Scheduling explicitly-speculative tasks

David Petrou, Gregory R. Ganger, Garth A. Gibson

November 2003
CMU-CS-03-204

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Large-scale computing often consists of many speculative tasks to test hypotheses, search for insights, and review potentially finished products. For example, speculative tasks are issued by bioinformaticists comparing DNA *sequences and computer graphics artists adjusting scene properties. This paper promotes a new computing model for shared clusters and grids in which researchers and end-users exploring search spaces disclose sets of speculative tasks, request results as needed, and cancel unfinished tasks if early results suggest no need to continue. Doing so matches natural usage patterns, making users more effective, and also enables a new class of schedulers.*

In simulation, we demonstrate how batchactive schedulers *significantly reduce user-observed response times relative to conventional models in which tasks are requested one at a time (interactively) or requested in batches without specifying which are speculative. Over a range of simulated user behavior, for 20% of our simulations, user-observed response time is at least two times better under a batchactive scheduler, and about 50% better on average. Batchactive schedulers achieve such improvements by segregating tasks into two queues based on whether a task is speculative and scheduling these queues separately. Moreover, we show how user costs can be reduced under an incentive cost model of charging only for tasks whose results are requested.*

Keywords: Operating Systems, Scheduling

1 Introduction

Large-scale computing often consists of many speculative tasks to test hypotheses, search for insights, and review potentially finished products. This work addresses how to reduce or eliminate user-observed response time by prioritizing work that the user is waiting on and wasting fewer resources on speculative tasks that might be canceled. This visible response time is the time that a user actually waits on a result, which is usually less than the time that a speculative task has been in the system. Our target architecture is a shared cluster [2, 16], distributed server, or computational grid [5]. Our deployment plan is to replace or augment the extensible scheduling policies in clustering software such as Condor, Beowulf, Platform LSF, Globus, and the Sun ONE Grid Engine [7, 4, 17, 10, 21].

Imagine a scientist using a shared computing cluster to validate a hypothesis. She issues a list of tasks that could keep the system busy for hours or longer. Tasks listed earlier are to answer pressing questions while those later are more speculative. Early results could cause the scientist to reformulate her line of inquiry; she would then reprioritize tasks, cancel later tasks, issue new tasks. Moreover, the scientist is not always waiting for tasks to complete; she spends minutes to hours studying the output of completed tasks.

This paper promotes a new computing model for such scenarios in which researchers and end-users exploring search spaces disclose sets of speculative tasks, request results as needed, and cancel unfinished tasks if early results suggest no need to continue. We call this the *batchactive model* in contrast to users that interactively submit needed tasks one at a time and users that submit batches of both needed and speculative tasks without labelling which tasks are which.

In the batchactive model are *batchactive schedulers* that order speculative tasks with respect to conventional tasks and to one another toward maximizing human productivity and minimizing resource costs. Batchactive schedulers achieve improvements by segregating tasks into two queues based on whether a task is speculative and scheduling these queues separately, resulting in better response time at lower resources usage. This organization is shown in Figure 1, in contrast to the interactive and batch models.

The approach applies best to domains in which several to a potentially unbounded number of intermediate speculative tasks are submitted and early results are acted on while unfinished tasks

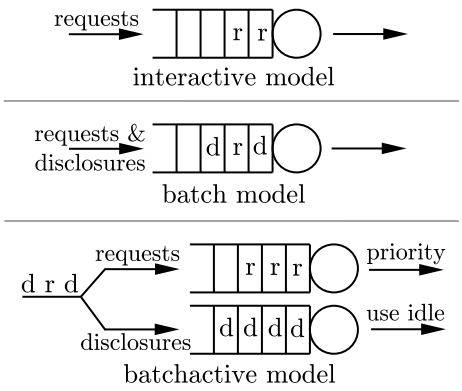

Figure 1: Models of user and scheduler behavior. Users either request needed tasks one at a time (interactive), or both request needed and disclose speculative tasks together (batch and batchactive). In the batchactive case, requested and disclosed tasks are segregated into different queues, in which the requested queue has priority.

remain. Considerable performance improvements are found even when the average depth across users of disclosed speculative tasks is 3 or 4. The following are several highly-applicable domains:

- Bioinformaticists explore biological hypotheses, searching among DNA fragments, using similarity search tools like BLAST [1] and FASTA. These tools differ on whether they are fast and inaccurate or slow and accurate. These scientists share a workstation farm — such as the 30 machines dedicated for such work by the Phylogenomics Group at the University of California at Berkeley — and issue series of fast, inaccurate searches that are followed speculatively by slow, accurate searches to confirm initial findings. The batchactive scheduler would enable scientists to explore ambitious ideas potentially requiring an unbounded amount of resources without fear that resources would be wasted on speculative tasks that might be canceled after early results were scrutinized.

- Teams of artists creating a computer-animated film, such as at Dreamworks or Pixar, submit frames for rendering; each frame, having roughly 50 layers can take up to an hour to render. An artist needs to see intermediate rendered versions of a scene to decide on the finished product. The artist submits the entire scene at once and requests key frames, those with more action, for example. With a batchactive scheduler, the artist would be confident that the scheduler would prioritize the key frames over the speculative, remaining frames. If

upon seeing the output of a key frame the artist changes lighting attributes or moves objects, speculative frames would not have unnecessarily competed against the key frames of other artists on the team.

- Computer scientists routinely use clusters to run simulations exploring high dimensional spaces. Exploratory searches for feature extraction, search, or function optimization, can continue indefinitely, homing in on areas for accuracy or randomly sampling points for coverage. In our department, clusters are used for, among other things, trace-driven simulation for studying microarchitecture, computer virus propagation, and storage patterns related to I/O caching and file access relationships. With a batchactive scheduler, such simulations could occur in parallel with the experimenter analyzing completed results and guiding the search in new directions, with the speculative work operating in the background when requested tasks are queued.

Augmenting a shared scheduler with speculative tasks requires rethinking the metrics and algorithms. In particular, when the system is aware of speculative tasks, the traditional response time metric should be refined. We introduce *visible response time,* the time between requesting and receiving task output, or the time 'blocked on' output. A user only accrues visible response time after requesting a task that may or may not have already been disclosed. In contrast, and less usefully, response time accrues as soon as a task enters the system. Beyond the base value of batchactive scheduling, we think the batchactive model of computing creates an interesting new challenge for scheduling theory and practice: how does one best schedule speculative, abortable tasks to minimize visible response time?

The batchactive model also suggests a new cost model. The cost models at deployed computing centers charge for resource usage irrespective of whether a task was needed [13]. We introduce an incentive cost model which charges for only resources used by tasks whose results are requested. This encourages users to disclose speculative work deeply, and prevents the type of gaming in which users mark all tasks as high-priority, requested tasks. We show that user costs are actually reduced under this cost model.

