Scheduling speculative tasks in a compute farm

David Petrou
Carnegie Mellon University

Garth A. Gibson
Carnegie Mellon University

Gregory R. Ganger
Carnegie Mellon University

Abstract

Users often behave speculatively, submitting work that initially they do not know is needed. Farm computing often consists of single node speculative tasks issued by, e.g., bioinformaticists comparing DNA sequences and computer graphics artists rendering scenes who wish to reduce their time waiting for needed tasks and the amount they will be charged for unneeded speculation. Existing schedulers are not effective for such behavior. Our 'batchactive' scheduling exploits speculation: users submit explicitly-labeled batches of speculative tasks, interactively request outputs when ready to process them, and cancel tasks found not to be needed. Users are encouraged to participate by a new pricing mechanism charging for only requested tasks no matter what ran.

Over a range of simulated user and task characteristics, we show that: batchactive scheduling improves visible response time — a new metric for speculative domains — by at least 2X for 20% of the simulations; batchactive scheduling supports higher billable load at lower visible response time, encouraging adoption by resource providers; and a batchactive policy favoring users who use more of their speculative tasks provides additional performance and resists a denial-of-service.

1 Introduction

Imagine a scientist using a shared compute farm to validate a hypothesis (Figure 1). She submits chains 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 outputs 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, attends meetings and lunches, and stops working as evening approaches.

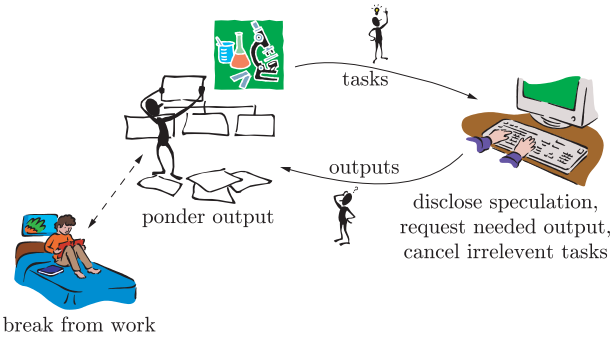

Figure 1: Users wish to pipeline the execution of chains of speculative — not known to be needed — tasks with the consideration of received task outputs and optional rest periods. This paper removes existing barriers to exploiting this way of working.

Existing schedulers do not leverage the speculation inherent in such situations, resulting in a mismatch of goals and suboptimal scheduling. Should a user who does not know which tasks will bear fruit submit one speculative task, a few, many, or every conceivable task? After all, defining tasks is a time-consuming activity in itself. A user wishes to reduce the time waiting for needed task output, increasing the rate at which scientific inquiry is accomplished, and reduce the amount charged for unneeded speculation. The right amount of speculation depends on considerations difficult and burdensome or impossible for a user to know, including to what extent a task is in fact speculative, whether receiving the results of

SC|05 November 12–18, 2005, Seattle, Washington, USA

a speculative task will be worth its cost, and the behavior (queuing) of other users. In situations in which resources are not directly charged or users have the means to pay for wasted work, users might overwhelm resources with speculative tasks in an attempt to reduce delay. This paper addresses these and other concerns with multiuser 'batchactive' schedulers (combining batch and interactive characteristics) to exploit the inherent and easily disclosed speculation common to many application scenarios.

With a batchactive scheduler, users submit explicitly-labeled chains or batches of speculative tasks exploring ambitious lines of inquiry, and users interactively request task outputs when they are found to be needed. Batchactive scheduling segregates tasks into two queues based on whether a task is speculative and gives the non-speculative tasks priority, and employs a novel incentive pricing mechanism that charges for only requested task outputs (i.e., unneeded speculative tasks are not charged). Users are encouraged to disclose speculative tasks because the prioritization provides better time metrics and the pricing mechanism provides better cost metrics. Endowing servers with the knowledge of tasks that may be needed enables servers to get an early start rather than being idle. Further, this knowledge can expose parallelism within a user's workload that can be used to leverage multiple compute nodes when tasks do not depend on outputs from one another. After receiving an output and considering it for some time, a user decides to request more tasks, cancel tasks, or disclose new speculative tasks.

We target loosely connected farms of compute nodes, which we define as collections of individually scheduled processors, as opposed to clusters which may imply tighter coordination. Examples include PCs connected by commodity networks or racks of servers such as the Apple Xserve G5. We use 'farm' to indicate that the workload consists of single-threaded, non-communicating tasks. While copies may run simultaneously across multiple compute nodes to obtain speedups, there is no requirement that any processes be coscheduled. By restricting our scope to such applications, we focus our research on the problems posed by speculation and leave the integration of coscheduling to future work. The number of important single processor tasks is large, as shown by our target applications below. Our scope is also restricted to processor-bound tasks; I/O-bound tasks should be addressed by high-performance parallel I/O architectures [Gibson et al., 1998; Hildebrand and Honeyman, 2004]. Our deployment plan is to augment the scheduling in, e.g., Condor, Platform LSF, Globus, or the Sun ONE Grid Engine.

Noting that not all tasks are equal — only tasks whose outputs users eventually desire matter — we introduce the 'visible response time' metric (the time between needing and receiving task output irrespective of when it was submitted) and batchactive pricing (charging only for tasks whose outputs the user views irrespective of what ran). We specifically show that: (1) speculative tasks are poorly exploited by existing schedulers; (2) speculative task disclosure and batchactive pricing support how people wish to work for many application scenarios, including their desire to pipeline think time and task execution; (3) in a single-server simulation, batchactive scheduling can substantially reduce visible response times (among other time metrics), reduce user costs, and in some cases improve resource provider revenue; and (4) batchactive scheduling is simple (thus, it should be easily deployable) and exhibits low overhead.

