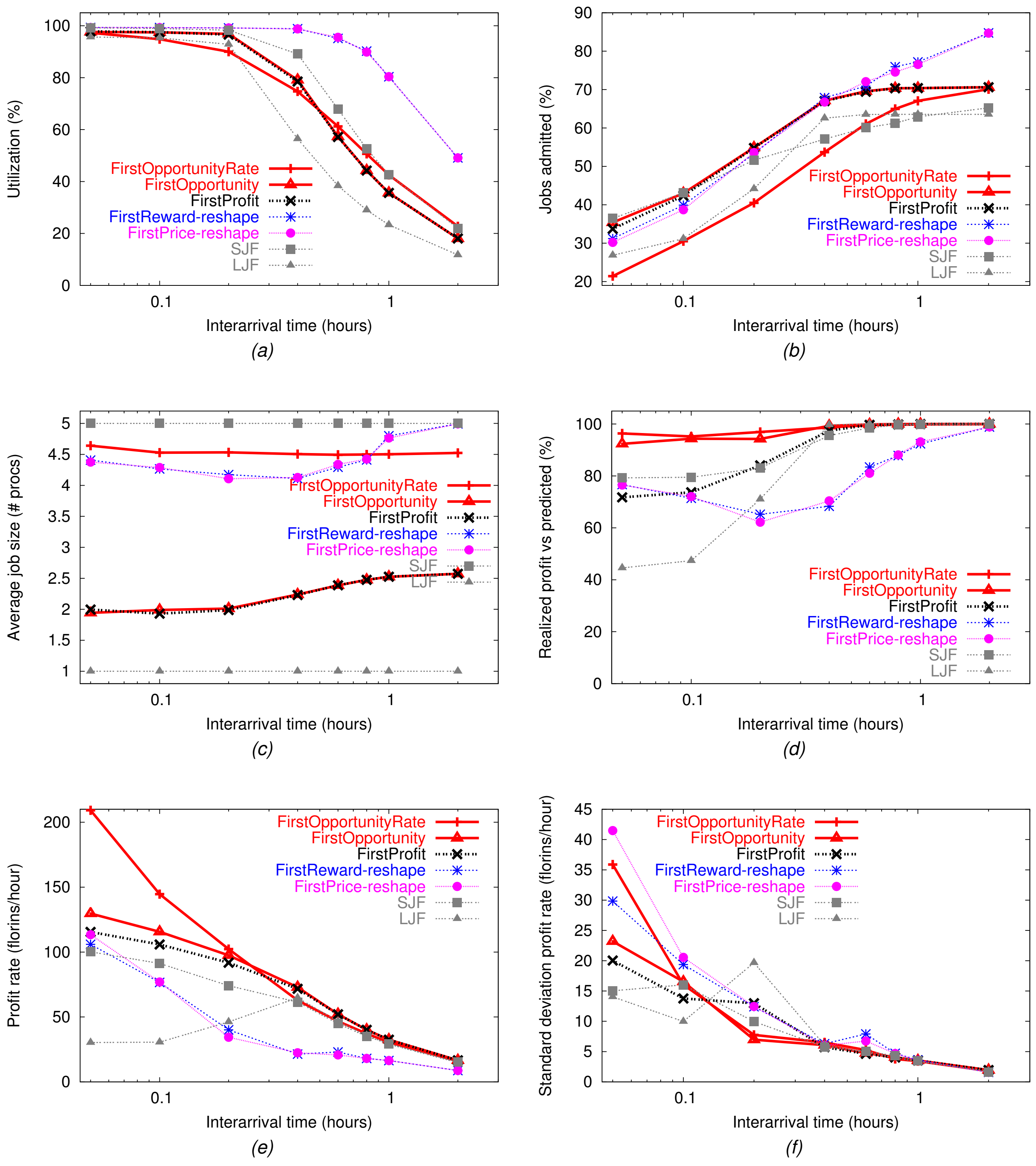

Figure 3. The effect of changing load, with no resource uncertainty. *(a) utilization, the portion of the available resources that are used; (b) the percentage of offered jobs that are accepted by the admission-control algorithm; (c) the average job size (number of processors); (d) percentage of the predicted profit that was realized; (e) the profit rate; and (f) standard deviation of the profit rate. Larger inter-arrival times correspond to lower loads.*

more jobs are arriving. All graphs present averages over 10 runs. The default parameter-settings for our runs are shown in Table 1.