We show in simulation that a scheduling algorithm should discriminate between speculative and conventional tasks and that such a scheduling algorithm improves on time and resource metrics

compared to traditional schedulers. Over a broad range of simulated user behavior, we show that for 20% of our simulations, visible response time is at least two times better under a batchactive scheduler, and that on average, visible response time is improved by about 50%.

2 User and task model

This section describes how our simulator models users and tasks. While we detail speculation, the reader should keep in mind that the full power of speculation is only leveraged with the batchactive model. In the interactive model, a user only submits one task at a time, and in the batch model, the scheduler does not discriminate between requested (demand-driven) and speculative work. The performance implications are described later.

We model a closed loop of a constant *number of users* interacting with the system. The number of users is a parameter varied across simulations.

Users *disclose* speculative work as *task sets,* which are organized as simple ordered lists of tasks. When a user needs a result from a task set, the user *requests* the task. After some *think time,* the user may request the next task or *cancel* all unrequested tasks and issue a new task set. The act of cancelation models a user that doesn't need any more results from the task set.

Figure 2 depicts this user interaction with the scheduler, and the scheduler's interaction with the computing system. Any number of users disclose, request, and cancel any number of tasks. The scheduling policy decides which and when disclosed and requested tasks run. If a disclosed task is canceled, it is no longer a candidate. The scheduler communicates decisions to the conventional clustering software which handles the details of running tasks on the servers and provides task statistics to the scheduling policy, such as how long a task took to run.

Figure 3 details the states that a task can be in. Each task has a corresponding *service time* and *resource usage* that increases as the task runs. When a task's resource usage equals its service time, the task is considered *executed.* If a task is both executed and requested, then the task is considered *finished* and the task's output is supplied to the requesting user. If a task executes and was disclosed but not requested, then the task's output is stored until requested or canceled.

The probability that a given task's results will cause a user to cancel and issue a new task set is a parameter. This *task set change probability* is modeled by a uniform random variable whose

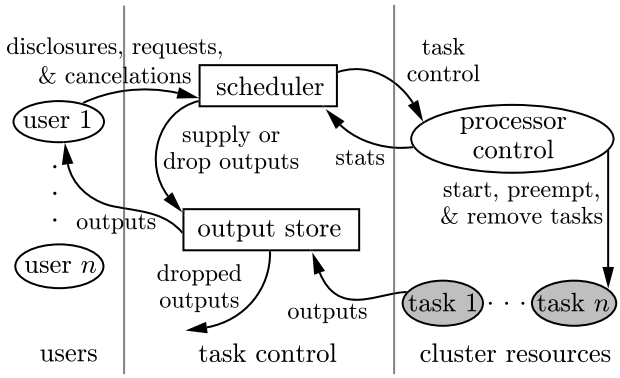

Figure 2: Interaction between users, the scheduler, and the computing center's resources.

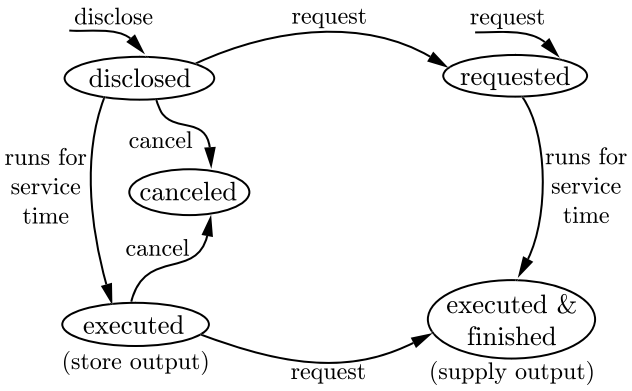

Figure 3: Task state transitions.

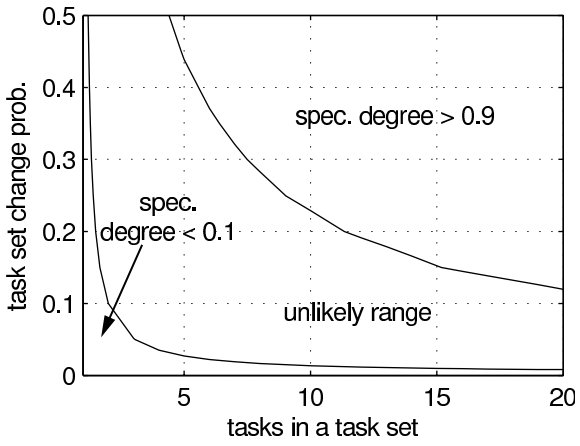

Figure 4: A contour plot of a user's probability of canceling a task set. This degree of speculation is a function of the number of tasks in a task set and the probability that the results of one task causes the user to cancel unrequested tasks. The higher the degree across users, among other considerations, the more a batchactive scheduler is beneficial.

lower bound is always 0 and whose upper bound varies across runs. Each user is assigned a change probability from this distribution for all the user's tasks, so that we simulate users who are more or less certain about whether they will request their disclosed, speculative work.

The number of *tasks per task set* is another simulation parameter and is also drawn from a uniform random variable for each user. Here, the lower bound is always 1, reflecting no disclosure. A low upper bound reflects shallow disclosure; a scientist planning up to five or so experiments ahead. A high upper bound, deep disclosure in the thousands, reflects task sets submitted by an automated process for users searching high-dimensional spaces.

Service time and think time for all tasks, irrespective of user, are varied simulation parameters drawn from the exponential distribution.

One way to reason about speculation is to combine the change probability, p, and number of tasks in a task set, n. The probability that a user will cancel a task set before it completes, which reflects the *degree of speculation,* is $1-(1-p)^{n-1}$. Assuming that users either practice speculation, with a degree of speculation exceeding 0.9, or don't practice speculation, with a degree less than 0.1, then the combinations of task set sizes and task set change probabilities will not be in the middle of Figure 4.

3 Batchactive scheduling

People wish to batch their planning and submission of tasks and pipeline the analysis of finished tasks with the execution of remaining tasks. Conventional interactive and batch models are obstacles to this way of working.

The salient difference between the batchactive and traditional models is that, in the batchactive model, users are free to disclose speculative tasks and requested tasks take precedence over disclosed tasks.

In the batchactive model, contrary to the interactive model, tasks can execute while task output is consumed. Contrary to the batch model, distinguishing between disclosed and requested tasks enables the user to disclose deeply, knowing that the scheduler will give precedence to users waiting on requested tasks. Endowing servers with knowledge of future work in the form of disclosed tasks gives the servers an early start, rather than being idle. Thus, users can observe lower visible response times, making them more productive and less frustrated. The profound effect on scheduling metrics exhibited by scheduling in a batchactive manner is detailed by the simulation results in the evaluation section.