For example, over a broad range of simulated user behavior and task characteristics, we show that under a batchactive scheduler visible response time is improved by at least a factor of two for 20% of the simulations. On a non-speculative scheduler, there are extreme situations (such as high load) in which users who submit one task at a time results in better performance than users who unconditionally submit batches of tasks at a time. While at other extremes (such as low load), the opposite is true. But users submitting work to a batchactive scheduler results in as good or better performance for both these extremes and better performance for intermediate situations, exhibiting adaptability. Moreover, visible response time can be improved without decreasing the throughput of tasks whose outputs were desired. User costs decrease while server revenue increases in the reasonable circumstance that lower cost for needed work leads to increased usage. Related is that more users can be supported and greater server revenue generated at the same mean visible response time. Our results may be verified and further research may be conducted using our scheduling simulator (`http://www.pdl.cmu.edu/PDL-FTP/Scheduling/ba_sim-0.1.tar.gz`).

1.1 Targeted applications

Users often submit speculative tasks to test hypotheses, search for insights, or review potentially finished products; called exploratory searches or parameter studies. For example, an application may be run repeatedly with different arguments to search a large parameter space first in broad stokes — randomly or at predetermined intervals (called iterative improvement) — then in detail at areas of interest. Any-time algorithms or imprecise computing generate output after using some amount of resources or after achieving some level of quality. Our system, which schedules activity at the task granularity, considers the creation of each intermediate output to be one task. It is often initially unclear which task outputs will be useful. [DeGroot, 1990] The following are important target applications (consisting of non-communicating tasks) fitting our scheduling approach:

Bioinformatics comprises methods for solving nucleotide sequencing problems. Bioinformaticists explore biological hypotheses, searching among DNA fragments with similarity tools like the non-parallel BLAST [Altschul et al., 1990] tool. Some algorithms are more sensitive to differences than others and the more sensitive ones are slower. Moreover, a single algorithm may have a parameter to control this sensitivity / time tradeoff. These scientists share workstation farms — such as the dedicated 30 machines at the Berkeley Phylogenomics Group [Holliman, 2003] — and issue series of fast, inaccurate searches (taking from 10 sec to 10 min) followed by slow, accurate searches to confirm initial findings. A batchactive scheduler would enable scientists to explore ambitious hypotheses without fear that resources would be wasted on tasks that might be canceled after early results were scrutinized.

Computer animation is increasingly used in motion pictures. Hundreds of artists and engineers creating a film such as at Dreamworks or Pixar submit scenes for rendering, where each scene has roughly 200 frames, to compute farms consisting of thousands of nodes. Each frame, consisting of independent, non-communicating tasks for lighting, shading, animation, etc., can take from minutes to hours to render. This work is highly speculative: the overwhelming majority of computation never makes the final cut. [Epps, 2004; Lokovic, 2004] Upon seeing initial frames (computed by a chain of tasks), an artist may decide that an object could be in a better location, e.g. With a batchactive scheduler, artists could prioritize key sections (such as those with more action) at rough quality to more quickly decide whether additional frames are worth seeing. If unviewed, possibly uncomputed, frames will not be needed, artists would cancel their renderings to free resources for other tasks belonging to them or their colleagues.

Computer-aided design is employed on shared compute farms to explore high dimensional spaces. Trace- and analytic-based tools in computer science are used to study, e.g., microarchitecture, disk characteristics, and network performance (using single processor tools such as Simplescalar, DiskSim, and ns-2). Parameter studies for feature extraction, search, or function optimization can continue indefinitely, homing in on areas for accuracy or randomly sampling points for coverage. With a batchactive scheduler, such chains of tasks could run in parallel with users analyzing desired and completed outputs and guiding the searches in new directions, canceling branches determined to not be useful. Speculative tasks would operate in the background when pressing outputs were needed. We treat computer-aided design as a subset of simulation, excluding tasks (such as physical grid modeling) requiring many communicating nodes and compute time on scales too large (weeks) to speculatively compute beyond the user's immediate needs.

2 Related work

Attempting to use traditional, deployed mechanisms to control speculation leads to poor scheduling and a cumbersome interface to the user. Mechanisms such as Unix `nice`, priorities in the Condor clustering system, FreeBSD's idle queue, and classed schedulers [Corbató et al., 1962] do not leverage inherent properties of user behavior (such as think time) or provide a useful cost model. (These considerations are amplified after describing the batchactive scheduling environment in Chapter 3.) Further, without scheduling support, and assuming user cooperation, which we do not, there is no clear way for users to throttle their own speculation because meeting individual and collective time and cost goals depends on many unknowns: the pattern of other users' task arrivals, task sizes, user think times, and the probabilities that speculative tasks will be needed.

Speculation to improve performance is found at the level of I/O requests, program blocks, and instructions across all areas of computing including architecture, languages, and systems [Hennessy et al., 2002; Osborne, 1990]. In batchactive scheduling, the mechanism of conveying that a task is speculative is a 'disclosure' hint [Patterson et al., 1995], revealing only user knowledge, enabling the system to globally optimize resource management, and remaining correct when the application execution environment changes. Here we compare against a varied set of relevant research systems.