5 Results: scheduling with accurate information

We now present simulation results for the scheduling and admission algorithms described above. We begin by exploring how things worked in the absence of resource uncertainty; the next section presents results that include it.

5.1 Varying the offered load

Figure 3 shows the results of running different schedulers as the inter-arrival time (load) changes. As expected, most schedulers see close to full resource utilization (about 95%) at high loads (Figure 3a),[3] and the percentage of jobs accepted also declines as the load increases (Figure 3b) from a maximum of about 70%, which reflects the fraction of the offered jobs in this workload that can be run profitably. FirstReward and FirstPrice do not consider resource costs, so they accept more jobs at low load, even if they are not profitable.

The graph of the average number of processors used by a job (job shape) as a function of load (Figure 3c) shows that the new profit-based schedulers (FirstProfit and FirstOpportunity) consistently choose roughly half as many processors per job as the cost-oblivious schedulers – less than all except LJF, which is forced to use only one processor; FirstOpportunityRate uses more resources, in order to get more profit per unit time.

Figure 3d shows how well the admission-control algorithm estimates the profit for the jobs it admits. FirstOpportunity and FirstOpportunityRate are the best because they take account of the new job on the already-accepted ones; LJF shows wide variation in behavior.

Figure 3e shows that higher loads increase the profit rate, as schedulers have more jobs to choose from. The new profit-aware schedulers do generally better, with FirstOpportunityRate performing better, since it is more selective in accepting high value jobs (Figure 3b), and more efficient in optimizing for profit-rate.

Figure 3f shows the standard deviation of the profit rate. At high loads and profit rates the variation increases. FirstReward and FirstPrice had profit rates coming close to FirstProfit, but they exhibit higher variation in the profit.

5.2 Varying user impatience: the value-decay rate

We can approximate the effect of impatient users by increasing the rate at which job values decline over time. Figure 4 shows the results: as the decay-rate increases, the number of resources per job increases as the schedulers try to finish work more quickly, and the profit rate drops, as jobs become harder to run profitably. This is reflected in the utilization (not shown), which decreases to close to 0 for decays larger than 100 florins/hour from more than 95% for decays smaller than 10 florins/hour.

The accuracy of profit prediction ((Figure 4b) is consistent with previous observations. The better predictions for high decay rates occur because of the low accepted load, which results in fewer changes to scheduling decisions that have already been made.

In almost all cases, the new schedulers outperform the older ones (Figure 4c), with FirstOpportunity and FirstOpportunityRate having better performance. The standard deviation of the profit rate (Figure 4d) shows erratic behavior, even when determined from 10 runs. The new schedulers tend to do slightly better than the others.

5.3 Varying resource availability

The service provider always fully uses the resources it rents; Figure 5c shows that all schedulers do better when there are more of them to rent, as more jobs can be run successfully, but that the new schedulers generally outperform the others. Being both value- and cost-conscious, the profit-based schedulers find profitable operating regimes. The profit based ones stabilize to use around 2.5 processors per job, while the profit-rate one uses more (around 4.5) as it prefers high value jobs with shorter running time. At high resource counts, the system is under-loaded and there is no more profitable work to accept, so the lines flatten out. We see the same variability in the profit rate (Figure 5d), as in previous experiments.

As the number of resources decreases, the effective load increases, and thus the accuracy of profit prediction declines (Figure 5b), as expected.

5.4 Varying resource price

Figure 6 shows how profitability and average job shape change when the rental price per processor hour is varied. As the price increases, the cost-aware schedulers use smaller job shapes, and consistently deliver greater profitability than the alternatives. The other schedulers do not adapt the job shapes, as they do not consider price in their decisions.

The profit rate (Figure 6c) curves show a rather surprising trend: profit rate sometimes decreases as resources get cheaper. This behavior results from admission-control algorithms that base decisions only on profit, rather than on profit-rate (i.e., that do not take the makespan of the load into account). To confirm this hypothesis, look at the line for FirstOpportunityRate, which optimizes explicitly for profit rate. This does not suffer this behavior at low

[3]Resource utilization here means the fraction of rentable resources that were rented, rather than the percentage of the resources rented that were successfully exploited by the application: while the service provider is renting a resource, we assume that it is fully used.