Note that if a user attempts to gain scheduling priority by requesting instead of disclosing speculative tasks, then, according to our cost model, the user will be charged for those speculative resources.

Before discussing our batchactive scheduler, we introduce the metrics that we are interested in and list several restrictions in scope that are in effect.

3.1 Metrics

Mean *visible response time* is our main metric. A task with service time S is requested at time t_r and is executed (completes) at time t_e. Each requested and executed task has a corresponding *visible response time* denoted by

$$V_{\text{resp}} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t_r > t_e, \\ t_e - t_r & \text{if } t_r \leq t_e. \end{cases}$$

Recent work [9, 3, 11] argues that mean slowdown is important to minimize for the following

reasons: (1) slowdown expresses the notion that users are willing to wait longer for larger amounts of work, and (2) mean slowdown often better reflects the performance of most jobs instead of just a few large jobs. Thus, in addition to visible response time, we also study *visible slowdown* denoted by

$$V_{\text{slow}} \stackrel{\text{def}}{=} \frac{V_{\text{resp}}}{S}.$$

Note the differences between these two metrics and the traditional response time and slowdown metrics. Because disclosed tasks can run before they are requested, visible response time can be less than service time and visible slowdown can be less than one. (Of course, assuming $S > 0$, both visible metrics are at least 0.)

We also measure task throughput. Specifically, we measure the number of finished tasks (tasks that were requested and executed) over the duration of our simulation. One of our results is that we improve visible response time without hurting task throughput, and in many cases we improve both visible response time and throughput.

We introduce a metric that reflects the billed resources wasted on disclosed tasks that were never requested. In the interactive and batchactive models, only requested tasks are billed, thus this metric is irrelevant. However, for the batch model, which runs disclosed tasks but does discriminate between them and requested tasks, it is important for this waste to be measured. *Scaled billed resources* is the ratio of the billed resources to the requested resources. For example, if a scheduler charged for ten seconds of resource time but the user only requested five seconds of resource time, then his scaled billed resources will be two. Across all users, we report mean scaled billed resources for the batch model.

For any server, its *load* (also known as device utilization) is the fraction of time that the server was running a task.

For any server, *requested queue length* is the number of tasks in the requested queue. *Disclosed queue length* is defined analogously. We count running tasks as in the queue.

Finally, when comparing schedulers, we sometimes report an *improvement* factor between a new, batchactive scheduler and some baseline. For example, if a metric is better when lower and if the baseline scheduler simulation gave 50 as the metric and the new scheduler simulation gave 25, then the improvement is 2.

We summarize these metrics in Table 1.

metric	description
visible response time	blocked time
visible slowdown	blocked time over service time
task throughput	number of finished tasks
scaled billed resources	billed over requested resources
load	fraction of server busy time
queue length	# of queued and running tasks

Table 1: Metrics reported among schedulers.

3.2 Scope

We make some simplifying assumptions in our evaluation.

Service time is required by SRPT and, thus, we assume that it is predictable with some precision. There exists much work detailing successful predictions of service time from task parameters. Spring, et al. show that the service time of the `complib` biological sequencing library is highly predictable from input size[19]. The service time for the BLAST DNA similarity searcher is dependent on the sizes of the sequences under comparison and the search accuracy. Kapadia et al. use regression to predict the resource requirements of applications over their parameters and these predictions are used for allocating resources on a computational grid [12].

A task can only use one resource significantly. For example, a processor-intensive task might use the disk to load a dataset, but those disk accesses must be an insignificant part of the task's work. Moreover, we assume that preemption costs are low. Thus, for example, tasks with out-of-core memory requirements are not ideal candidates. Further, we assume no complex interactions among tasks, like tasks that contend for shared locks.

3.3 Policies

Any scheduler that behaves differently based on whether a task is requested or disclosed is a batchactive scheduler. Thus, there are many possible batchactive schedulers, such as using any combination of traditional policies for requested and disclosed tasks.

We wish to minimize the mean visible response time across all tasks. Recall that there are

two queues: the queue of requested tasks and the queue of disclosed tasks. Our approach is to give the requested queue priority. That is, when there is a processor available and a requested task available to run, that requested task runs before any available disclosed task. Simply stated, requested (demanded) work is more important than speculative work. Therefore, borrowing a well-known result [8, ch. 8], requested tasks are scheduled with SRPT (shortest-remaining-processing-time).

What remains is to answer how disclosed tasks should be scheduled. The batchactive results in the evaluation section are from a scheduler that runs tasks in the requested queue according to SRPT and tasks in the disclosed queue according to FCFS (first-come-first-serve). We term this scheduler SRPT $\times$ FCFS. The motivation behind FCFS for the disclosed queue is to quickly run tasks that will be requested first. Recall that task sets are ordered lists of disclosed, speculative tasks. Since our simulated users request tasks in this order, then applying FCFS to the task set order is a perfect estimate of request order within one user, and a not completely unreasonable estimate across all users.

Under certain selections of user and task parameters, a different batchactive scheduler SRPT $\times$ SRPT does better than SRPT $\times$ FCFS for mean visible response time. However, for this paper we felt it clearer to only advocate one batchactive scheduler. Of course, when we compare against baseline schedulers, we pick the scheduler that we have found that provides the most competition to SRPT $\times$ FCFS for the metric under examination.

Figure 5 illustrates batchactive scheduling by showing the lengths of the queues of requested and disclosed tasks. The length of the requested queue increases when users request tasks and decreases when these tasks run. The length of the disclosed queue increases when users submit task sets and decreases when tasks in this set execute, are requested, or are canceled. Moreover, disclosed tasks only run when there are no requested tasks available to run on an idle server.

In this paper, we focus on demonstrating our central thesis that segregating requested and disclosed tasks is highly beneficial. Thus, our choice for a disclosed task scheduler is not critical. More refined task schedulers for disclosed tasks is an interesting topic for future research. In particular, there appears to be little existing theory regarding the proper scheduling of speculative, cancellable tasks with respect to visible response time.

We have implemented two schedulers closer to optimal so that we could estimate what im-

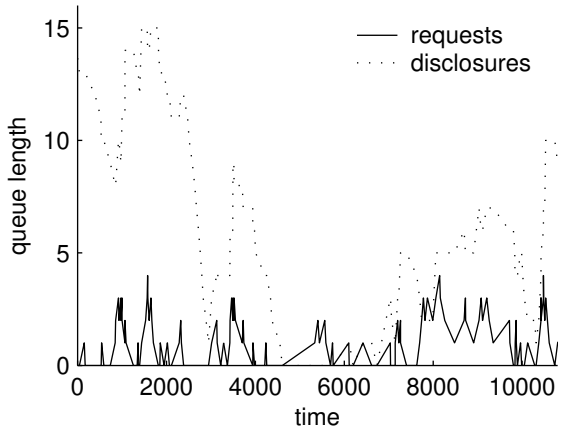

Figure 5: Shown are the lengths of the queues of requested and disclosed tasks over a three-hour simulation of SRPT $\times$ FCFS. Batchactive schedulers only take from the disclosed queue when there are no requested tasks available to run on an idle server.

provements can be had by replacing the disclosed task scheduler with something more sophisticated.