Bubenik and Zwaenepoel [1989] modeled a set of users engaged in software development using a modified `make` tool. At each save of a source code file, their system speculatively runs the compiler using the build rules encoded in the project's `Makefile` and isolates speculative compilations from the rest of the system until the compilations are known to be needed. Their simulator modeled one task (rebuild) pending per user. Our model is broader, encompassing users who operate interactively or who submit batches of speculative work for a number of scenarios, including users behaving speculatively with non-speculative schedulers. Beyond their study of time-based metrics, we also study resource cost as it relates to user charges and server revenue.

Patterson et al. [1995] have shown in the TIP system how application performance can increase if the application discloses storage reads in advance of when data is needed. Programmers insert speculative data reads as program annotations in the hope that the system can use this information to reduce application I/O latency. Our work applies the same concepts and terminology to the processor resource at the granularity of tasks. As TIP uses disclosed reads to exploit storage parallelism, batchactive scheduling uses disclosed tasks to exploit compute farm parallelism.

In the database realm, Polyzotis et al. built a speculator that begins work on database queries, where each query could be considered a task, during the user think time in constructing complex queries [Polyzotis and Ioannidis, 2003]. Their system predicts what the user will need before the query is finished. They do not consider the scheduling issues of a competing set of users submitting needed and speculative queries.

One could phrase the goal of batchactive scheduling as minimizing a soft real-time utility function defined as the total amount of task visible response times.

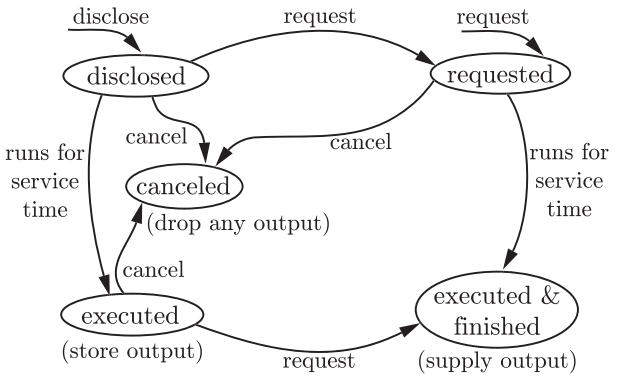

Figure 2: Batchactive task state transitions. When a task's resource usage equals its service time, the task becomes executed. If a task is both executed and requested, then the task is finished and the its output is supplied to the requesting user. If a task executes and was disclosed but not requested, then the task's output is stored in an isolated location until requested or canceled. If the task is canceled after executing, its output is dropped. Disclosed and requested tasks may also be canceled.

However, batchactive schedulers cannot know when a user will need task output, if the user will need speculative task output at all (i.e., the real-time *deadlines* are unknown), making a mapping to existing real-time scheduling difficult.

3 Batchactive scheduling

In a batchactive scheduler, users tag tasks as either speculative or certainly needed. (There are no 'levels' of speculation which could be burdensome to provide.) Users *disclose* tasks whose outputs they are not sure they will need at that time, i.e., the speculative tasks, and *request* tasks whose outputs they already know they need. Later, a user may promote a disclosed task to a requested task, or a user may *cancel* any task. This work makes no attempt to guess what tasks are more or less useful to the user. The application scenarios in Section 1.1 show that there exist important problems in which tasks can be categorized as either speculative or needed. The states in which a task may reside are depicted in Figure 2.

Tasks may be desired as *task sets* in a flat list order or with no ordering preference as shown in Figure 3. (We believe arbitrary DAG orderings would be tedious to maintain.) List order indicates the order that

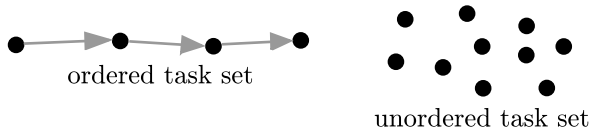

Figure 3: Flat list and unordered task sets. Tasks to the right of the list are more speculative.

the user desires outputs (usually tasks listed later are more speculative), not data dependencies among tasks. Applications with data dependencies are not good candidates for speculative execution. The unordered collection (called 'dynamic sets' in the realm of I/O [Steere, 1997]), often used when sampling a large space, indicates that the user does not mind which task outputs are returned first; any answer is helpful until more is known about the space. We only present flat list order results (Section 4) because for the applications under consideration, we think that users can and will want to order tasks so that they will not be inundated by many outputs.

Figure 4 depicts user interaction with batchactive software and the software's interaction with compute farm resources. Any number of users disclose, request, and cancel any number of tasks. The scheduler decides which and when disclosed and requested tasks run. If a task is canceled, it is no longer a candidate. The scheduler communicates decisions to the operating systems running on the resources which handle the details of running the tasks. If a task executes and was requested, then the task's output is supplied to the requesting user. If a task executes and was disclosed but not requested, then its output is stored in an isolated location until requested or canceled. There are file system mechanisms, such as permissions or manipulating the namespace with hard links, that can supply the output without an extra copy.

Hidden outputs impact capacity management. An administrator may choose from multiple solutions. If storage limits are reached, the oldest unrequested outputs may be deleted. Or, the system may guarantee that isolated outputs are kept for a minimum amount of time, and an administrator would add storage if necessary to meet this guarantee. Alternatively, hidden outputs may consume the user's file system quota (with the downside that a clever user may analyze quota usage to gain information without requesting and paying for tasks).