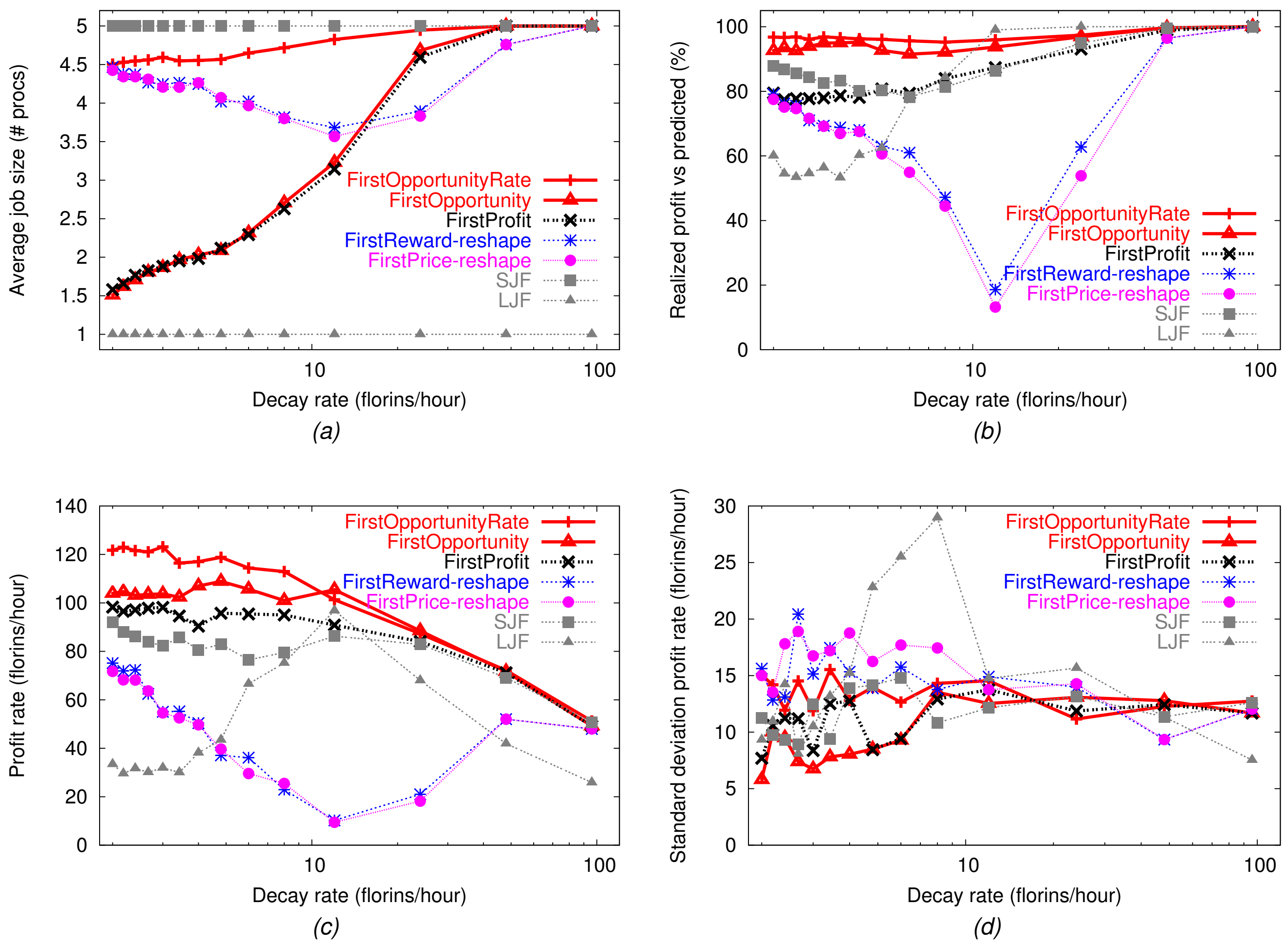

Figure 4. The effect of user impatience, with no resource uncertainty. *(a) the average job size (number of processors); (b) percentage of the predicted profit that was realized; (c) the delivered profit rate; and (d) standard deviation of profit rate. User impatience is expressed as a value-decay or penalty rate in florins/hour.*

resource prices, although it seems to be unnecessarily pessimistic at high ones.