The impractical optimal algorithm for tasks in the disclosed queue selects the task that will eventually be requested with the lowest $t_r + S_{\text{left}}$, where t_r is when the user will request the task and S_{left} is the remaining service time before completion. Intuitively, this algorithm gets the most tasks which will be requested soonest out of the system. Further, the impractical optimal algorithm would ignore disclosed tasks that will never be requested. The reason why this algorithm cannot be implemented is because a scheduler cannot know with complete certainty whether or not a disclosed task will be requested, nor, if so, when it would be requested.

It is difficult to assume task request times for even simulating the impractical optimal algorithm because request times are dependent on when users are in their think times. Instead, we implemented two algorithms, OFCFS and OSRPT, that give a better lower bound on the performance of the impractical optimal algorithm than SRPT $\times$ FCFS and they work like FCFS and SRPT except that disclosed tasks that will never be requested are never run. OFCFS is based on the request time estimate described earlier and ignores remaining service time. OSRPT schedules according to remaining service time without attempting to estimate when a task will be requested. Across most parameters, SRPT $\times$ OFCFS outperforms SRPT $\times$ OSRPT for time metrics, thus we only use this scheduler in our measurements.

model	scheduler
interactive	SRPT
batch	FCFS
batch	SRPT
batchactive	SRPT × FCFS
batchactive	SRPT × OFCFS

Table 2: Evaluated schedulers. There are two batch schedulers under consideration because one outperformed the other under different circumstances. The SRPT × FCFS batchactive scheduler is implementable while SRPT × OFCFS is idealized.

4 Evaluation

We demonstrate that the batchactive user behavior model of disclosing speculative tasks combined with the batchactive SRPT × FCFS scheduler that segregates requested and disclosed tasks into two queues provides better visible response time, visible slowdown, task throughput, and scaled billed resources. These metrics were described in Section 3.1. Evaluation is a scheduling simulation fed the synthetic tasks and user behaviors described in Section 2. The evaluated schedulers are listed in Table 2. Time, such as a service time parameter or visible response time metric, is measured in seconds.

We use simulation in place of an analytic model due to the complexities of our user model which includes task cancelation and which is based on the preceding realistic scenarios.

Our simulation runs explore a variety of user load, the size and how speculative tasks sets are, how big tasks are, and how long users think about task output before making their next requests. Each run ran for two weeks of simulated time after two simulated days of warmup time were ignored. (See Section 4.3.1 for details about the warmup period.)

The choice of simulation parameters is key to arguing for the batchactive computing model. We can make batchactive computing look arbitrarily better than common practice by selecting parameters most appropriate to the batchactive scheduler. However, this would not be a convincing argument. Instead, we have chosen parameter ranges that not only include what we believe to be reasonable uses of speculation for our target applications, but also ranges that include little or no

parameter	range
number of users	4–16
task set change prob.	0.0 to 0.0–0.4 (uni.)
tasks in a task set	1 to 1–19 (uni.)
service time (s)	20–3,620 (exp.)
think time (s)	20–18,020 (exp.)

Table 3: The parameter ranges used in simulating users. The parameters were defined in Section 2. Parameters with parentheses indicate random variables (uniform and exponential). Uniform distributions are described by 'lower bound to upper bound,' where the upper bound is a range.

speculation. We show that under no speculation, we do no worse than conventional models.

4.1 Summarizing results

The parameter ranges listed in Table 3 were used for the results reported in this section unless otherwise specified. For each parameter, we sampled several points in the parameter range. All told, each of the five schedulers were evaluated against 3,168 selections of parameters.

We simulate one server, and restrict the maximum number of users to 16 concurrently active users that are issuing task sets into the server. Further, we make the think time range up to about five hours while the service time reaches only about one hour. Higher think times are better for batchactive scheduling. The probability that after each task the rest of the task set is canceled ranges from users that always need their disclosed speculative work to those who cancel task sets 40% of the time after receiving a requested task's output and thinking about it. Higher task set change probabilities is better for batchactive scheduling. Tasks per task sets range from no disclosure to about twenty disclosures deep, reflecting users using their domain-specific experience to manually plan small to medium-sized task sets. Again, higher values are better for batchactive scheduling.

Service times, which vary from a third of a minute to about one hour are modeled after a BLAST DNA similarity search run. This application usually takes tens of minutes, but service time can vary by significantly based on the sizes of the sequences under comparison and how accurate the search is, which is determined by different convergence algorithms. Think times, which vary from a third of a minute to roughly five hours, were selected to reflect a user who can make a quick

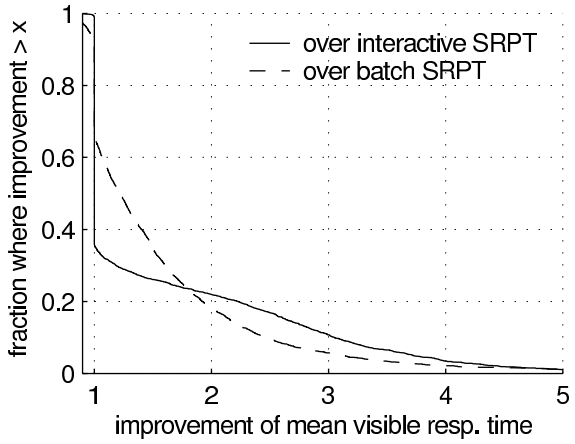

Figure 6: Cumulative factor improvement of SRPT × FCFS over interactive and batch SRPT for visible response time. Against both schedulers, batchactive performs at least twice as better for about 20% of the simulated behaviors. The mean improvement is 1.525 over interactive and 1.537 over batch.

decision about a task's output to one that needs to graph, ponder, or discuss results with colleagues before deciding to request the next task or cancel and start a new task set.

The first set of results summarize the improvement of using the batchactive scheduler SRPT × FCFS over other schedulers for several metrics. The figures are cumulative improvement graphs that show the fraction of runs in which performance was at least a certain factor better than the baseline. The x-axes are factor improvements and the y-axes are the metrics. For example, in Figure 6, the solid line intersection with the x-axis of 3 says that in 10% of all simulation parameters, the ratio of the mean visible response time for interactive SRPT to the mean visible response time for SRPT × FCFS was at least 3. Because the improvements are not Gaussian, these graphs reflect the improvement distribution better than reporting the mean and standard deviation.