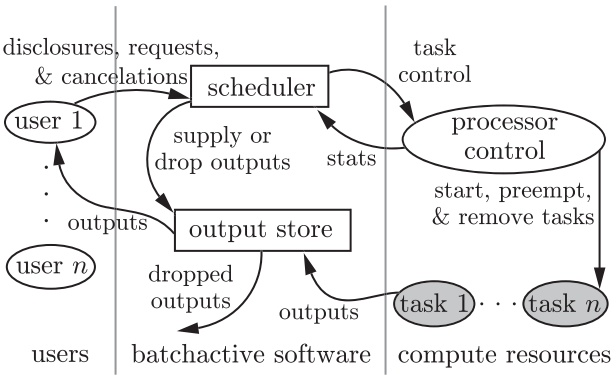

Figure 4: Interaction between users, the batchactive software, and compute farm resources.

A resource owner is either a *cost-center* which wishes to recoup its costs only, or a *profit-center* (a/k/a 3rd-party compute outsourcer, e.g., the hosting services provided by IBM and EDS) which wishes to maximize profit. A cost-center wishes to maximize the time the resource is busy, i.e., server load (varying from 0–1), to gain maximum utility of its resource. A profit-center also wishes to do this, but for the purpose of maximizing profit. Users, however, wish to pay as little as possible. Under traditional pricing per cycle, a user might hesitate to disclose speculative work for fear of being charged needlessly. (Even in a cost-center, some entity is charged, if not the user directly, then some part of the user's organization.) This is problematic because batchactive scheduling policies work better with more disclosed tasks.

We propose and study a novel pricing mechanism that does not charge for disclosed tasks that were never needed, irrespective of whether such a task did not run at all, ran partway, or finished. That is, batchactive pricing charges for resources used only by tasks whose outputs are requested. The user need not weigh the estimated cost (wasted money) and benefit (lower visible response time) of each disclosure, encouraging the user to freely disclose. Results indicate that this mechanism often does not hurt server revenue because users are able to submit work faster, resulting in more normally idle time being billed. (It is possible that paying less for computed but less desired tasks may make sense, but this is beyond the scope of the experiments we have done.)

Batchactive pricing is resilient to abuse. If a user requests speculative work, the user will be charged.

(In settings where the user is not charged for resource use, batchactive pricing does not add any potential for abuse that did not already exist.) Now consider a user who attempts to game the system by disclosing tasks that the user will never request (and thus never be billed for). For such tasks to benefit a user, they would have to cause tasks that the user will actually request to be preferentially scheduled. However, no policy introduced in this paper does this. The worst a user could do is swamp the disclosed queue, lessening its utility for other users. This can be countered by a policy that favors users who request speculative tasks more often (which we use below mainly to achieve better visible response times), or by an observant administrator.

As the server wishes to maximize *billed load,* users wish to minimize *scaled billed resources,* which we define as the ratio of the billed resources to the needed resources. Results look at scaled billed resources averaged over all users. It is always optimal (i.e., mean scaled billed resources takes the minimum value of 1) under batchactive pricing. But under traditional pricing, when users request tasks they later find they will not need, the ratio is greater than 1.

With respect to time, users wish to minimize *mean visible response time* over all tasks. (Without knowing relative task importance, as in most online settings, taking an unweighed mean to measure scheduling effectiveness is common practice.) A task is needed by a user at time t_n and executes (completes) at time t_e. A requested and executed task a has a corresponding visible response time defined as

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

Visible response time accrues only after a user asks for output from a task that may have been submitted much earlier and thus measures the time that a user actually waits for output, which is usually less than the time that a speculative task has been in the system. In particular, a task can and often does have 0 visible response time if it was speculatively disclosed and was completed while its user was examining the output of some other task. This consideration of when a task was needed, instead of when a task was submitted by users behaving speculatively with a non-speculative scheduler or disclosed by users with a batchactive scheduler, is more useful than conventional response time which conflates the time a task was requested with when it was needed. (We suspect, along with Feitelson et al. [1997], that this overloading has subsisted because of the difficulty in knowing when speculative task outputs are needed.)

A batchactive scheduler uses knowledge that some tasks are speculative. Policies may require information easy to obtain or uncertain information that must be predicted (when a speculative task will be needed and the probability that a task will be needed). This paper focuses on the former with *two-tiered* batchactive schedulers having two queues: one for requested (known-needed) tasks, and one for disclosed (speculative) tasks. Priority is given to the requested queue. Deployed supercomputer schedulers [NAS, 2002] with debug queues, administrator queues, etc., should be easy to extend to include a disclosed queue for batchactive scheduling.

For the requested queue, we studied standard policies such as FCFS (first-come-first-serve), SRPT (shortest-remaining-processing-time), and user-FB (user-based foreground-background, i.e., select the task from the user who has used the fewest resources) [Conway et al., 1967]. For the disclosed queue, in addition to these policies, we added a novel policy HRP (highest-request-probability) which selects the next speculative task from the user who has historically requested a greater fraction of speculative tasks. By avoiding executing speculative tasks that are less likely to be eventually requested, visible response time and billed load should improve. HRP also resists an attempt by a user to swamp the disclosed queue with tasks that will not be requested.

Batchactive scheduling introduces an interface different from traditional scheduling: (1) speculative tasks are initially *disclosed,* (2) needed tasks are explicitly *requested* when needed (or *canceled* if found not to be needed), and (3) task output is *isolated* until requested. Disclosure enables a scheduler to prioritize speculative tasks differently. Requesting (or 'pulling' task output) enables better policies based on learning user behavior (e.g., HRP) and enables the system to provide feedback to the user of his or her visible response times. Isolation enables batchactive pricing, which should encourage speculation.