Figure 6b is consistent with previous experiments: as prices drop, the accepted load increases, and the accuracy of profit prediction declines. FirstPrice and FirstReward schedulers start to operate in a negative profit regime when the price increases to more than 8 florins/hour.

6 Results: scheduling with resource uncertainty

The previous section described our exploration of how the service provider algorithms operated without any resource-availability uncertainty. This section relaxes that restriction, and explores the tradeoff between risk and reward when the service provider cannot be sure that resources will be available. Note that with a normally-distributed resource-availability profile, there is always a possibility that zero resources will be available, so a completely risk-averse admission-control algorithm would refuse to accept *any* jobs.

The uncertainty-oblivious policies simply ignored any projected variance in the resource estimate, and assumed that its mean was the actual value. Rather generously, we allowed the uncertainty-oblivious schedulers to reshape a job if it could not fit at attempted-execution time, rather than forcing it to be delayed until a later time.

The uncertainty-aware scheduling policy we examined was FirstProfit, with the admission policy extended to accept jobs only if the probability of achieving the expected profit was greater than $(1 - risk)$. It achieved this by cal-

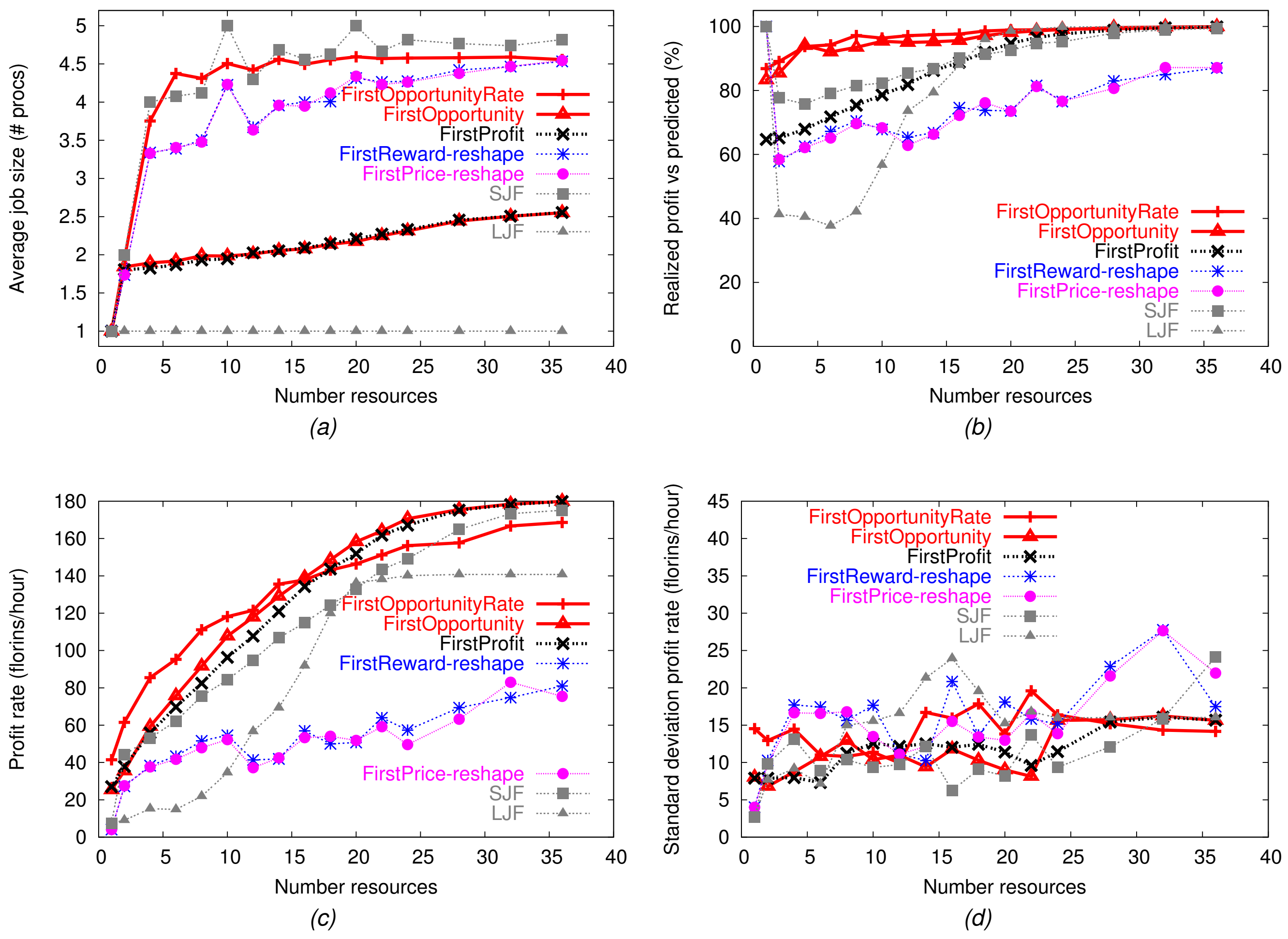

Figure 5. The effect of changing available resources, with no resource uncertainty. *(a) the average job size (number of processors); (b) percentage of the predicted profit that was realized; (c) the delivered profit rate; and (d) standard deviation of profit rate.*