As an overall summary of visible response time in the batchactive model, Figure 6 shows how SRPT × FCFS compares to interactive SRPT and batch SRPT. SRPT × FCFS and interactive SRPT perform the same for about 65% of the runs, while SRPT × FCFS does better than batch SRPT for many more situations. However, there are more situations in which the batchactive scheduler is between two and four times better than the interactive scheduler. Against both schedulers, batchactive performs at least two times better for about 20% of the simulated behaviors. The mean improvement is 1.525 over interactive and 1.537 over batch.

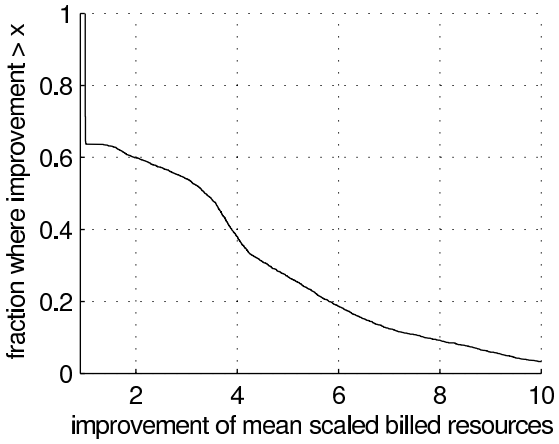

Figure 7: Cumulative factor improvement of SRPT × FCFS over batch FCFS for mean scaled billed resources. The batchactive scheduler charges much less per resource than batch because under batchactive, only requested tasks are charged. Batchactive charges at least four times less for about 40% of the runs. The mean improvement is 3.647.

We now turn to resource cost. The interactive and batchactive models only charge for requested tasks. However, the batch model runs both requested tasks and disclosed tasks that will never be requested. Since the batch scheduler does not discriminate between requested and disclosed tasks, it must charge for them all. We compare batchactive with batch in Figure 7 for scaled billed resources (the ratio of the billed resources to the requested resources) and show that the cost for using resources is considerably lower under the batchactive model. Batchactive charges at least four times less for about 40% of the runs. The mean improvement is 3.647. Recall that in the batchactive model, only requested resources are charged and that disclosed tasks are only run when the server would otherwise be idle. Enabling the user to run disclosed tasks that are never requested for free encourages users to disclose deeply, which provides the other scheduling benefits in this section.

We expect that a batchactive model can provide more total billed resources over the same time period compared to the interactive model. Both models charge only for requested resources. But since the batchactive model provides better task throughput, there can be more requested tasks completed in the same time.

We then set SRPT × FCFS to be the baseline scheduler and compare the oracle-like SRPT × OFCFS against it for visible response time in Figure 8. We see that SRPT × FCFS, although very simple, performs similarly to a scheduler that knows not to run disclosed tasks that won't

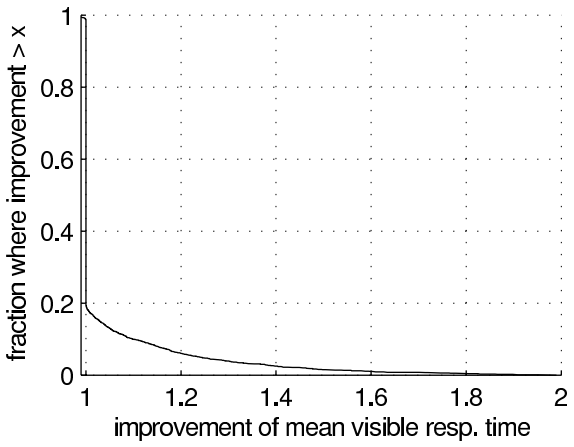

Figure 8: Cumulative factor improvement of SRPT × OFCFS over SRPT × FCFS for mean visible response time. This shows that SRPT × FCFS performs close to a scheduler that ignores disclosed tasks that won't be requested. Yet there is potential for improvement for approximately 20% of the runs.

be requested. Note, however, that the upper bound of batchactive performance is unknown; SRPT × OFCFS does not take into account task size nor does it know when a task will be requested. Designing superior schedulers for disclosed tasks will be an interesting area for future research.

For less than 10% the of runs (sometimes less than 0.1%, and sometimes never), batchactive did roughly 10% worse than the baseline schedulers. Error is introduced due to variations in the number of finished tasks across runs: While all simulations run for the same amount of virtual time, during that time different schedulers complete different numbers of tasks. Thus, the mean of visible response time among two different schedulers running with the user and task parameters are often comprised of a different number of finished tasks. Although we do not have confidence intervals across all of our data, a look at the confidence intervals for a small subset in Section 4.3.2 suggest that for all the cases in which batchactive's mean is worse, the 95% confidence intervals between batchactive and baseline scheduler overlap.

4.2 Per-parameter investigations

At this point we provide a closer look at what affects performance. The following data consists of slices of the scheduling models, user behaviors, and metrics.

parameter	setting
number of users	8
task set change prob.	0.0 to 0.2 (uni.)
tasks in a task set	1 to 15 (uni.)
service time (s)	600 (exp.)
think time (s)	6000 (exp.)

Table 4: The base parameters used in comparisons between schedulers. For each run, all save one of these parameters was held constant at the listed values. Parameters with parentheses indicate random variables. Uniform distributions are described by 'lower to upper bound.'

The graphs in this section have the same format. Each parameter point is associated with a set of three bars. The leftmost bar is the batch model, the middle bar is the interactive model, and the rightmost bar is the batchactive model. The goal is for the rightmost bar to be lowest (or highest, depending on the metric) as much as possible. Each graph varies one parameter. The parameters that are not varied are taken from the base parameters listed in Table 4 unless otherwise noted.

Figure 9 demonstrates the effect of varying the number of users on visible response time. Batch is suited to a small number of users because execution time and think time can be pipelined, and the load is sufficiently low that one's disclosed but never requested tasks won't interfere with the requested tasks of others. Interactive is more suited to a high number of users, in which the load is higher, because the server is always busy with requested tasks. Even though these models alternate on which is best suited to which scenario, in nearly every case, batchactive performs best, exhibiting adaptability.

Batchactive is better than batch under low numbers of users because requested tasks never wait for disclosed tasks; it is better than interactive at the busier end because it consumes any otherwise idle time by executing disclosed tasks. At the busiest part, interactive and batchactive are equivalent, because the requested task queue is never empty.