Deployed schedulers (e.g., FreeBSD's idle queue and other classed schedulers) can emulate the two-tiered nature of batchactive scheduling to different degrees. Existing interfaces, however, do not provide information provided by the batchactive interface.

Many task requests and cancelations will not be seen by the system: If a needed task executes before the user needs its output, the user will likely directly consume task output (often stored on a distributed file system) and if an unneeded task executes before the user knows that its output is unneeded, the user has no motivation to cancel it. Moreover, the traditional interface of not isolating speculative output makes batchactive pricing impossible, for without isolation a user may never request (and pay) for needed output. When schedulers are available that permit two-tiered batchactive policies, we recommend a batchactive layer between the user and system so that the user is presented with the batchactive interface necessary to achieve all the characteristics of batchactive scheduling.

4 Simulation results

We study batchactive versus traditional scheduling by simulating synthetic users and tasks on a model of a single server, and we contribute our simulator called `ba_sim` for further research (`http://www.pdl.cmu.edu/PDL-FTP/Scheduling/ba_sim-0.1.tar.gz`).

Model The simulator models a constant number of users who enter the system at the start and cycle between submitting tasks to a single server, waiting for task output, and thinking about task output. Before submitting, each user plans speculative work as task sets which are organized as finite lists of tasks. On a non-speculative scheduler, users *request* these tasks either as needed (i.e., one at a time) or all at once, which we call *interactive* and *batch* usage, respectively. On a batchactive scheduler, users *disclose* all these tasks and *request* them only when needed, which we call *batchactive* usage. A user receives output after a requested task completes and considers the output for some think time. Then the user may need the next task, *cancel* remaining tasks and submit a new task set, or submit a new task set if the end of the current task set has been reached. (Only whole task set cancelation is performed because we believe it is simplest for the user.)

We could make batchactive scheduling look arbitrarily better with parameters highlighting its strengths. However, this would not be convincing. Instead, we chose parameters that are not only what we believe to be reasonable uses of speculation for the target applications, but that also include little or no speculation. Summarizing graphs were generated by sampling points in the 5-dimensional parameter space listed in Table 1. When varying individual parameters, the remaining parameters were fixed to the values in Table 2, unless otherwise noted.

parameter	range
number of users	1 to 16
task set change prob.	0.0 to 0.0–0.4 (uni.)
# of tasks per task set	1 to 1–21 (uni.)
service time (s)	20 to 3,620 (exp.)
think time (s)	20 to 18,020 (exp.)

Table 1: The parameter ranges used in summarizing results. A uniform distribution (uni.) is described by 'lower bound (a) to upper bound (b),' where the upper bound is specified by a range varied across runs. An exponential distribution (exp.) is described by its mean ($1/\lambda$).

The range of the *number of users* was chosen, based on other parameters, to provide minimal resource contention among users at the lower bound and to consume all of the simulated single server's resources at the upper bound.

The *task set change probability* is the probability that, after considering a task's output, a user will cancel his or her current task set and submit a new one. Each user is assigned a task set change probability from a uniform distribution whose lower bound is always 0 and whose upper bound is varied across runs. The upper bound ranges from modeling a user who always needs his or her speculative tasks (0%) to one who cancels his or her task sets 40% of the time after considering a single task's output. The *number of tasks per task set* dictates how many speculative tasks make up a user's task sets. Each user is assigned a number of tasks per task set from a uniform distribution whose lower bound is always 1, reflecting no disclosure, and whose upper bound is varied across runs. The upper bound ranges from no disclosure to a little over twenty disclosures, modeling a user who uses domain-specific knowledge to make small to medium-sized task plans.

Service time dictates task sizes. Each task is assigned a service time, regardless of who submitted it, from an exponential distribution whose mean is varied across runs. This value varies from one third of a minute to about one hour, based on BLAST DNA

parameter	setting
number of users	8
task set change prob.	0.0 to 0.2 (uni.)
# of tasks per task set	1 to 15 (uni.)
service time (s)	600 (exp.)
think time (s)	6,000 (exp.)

Table 2: The fixed parameters used in sensitivity analyses. For each sensitivity analysis, all but one parameter were held constant at these values.

similarity searches [Biowulf, 2004], film frame rendering [Hillner, 2003], and anecdotal surveys of wide-ranging exploratory searches and parameter studies [Petrou, 2004, ch. 4.7]. *Think time* dictates the time that users consider task outputs. Each time a task's output is delivered to a user, a think time is chosen from an exponential distribution whose mean is varied across runs. This value varies from one third of a minute to roughly five hours, reflecting a user who can make a quick decision to one who graphs, ponders, or discusses output with colleagues.

For each run, metrics were tabulated over two weeks of simulated time after two warmup days were ignored. Verification to increase confidence in the results, including hand-inspected trace outputs and non-speculative runs compared to operational laws such as Little's Law, may be found in Petrou [2004, ch. 6.1.4]. Further, Petrou [2004, ch. 6.3.2] presents 95% confidence intervals for a subset of the data in which each particular configuration was run 40 times with different random seeds, suggesting that the results are statistically significant, and thus confidence intervals are omitted for clarity.

Results Two-tiered batchactive policies are specified as *requested task subpolicy* × *disclosed task subpolicy.* In the graphs, simulations are identified by the scheduling policy followed optionally by the user behavior when it is either batch or interactive.

The improvement factors of mean visible response time, drawing parameters from Table 1, are shown in Figure 5. An improvement is 2, e.g., if batchactive mean visible response time was half of the non-speculative mean visible response time for a particular set of simulation parameters. The solid curve intersection with the value of 3 measured on the horizontal axis indicates that in about 10% of the runs, the improvement of mean visible response time for batchactive users using FCFS × FCFS compared to interactive users using FCFS was at least 3X.