culating the profit for a schedule, and then calculating the probability of being able to achieve that schedule, using the uncertainty for the resource estimates.

Figure 7a shows that the uncertainty-aware schedulers admit fewer jobs as the resource-uncertainty increases to reduce risk, while the others do not.

The number of resources used per job decreases slightly as the uncertainty increases (Figure 7b), because the absence of resources forces schedulers to reshape jobs to fewer processors than initially scheduled.

As the uncertainty increases, the profit rate decreases (Figure 7c); interestingly, the decrease is larger for the uncertainty-aware schedulers than for the oblivious ones.

The variance in expected profit (Figure 7d) increases with uncertainty, and with willingness to take risks; it is generally lower for the uncertainty-aware schedulers. We saw cases with negative profit with the uncertainty-oblivious schedulers at high uncertainty, but did not see this for the uncertainty-aware ones.

In general, increasing profit rate comes with higher profit-rate variance at higher levels of uncertainty – that is, there is a direct trade-off between these two metrics. The risk-aware schedulers allow this tradeoff to be made explicitly.

The uncertainty-aware schedulers generate better profit rates than almost all the uncertainty-oblivious ones. The main exception is the SJF policy, which has higher average profitability at higher resource uncertainty—but this improvement comes with greater profit variability. We observed a similar phenomenon with the LJF scheduler, which