Figure 10 looks at how the number of users affects mean visible slowdown. Recall that visible slowdown can be less than one (Section 3.1). Batchactive still outperforms the alternatives. The difference compared to visible response time is that visible slowdown equalizes batch and interactive when there are many users. This occurs because the visible response time of the large jobs

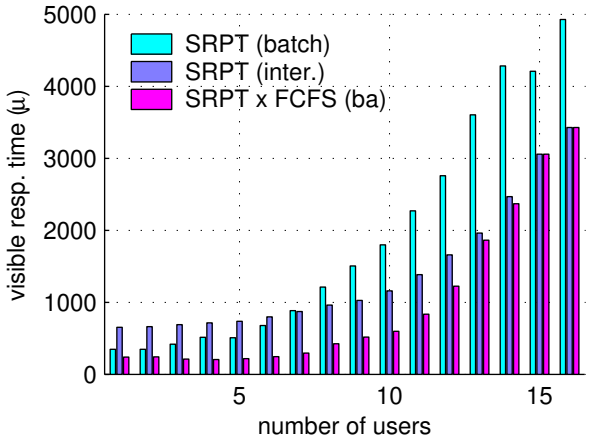

Figure 9: The effect of varying the number of users while holding other parameters constant on visible response time. At low numbers of users, interactive is better than batch, which at high numbers, batch is better than interactive. In nearly all cases, SRPT $\times$ FCFS wins, exhibiting adaptability.

have less of an effect on the visible slowdown.

Figure 11 looks at how the number of users affects task throughput (the number of finished tasks) over the two weeks of simulated time. Batchactive is able to finish more tasks while providing the better visible response times and visible slowdowns shown in Figures 9 and 10. Only toward the highest number of users does the throughput of interactive meet, but not exceed, batchactive.

Additional insight into batchactive scheduling is found by comparing Figures 9, 10, and 11 with Figure 12, which is the same run except that load is plotted. The load under the batch and batchactive models are similar. As their loads increase, batchactive does better than batch with respect to visible response time because batchactive favors requested tasks. Batchactive does better than interactive with respect to visible response time and slowdown because batchactive pipelines disclosed tasks with think time. Counterintuitively, when batchactive's load is higher than interactive, it performs better than interactive. What matters isn't merely load, but the fraction of load made up of tasks that have been or will be requested. Only when the batchactive and interactive loads approach one (near 14 users), do their visible response times slowdowns begin to match.

In Figure 13, we vary the upper bound of the task set change probability from 0.0–0.4. As expected, this parameter has no significant effect on the interactive model, since it does not submit disclosed tasks. We notice a greater dependence on this parameter by the batch model compared to the batchactive model because the batchactive model avoids speculative tasks when requested

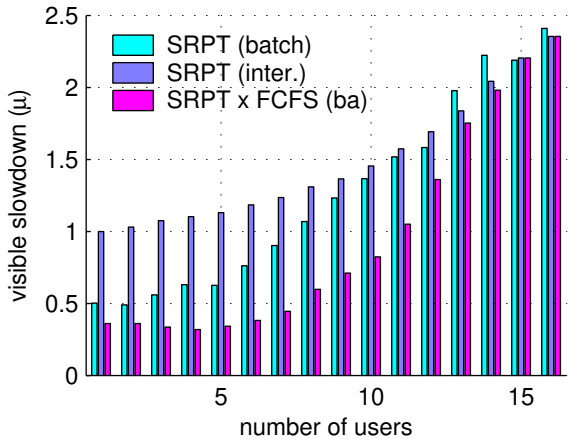

Figure 10: The effect of varying the number of users on visible slowdown, holding other parameters constant. Note: visible slowdown can be under one.

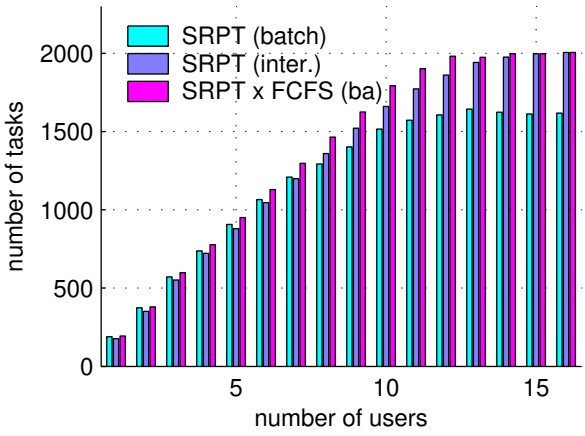

Figure 11: The effect of varying the number of users on task throughput (number of finished tasks) over the simulation run, holding other parameters constant. Batchactive provides the best or equals the best task throughput.

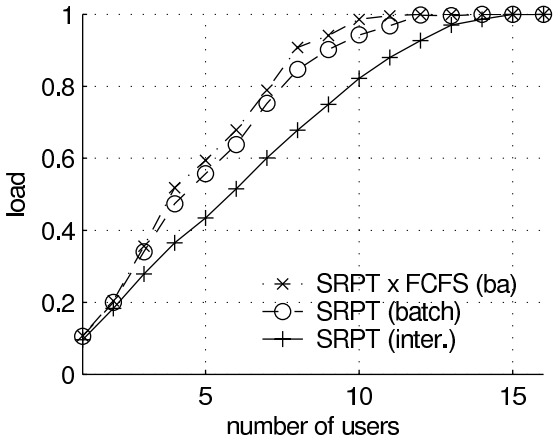

Figure 12: The effect of varying the number of users on load (utilization). This graph illustrates that load in itself does not fully convey batchactive behavior: what matters is the fraction of load that is made up of tasks that have been or will be requested.

work remains.

Now we show the effect of large task sets, such as those submitted by an automated process reflecting users searching high-dimensional spaces. We varied the upper bound of the tasks per task set uniform distribution from 1–1024 in multiples of 2. We also set the task set change probability to 0.1 so that task sets are not canceled as often.

The results with respect to visible slowdown are in Figure 14. When all task sets have only one task, then all models provide the same visible slowdown. Task sets as small as several tasks, which is easily realizable by users performing exploratory searches, provide good improvement over interactive. As the task set size increases, both batch and batchactive provide visible slowdowns less than one. This occurs because there is now user think time that can be leveraged to run disclosed tasks. Soon batch performance becomes unusable as its single queue is overwhelmed with speculative tasks. The interactive model is immune to the task set size because each user will only have one task (a requested task) in the system at any time.

In Figure 15, we vary service time to see its effect on visible response time. As the service time increases, the load increases and the performance of interactive and batchactive become the same. As a limiting case, this shows that when a server is always running requested work, the batchactive model does not help performance.