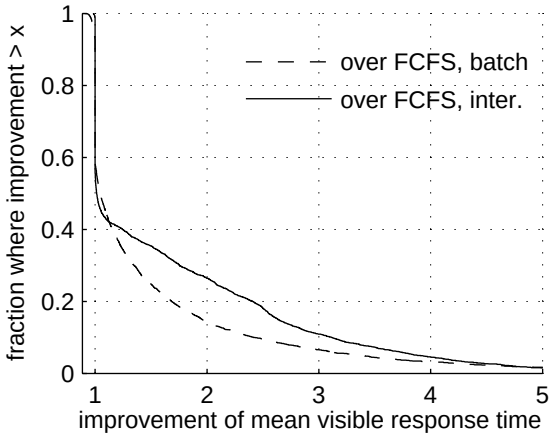

Figure 5: Inverse cumulative improvement of batchactive scheduling for mean visible response time. The horizontal axis shows improvement factors and the vertical axis indicates the fraction of the runs in which the improvement was *at least* as much indicated on the horizontal axis. FCFS × FCFS performs at least twice as well for about 15% and 25% of the simulated behaviors of batch FCFS and interactive FCFS, respectively.

For half of the cases in Figure 5, batchactive scheduling provides lower (better) mean visible response time than non-speculative scheduling (i.e., improvement greater than 1). The reasons for batchactive improvements are examined in the per-parameter investigations below. This summarizing graph shows that batchactive scheduling can provide factors of improvement (at least 1.5X better than batch usage for 25% of the runs, at least 3X better than interactive usage for 10%, etc.) and almost never performs worse.

Figure 6 shows how the number of users affects mean visible response time, drawing other parameters from Table 2. Nearly always, batchactive FCFS × FCFS performs best, exhibiting adaptability. This metric improves while simultaneously improving the throughput of needed tasks as discussed in detail in Petrou [2004, ch. 6.2.4]. Batchactive FCFS × FCFS is better than batch FCFS under many users because requested tasks never wait for speculative tasks; it is better than interactive FCFS under few users because it fills idle time with speculative tasks. When very busy, interactive FCFS and batchactive FCFS × FCFS begin to converge because the requested task

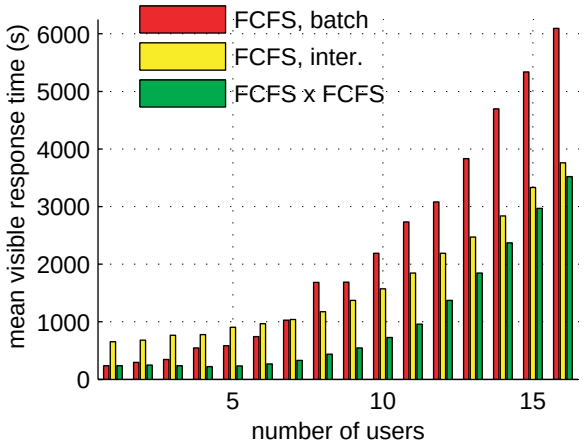

Figure 6: How the number of users affects mean visible response time (lower is better). With few users, batch FCFS is better than interactive FCFS; with many, interactive FCFS is better than batch FCFS. FCFS × FCFS adapts and always performs best. (In these bar graphs, the left bar is always batch usage, the middle interactive, and the right batchactive.)

queue of the batchactive scheduler is never empty. Thus, with a batchactive system, users do not need to decide how aggressively to submit speculative work: they may disclose all work not known to be needed to obtain these time-based improvements. Even on saturated resources, so long as some work is speculative, batchactive scheduling is beneficial (i.e., idle time is not a prerequisite for batchactive benefits). Batchactive scheduling also improves visible slowdown (visible response time scaled by task size). These improvements occur while also improving the variance of visible response time and without increasing the number of scheduling decisions the system needs to make.

SRPT is known to optimally minimize mean response time. As it requires task size knowledge, it is not often used. Our experiments show that FCFS × FCFS outperforms the size-based non-speculative cases (interactive and batch usage of SRPT), implying that a task size oracle will not diminish the value of batchactive scheduling. Size- and usage-based batchactive scheduling also outperforms their non-speculative, single-queue counterparts. While providing about the same improvement relative to FCFS interactive usage, the improvement relative to batch usage is not as pronounced. A more elaborate discussion is in Petrou [2004, ch. 6.2.7–8].

Perhaps unexpectedly, think time is required for speculation to provide visible response time benefits: Think time enables a batchactive scheduler to choose a better task ordering — favoring known or more likely to be needed tasks — because a user does not need the outputs of every speculative task at once; the user is either 'blocked on' one task's output or 'thinking about' the output of the previously received task output (Figure 1). We isolated the benefit of user think time (which exists in many contexts [Bubenik and Zwaenepoel, 1989; Crovella and Bestavros, 1995]) by turning off speculation (setting the task set change probability to 0). When think time was set to 0, batchactive and non-speculative usage resulted in equal mean visible response time. Under the presence of think time, batchactive scheduling provided performance better than common practice at all save the lowest and highest loads, though not as much benefit as when task sets were speculative (Figure 6). There is no model of think time in traditional scheduling. Batchactive scheduling and its interface — disclosing and requesting — exposes and successfully leverages think time.