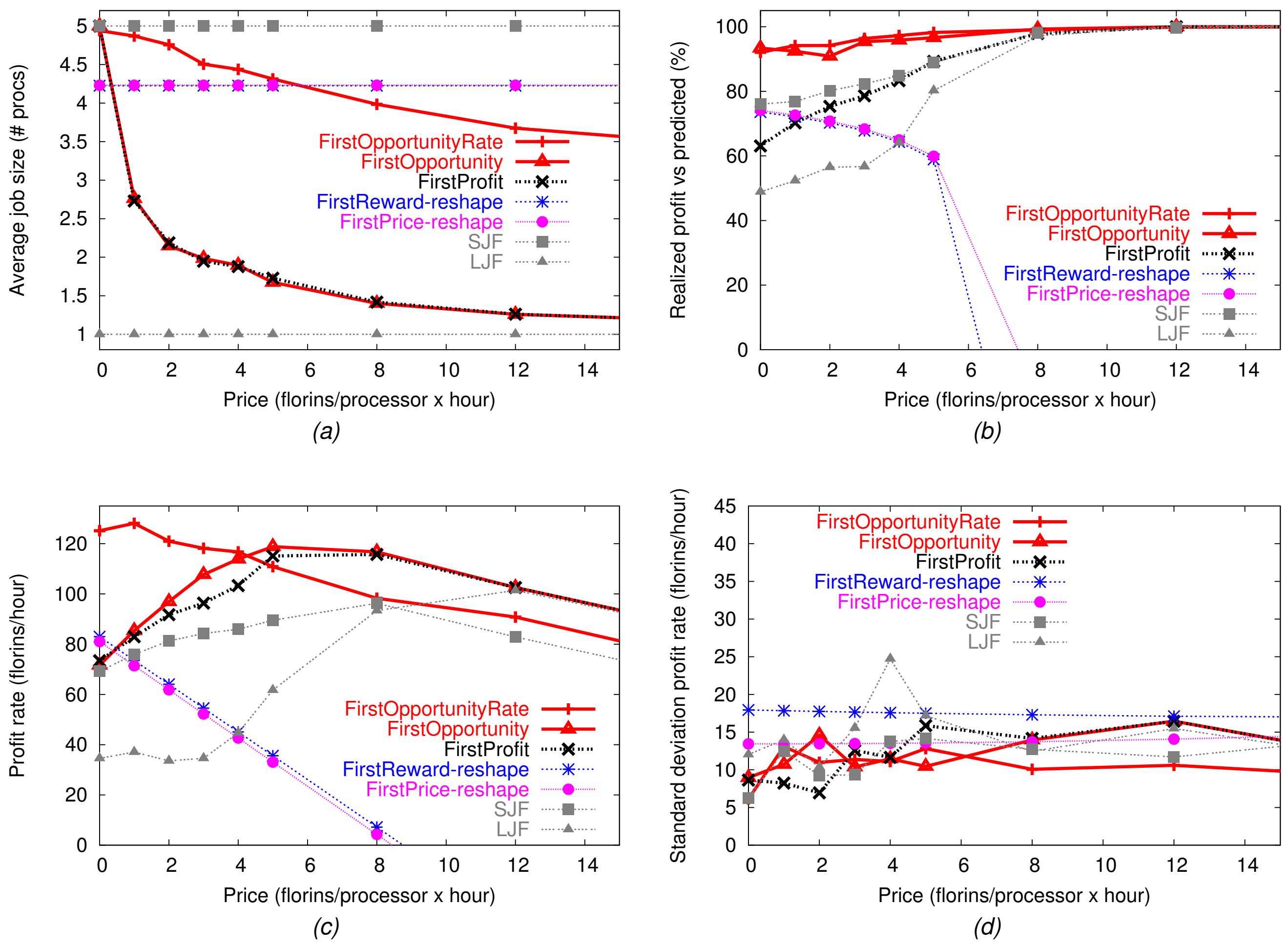

Figure 6. The effect of changing resource prices, with no resource uncertainty. *(a) the average job size (number of processors); (b) percentage of the predicted profit that was realized; (c) the delivered profit rate; and (d) standard deviation of profit rate.*

does poorly at the load used for Figure 7, but better at some other loads.

Setting the acceptable-risk to too low a level has the effect of making the service provider too conservative. We notice this in the case of the FirstProfit(5%) scheduler: the profit rate decreases significantly as uncertainty increases, though the benefit is that the profit variability is almost zero.

7 Conclusions

We investigated the job-scheduling design space for service providers that rent resources rather than own them, and extended this to the case with resource uncertainty, with particular attention to profit-aware schedulers. We studied how load, user impatience, number of resources, price, and resource uncertainty influence the service provider's profit and showed that the new profit-aware schedulers outperform previous ones across a wide range of conditions.

Based on our experiments, we make the following additional observations:

- Admission control is crucial in an over-subscribed environment, and matching the admission-control policy to the scheduling policy is vitally important. This is a particular case of the need for accurate predictions of scheduling behavior when maximizing profitability.
- In this environment, SJF did significantly better than FirstPrice and FirstReward.
- Picking the job-shape causes additional complexity, but this can be handled by reasonably simple scheduler extensions.