The remaining parameter is think time. It works inversely to service time: the more think time,

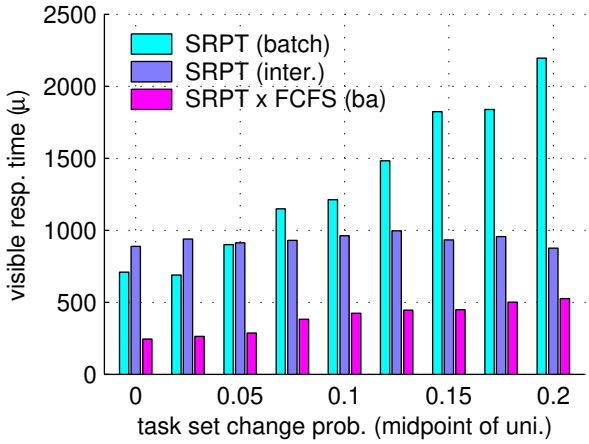

Figure 13: The effect of varying the task set change probability on visible response time. The probability is a uniform random variable and shown on the x-axis is the average of the lower and upper bounds of this distribution. The interactive model is unaffected by this parameter, while the batchactive model is affected less than the batch model.

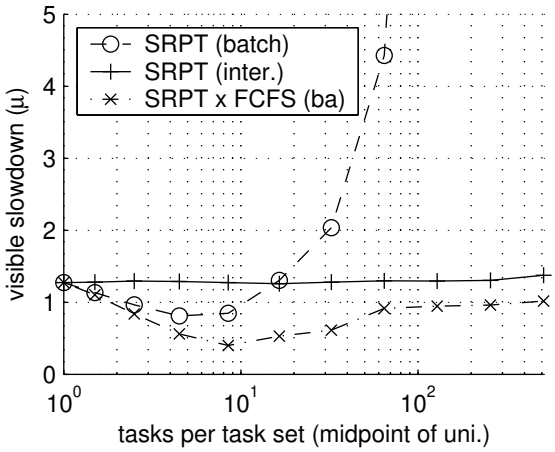

Figure 14: The effect of varying the tasks per task set on visible slowdown. Note that this graph is log-linear. The tasks per task set is a uniform random variable and shown on the x-axis is the average of the lower and upper bounds of this distribution. Batch is unusable at a task set size with an upper bound of 100. (At an upper bound of 1024 tasks per task set, the batch visible slowdown is over 170.)

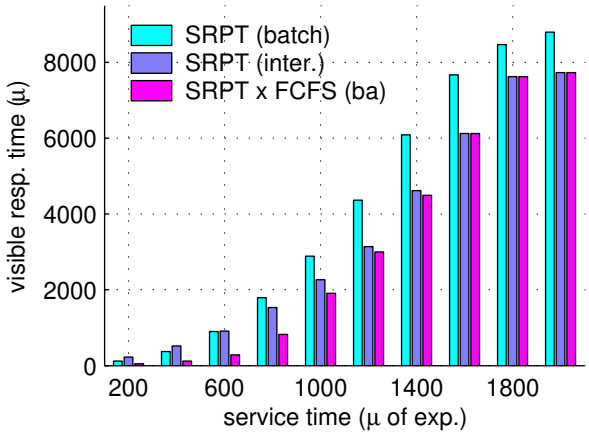

Figure 15: The effect of varying service time visible response time. At low service times, there is more opportunity for the batchactive model to improve performance.

the more opportunity for the batchactive scheduler to order tasks to reduce visible response time. Of course, once the ratio of think time to service time is sufficiently high, which results in a low load, batchactive performs no better than batch. We omit graphs on think time to save space.

4.3 Simulation details

4.3.1 Warmup period

When the simulation starts, all users begin submitting tasks. Only after task results are received do users enter their think times. Thus the queue length near the start of the simulation is not representative of the behavior of the system over steady-state. The most extreme example that we found after looking at several runs is illustrated in Figure 16. We avoid including such warmup-data in our reported metrics by conservatively dropping all data in the first two days of simulation time. Since each run simulates 16 days, our metrics reflect two weeks at steady-state.

4.3.2 Confidence intervals

Varying simulation parameters clearly indicate that the parameters have a significant effect on scheduling metrics and that batchactive scheduling performs best nearly all of the time. To reinforce these observations, we took confidence intervals of mean visible response time for a small interactive SRPT run in which service time was varied in six minute increments while other pa-

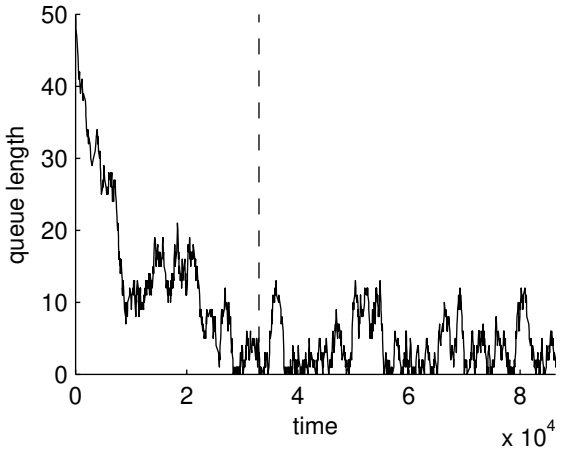

Figure 16: The queue length of requested tasks for an extreme selection of simulation parameters only stabilizes after approximately 10 hours of simulated time. For this reason, all reported metrics ignore the first two days of simulated time.

rameters were held constant (Figure 17). Each reported visible response time mean and confidence interval was the result of 40 simulations that were started with different random seeds. Out of ten selections of service times, only one pair of 95% confidence intervals overlapped, and all confidence intervals were less than 5% of their respective mean visible response times. A normal probability plot (not shown) of the response times for each service time was sufficiently Gaussian to suggest that the confidence intervals are reliable.

4.3.3 Implementation

Our scheduler simulator was written in C and runs on Linux and Darwin. The simulator is broken up into modules that simulate user, task, and server behavior and contains a large number of checks which give us confidence that the results are accurate and that there are no bugs in its operation. One Perl script executes large numbers of this program to explore the simulation parameter space[1] and stores its results into a MySQL relational database. Other scripts query this database, analyze the results in conjunction with MATLAB, and process them for visualization.

All runs were performed on a set of $2.4\,\mathrm{GHz}$ Pentium IV machines each with $512\,\mathrm{MB}$ of memory. Most runs took tens of seconds and less than $100\,\mathrm{MB}$ of memory to simulate 16 days of

[1]Ironically, we found ourselves desiring a computing cluster equipped with a batchactive scheduler during the exploration of our simulator's five-dimensional parameter space.