Now we show the benefits of FCFS × HRP, which learns the likelihood of task request by a user. We varied how likely users would cancel and issue new task sets and show this effect on mean visible response time in Figure 7. The mean visible response time of both batchactive FCFS × FCFS and FCFS × HRP increases (worsens) as users become more speculative while the relative advantage of HRP remains unchanged. HRP schedules disclosed tasks well, even as well as an impractical oracle policy (discussed in Petrou [2004, ch. 6.2.9]) which never runs disclosed tasks that a user will not eventually request.

We varied the number of tasks per task set to large values — reflecting a search of high-dimensional spaces — to show its effect on visible response time in Figure 8. When all task sets have only one task, all cases provide the same mean visible response time. Interactive FCFS is immune to task set size because these users will have submitted at most one task from their task sets. Mean visible response time for batch FCFS and FCFS × HRP initially improve when there is some speculative work that may be performed while users are in their think times. Soon batch FCFS becomes unusable. Both FCFS × FCFS (not shown) and FCFS × HRP are resilient to large task sets. Between these two, FCFS × HRP does best, showing that the

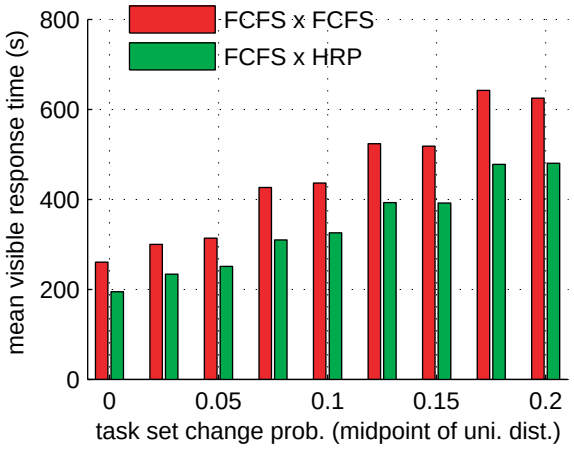

Figure 7: How task set change probability affects mean visible response time when introducing HRP. Shown on the horizontal axis is the average of the probability's lower (always 0) and upper bounds. Favoring the user who has requested the most disclosed tasks does best.

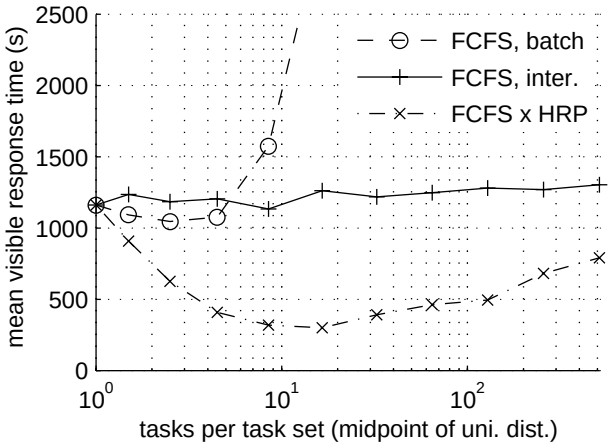

Figure 8: How the number of tasks per task set affects mean visible response time. This number is a uniform random variable and shown on the horizontal axis is the average of its lower (always 1) and upper bounds. HRP is resilient to large task sets. (This graph is log-linear.)

disclosed queue must be scheduled carefully to avoid a diminishing returns of batchactive improvement as the queue fills with tasks less likely to be requested. Eventually batchactive benefits decline as the disclosed queue consists largely of unneeded tasks.

The charges for unneeded speculation incurred by batch users of a non-speculative scheduler are shown in Figure 9. Under our novel batchactive pricing (not charging for unneeded speculation), batchactive users always pay less than users submitting batches of work to non-speculative schedulers. Interestingly, batch users pay even more for unneeded speculation under SRPT-based schedulers than here: the better the scheduling (the faster tasks run to completion), the more needless work completed.

Resource provider revenue is a function of billed (requested) load, shown in Figure 10 as the number of users is varied. FCFS × FCFS provides higher (better) requested load than interactive FCFS, because, by providing lower (better) mean visible response time, users submit needed work more quickly. At mid load, FCFS × FCFS is roughly 10% better than interactive FCFS. In FCFS × FCFS, some actual load is not charged (not shown). The tradeoff for batchactive schedulers improving visible response time relative to interactive usage is increased load. FCFS × FCFS cannot meet the requested load of batch FCFS, in which users request

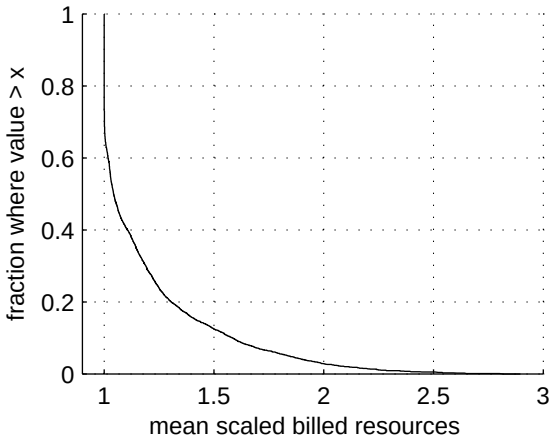

Figure 9: Inverse cumulative graph of mean scaled billed resources for batch FCFS. Batch usage bills for speculative tasks that run whether or not they are needed. The average user who speculates using non-speculative FCFS pays at least 30% more than necessary for 20% of the runs. (Mean scaled billed resources is optimal for interactive and batchactive users, and thus they are omitted.)