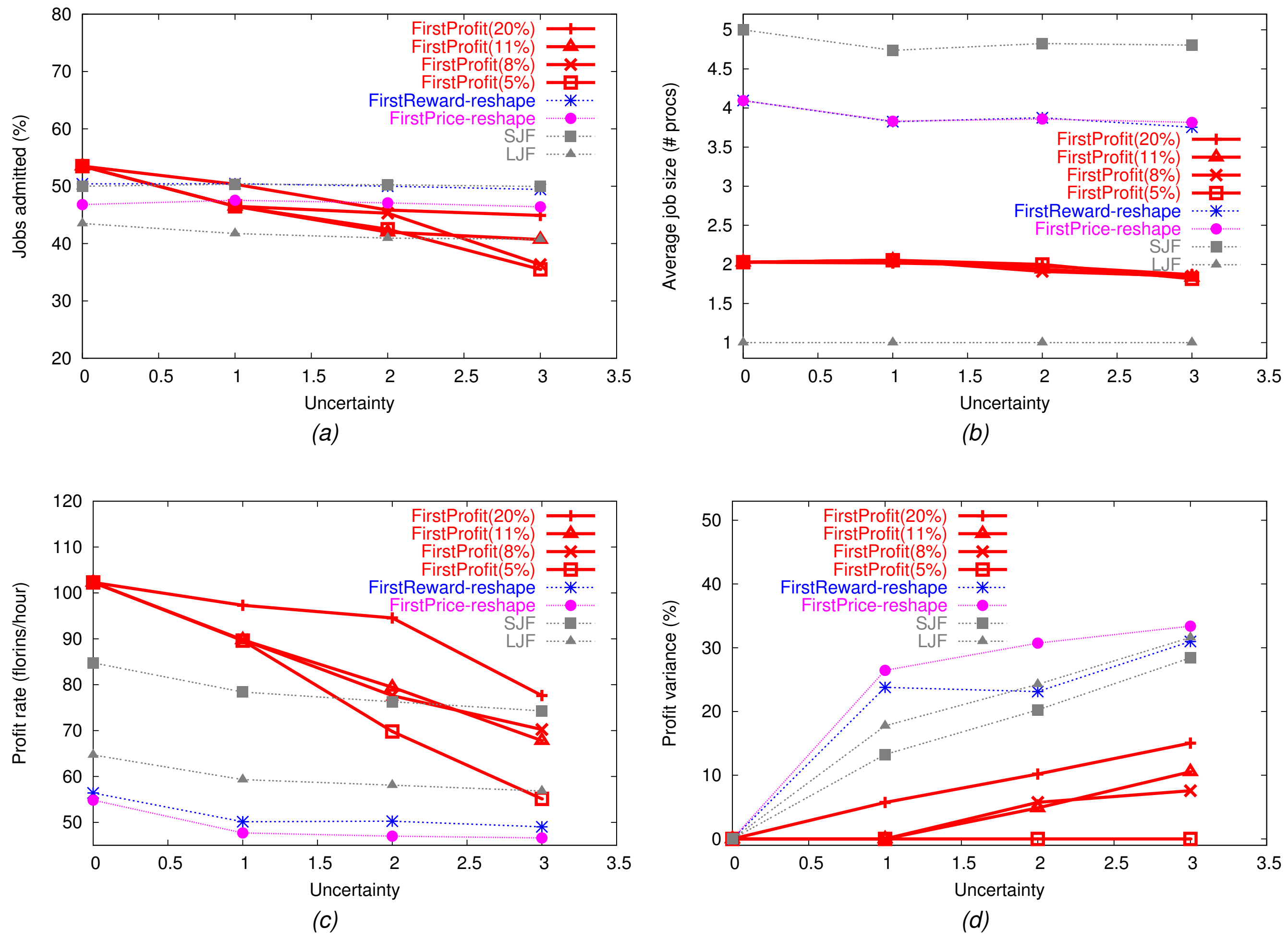

Figure 7. Scheduler decisions when resource uncertainty varies. *(a) percentage of jobs admitted; (b) average number of processors per job; (c) average profit rate; and (d) observed variance in profit rate. In all graphs, uncertainty is measured as the standard deviation of the resource-availability prediction, and each point is derived from 10 runs.*

- Not only did the uncertainty-aware admission control and scheduler algorithms do well in the face of resource uncertainty, but they did no worse than the uncertainty-oblivious alternatives in its absence.
- Our new algorithms offer a controllable tradeoff between risk and potential reward – and in the uncertainty in the size of that reward.
- A larger profit can be obtained at the cost of greater profit uncertainty, but accepting a small amount of risk gives access to most of the available profit.

Our results demonstrate clear benefits for profit-aware schedulers, including realistic situations where profit was double that obtained using previously-published algorithms. Although FirstOpportunity and FirstOpportunityRate performed slightly better than FirstProfit, we believe that the simplicity and low running costs of the latter probably makes it a preferable choice.