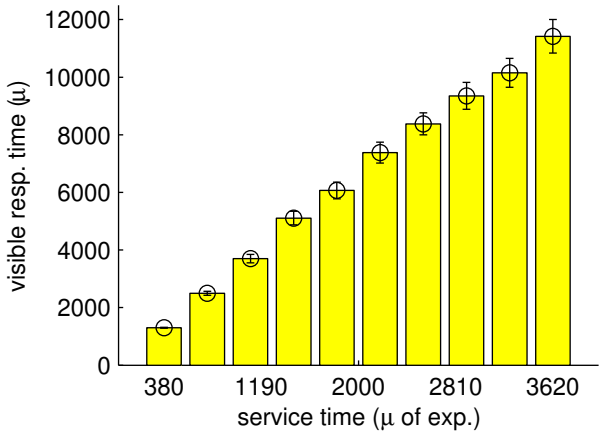

Figure 17: Confidence intervals for a small run suggest that our results are significant. Plotted is the mean visible response time across ten service times. The 95% confidence intervals were from 40 runs using different random seeds at each service time.

virtual time. A small percentage of runs with parameters causing many more tasks to be created took roughly ten minutes to run.

5 Related work

Speculation to improve performance is a pervasive concept across all areas of computing including architecture, languages, and systems. Speculation is found at the level of I/O requests, program blocks, and instructions. We bring this notion to the level of tasks for the processor resource. In this section, we focus on the closest related work in systems whose experience and potential have inspired our work.

In the storage realm, Patterson et al. have shown in the TIP system how application performance can increase if the application discloses storage reads in advance of when data is needed [15]. While their work focussed on storage questions, such as how to balance cache space between prefetches and LRU caches, our work applies the same concepts and terminology to the processor resource at the granularity of tasks.

In the network realm, researchers have sought to discriminate between speculative and requested network transmissions. Padmanabhan et al. have shown a tradeoff in visible response time and fractional increase in network usage when varying the depth of their web prefetcher [14]. In

the approach of Steere et al., people manually construct sets of web prefetch candidates and the browser prefetches as much as three such candidates simultaneously until all are fetched, or until the person initiates new activity [20].

In the database realm, Polyzotis et al. built a speculator that begins work on database queries during the user think time in constructing complex queries. [18]

We have not found a direct analogue of our work for the processor resource. Speculation has been used for process scheduling in the context of estimating task runtime and deciding when to power down low-powered devices to maximize battery power and performance metrics. But we have not found approaches explicitly for scheduling among requested and disclosed tasks.

One way to organize speculative approaches is by whether they require a user or agent working on behalf of the user to disclose speculative work, or whether the system automatically generates speculative work. In our work, Steere's system, and the TIP system, speculative work is generated externally. In the systems of Padmanabhan et al. and Polyzotis et al., the system constructs speculative work in addition to scheduling between disclosures and requests. An extension to TIP by Chang et al. introduces automatic I/O disclosure generation [6]. As opposed to these automatically speculating storage, network, and database examples, it does not seem possible to do the same for the processor resource without domain-specific knowledge or specialized user agents.

6 Conclusion

Ideally, a cluster task scheduler would run speculative tasks while users were analyzing completed tasks, minimizing the blocked time that the users experience. The catch is that speculative tasks will take contended resources from users who are waiting for requested tasks unless the two types of tasks can be discriminated.

The solution we promote is a new computing model of users who disclose sets of speculative tasks, request results as needed, and cancel unfinished tasks if early results suggest no need to continue. We observe that not all tasks are equal — only tasks blocking users matter — leading us to introduce the *visible response time metric* which measures the time between a task being requested and executed, independent of when it was speculatively disclosed. Our *batchactive scheduler* segregates requested and disclosed tasks into two queues, giving priority to the requested

queue, toward minimizing mean visible response time.

We simulated a variety of user and task behaviors and have found that for several important metrics our batchactive model nearly always does better than conventional models in which tasks are requested one at a time (interactively) or requested in batches without specifying which are speculative. We have found that for 20% of the runs, our scheduler performs at least two times better for visible response time. Compared to a batch scheduler, for about 40% of the runs, the average simulated user pays for a fourth of the resources.

Given the benefits of task speculation we have presented, one may wonder why this idea is not widely deployed. We surmise that the enabling technologies have only recently been in place, and that there is an initial learning curve to be surmounted. First, the benefits of speculation require that not all resources are consumed by requested tasks; this is increasingly true in resourceful cluster and grid environments. Second, users must be able to anticipate, with a minimum degree of accuracy, tasks that they might need. These points are related: the better users separate disclosures from requests, the more resources are available to a batchactive scheduler.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–10, 1990.

[2] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[3] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (DA '98)*, pages 270–279, San Francisco, CA, Jan. 1998.

[4] Beowulf.Org — The Beowulf Cluster Site. `http://www.beowulf.org/`, Nov. 2003.

[5] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[6] F. Chang and G. A. Gibson. Automatic I/O hint generation through speculative execution. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 1–14, New Orleans, LA, Feb. 1999.

[7] Condor project homepage. `http://www.cs.wisc.edu/condor/`, May 2003.

[8] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, Reading, MA, 1967.

[9] D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing (IPPS / SPDP '97)*, pages 238–261, Geneva, Switzerland, Apr. 1997. Lecture Notes in Computer Science, vol. 1291, Springer-Verlag.

[10] The Globus toolkit. `http://www-unix.globus.org/toolkit/`, Nov. 2003.

[11] M. Harchol-Balter, K. Sigman, and A. Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. In *IFIP WG 7.3 International Symposium on Computer Modeling, Measurement and Evaluation (Performance '02)*, Rome, Italy, Sept. 2002.

[12] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99)*, pages 47–54, Redondo Beach, CA, Aug. 1999.

[13] The Lemieux Supercomputer. Pittsburgh Supercomputer Center, `http://www.psc.edu/machines/tcs/lemieux.html`, 2003.

[14] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review (CCR '96)*, 26(3):22–36, July 1996.

[15] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*, pages 79–95, Copper Mountain Resort, CO, Dec. 1995.

[16] G. F. Pfister. *In Search of Clusters*. Prentice Hall, Upper Saddle River, NJ, 1995.

[17] Platform Computing — Products — Platform LSF. `http://www.platform.com/products/LSF/`, Nov. 2003.

[18] N. Polyzotis and Y. Ioannidis. Speculative query processing. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR '03)*, Asilomar, CA, Jan. 2003.

[19] N. Spring and R. Wolski. Application level scheduling of gene sequence comparison on metacomputers. In *Proceedings of the 12th ACM International Conference on Supercomputing (SC '98)*, pages 141–148, Melbourne, Australia, July 1998.

[20] D. C. Steere. Exploiting the non-determinism and asynchrony of set iterators to reduce aggregate file I/O latency. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, Saint Malo, France, Oct. 1997.

[21] Sun ONE Grid Engine Software. `http://wwws.sun.com/software/gridware/`, Nov. 2003.