In the future, we would like to experiment with non-technical aspects of the economics-based approach, such as determining how to help customers express utility functions easily. We would also like to explore alternative ways to describe and manage risk. For example, it might be beneficial to accept a higher risk for high-profit jobs, and to support resource providers whose resource-price is a function of the level of certainty they offer. It would also be interesting to consider how best to extend these ideas to handle load-related uncertainty, especially if there was partial predictability, such as a cyclic load [Rolia2004].

7.1 Acknowledgments

We would like to thank our HP Labs colleagues (in particular Eric Anderson and Marsha Duro) for their support, and our partners at DreamWorks [Beckett2004], who let us run many early simulations in their portion of the HP compute cluster.

References

[Amdahl1967] Gene Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. *Proceedings of AFIPS*, volume 30, pages 483–5, 1967.

[Beckett2004] Jamie Beckett. HP Labs goes Hollywood: researchers help bring "Shrek 2" to life. HP Laboratories, Palo Alto, CA, April 2004. Web page.

[Chen1996e] K. Chen and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-time Systems*, **10**(3):293–312, May 1996.

[Chiang2001] Su-Hui Chiang and Mary Vernon. Characteristics of a large shared memory production workload. *7th Annual Workshop on Job Scheduling Strategies for Parallel Processing* (Cambridge, MA), volume 2221 of Lecture Notes in Computer Science. Springer Verlag, Berlin, June 2001.

[Chun2002] Brent N. Chun and David E. Culler (UCB.). User-centric performance analysis of market-based cluster batch schedulers. *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid* (Berlin, Germany), May 2002.

[Clark1993] Raymond K. Clark, E. Douglas Jensen, and Franklin D. Reynolds. An architectural overview of the Alpha real-time distributed kernel. *Winter USENIX Technical Conference*, pages 127–46, April 1993.

[He2004] L. He, S. A. Jarvis, D. P. Spooner, X. Chen, and G. R. Nudd. Dynamic, hybrid performance-oriented scheduling of moldable jobs with QoS demands in multiclusters and grids. *3rd International Conference on Grid and Cooperative Computing (GCC 2004)* (Wuhan, China), October 2004.

[Irwin2004] David E. Irwin, Laura E. Grit, and Jeffrey S. Chase. Balancing risk and reward in a market-based task service. *Proceedings of 13th IEEE Symposium on High Performance Distributed Computing (HPDC)* (Honolulu, HI), June 2004.

[Kleinrock1992] Leonard Kleinrock and Jau-Hsiung Huang. On parallel processing systems: Amdahl's law generalized and some results on optimal design. *IEEE Transactions on Software Engineering*, **18**(5):434–47, May 1992.

[Lee1999b] Chen Lee, John Lehoczky, Dan Siewiorek, Raj Rajkumar, and Jeff Hansen. A scalable solution to the multi-resource QoS problem. *20th IEEE Real-Time Systems Symposium (RTSS'99)*, December 1999.

[Petrou2004] David Petrou, Gregory R. Ganger, and Garth A. Gibson. Cluster scheduling for explicitly speculative tasks. *Proceedings of International Conference on Supercomputing (ICS'04)* (St Malo, France). Association for Computing Machinery, Jun–July 2004.

[Rolia2004] Jerry Rolia, Xiaoyun Zhu, Martin Arlitt, and Artur Andrzejak. Statistical service assurances for applications in utility grid environments. *Performance Evaluation*, **58**(2+3):319–39, November 2004.

[Segall2002] M. D. Segall, Philip J. D. Lindan, M. J. Probert, C. J. Pickard, P. J. Hasnip, S. J. Clark, and M. C. Payne. First-principles simulation: ideas, illustrations and the CASTEP code. *Journal Physics: Condensed Matter*, **14**(11):2717–44, 2002.

[Sherwani2004] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software—Practice and Experience*, **34**(6):573–90, May 2004.

[Squillante1999] M. S. Squillante, D. D. Yao, and L. Zhang. Analysis of job arrival patterns and parallel scheduling performance. *Performance Evaluation*, **36–7**:137–63, August 1999.

[Sun2004] Jiong Sun and Norman M. Sadeh. *Coordinating multi-attribute procurement auctions subject to finite capacity considerations*. Technical report CMU–CS–03–184 and CMU–ISRI–03–105. Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, April 2004.