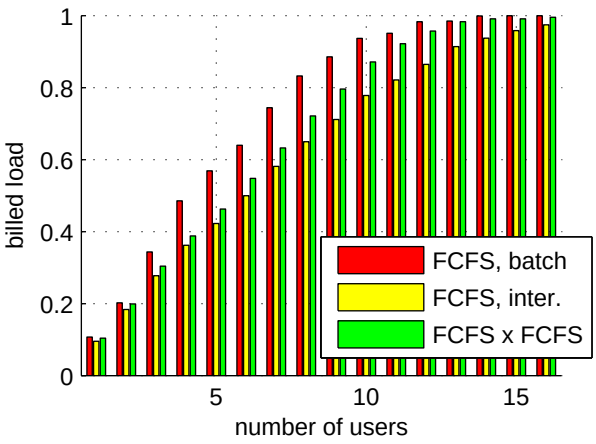

Figure 10: How the number of users affects billed load. Compared to interactive FCFS, FCFS × FCFS provides higher (better) billed load. FCFS × FCFS can only match batch FCFS's billed load with many users.

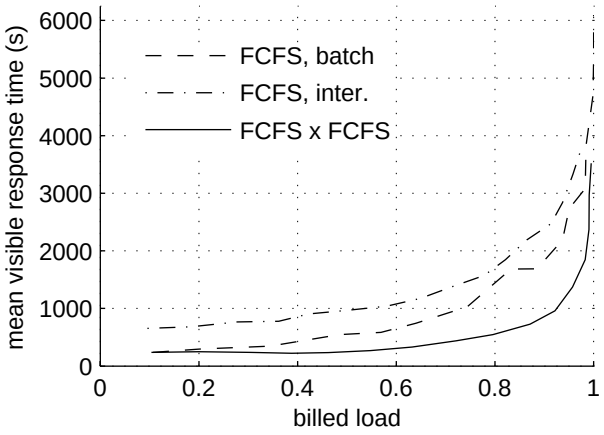

Figure 11: The relationship between requested load and visible response time as the number of users was varied. (Both axes are dependent axes.) FCFS × FCFS always simultaneously provides higher (better) billed load for the resource owner and lower (better) mean visible response time for the users. At a mean visible response time of 1,000, FCFS × FCFS provides 0.92 billed load while batch FCFS only provides 0.75.

their entire task sets and are billed for any execution, even that of speculative tasks that will be canceled.

Batchactive scheduling simultaneously provides lower (better) mean visible response time and higher (better) requested load (Figure 11). While batch FCFS can provide better billed load, latency-sensitive users will not push traditional schedulers into regions of high billed load, because at those levels of revenue visible response times are too high. Further, batch FCFS assumes a willingness for users to pay for potentially unneeded speculation (Figure 9). FCFS × FCFS achieves higher billed load relative to both batch and interactive FCFS under workloads exhibiting good mean visible response times. In other words, batchactive scheduling's uncharged load is actually smaller than the amount of underutilized resources that would be necessary in a traditional system to achieve the same kinds of visible response times.

As service time increases, the time-based performance of interactive usage and batchactive usage converge. As a limiting case, when a server is always running requested work, batchactive usage does not improve performance over interactive usage. Likewise, the performance of batch usage and batchactive usage converge when service time decreases as there is less needed work to perform. Think time affects batchactive improvement in opposite ways than service time.

5 Conclusions

Users often plan ahead, wishing to pipeline the consideration of received outputs with the execution of tasks whose outputs were not known to be needed when submitted. Existing limitations cause users who submit speculative tasks to be charged for unneeded work and to experience long delays, or behave interactively to avoid these risks. We presented batchactive scheduling for farm computing to maximize human productivity while minimizing unnecessary resource use. Users disclose speculative tasks, request tasks whose outputs they know they need, and cancel tasks if received outputs suggest their irrelevance. Batchactive scheduling exposes and leverages user think time and tracks speculation history to improve performance. Batchactive scheduling lowers the new visible response time metric (tracking only tasks blocking users) compared to non-speculative scheduling and users are encouraged to speculate with batchactive pricing (charging for only needed tasks).

In simulation, for mean visible response time, FCFS × FCFS performed at least twice as well for about 15% and 25% of the scenarios as batch usage and interactive usage, respectively, while almost never do-

ing worse. This metric was improved while simultaneously improving the throughput of needed tasks. We showed significant improvements when several to many speculative tasks were submitted and early task outputs were acted on while uncompleted tasks remain. As the number of tasks in the average task set increases beyond 10, the mean visible response time of batch usage of FCFS became unusable. Throughout a range of average tasks per task set from 10 to 512, the improvement of batchactive FCFS × HRP (which also resists a denial-of-service) over interactive FCFS was between a factor of 1.5 and 3. As the cancelation probability for the average task set increased from no speculation (0%) to 20%, we showed the advantage of favoring users who historically requested more disclosed tasks using HRP compared to the simplest batchactive scheduler FCFS × FCFS.

Batch users on non-speculative schedulers sometimes pay greatly for unneeded speculation. The potential fear of paying for unneeded speculation motivates batchactive pricing. While batchactive pricing leads to some resources being consumed without being billed, our results suggest that disclosed speculation benefits the resource's overall utility. Simulations showed that higher billed load cannot be sustained on traditional systems without resulting in dismal time metrics for users (and much lower needed task throughput). Further, if users are latency-sensitive, they will actually request and pay for more needed work under batchactive usage compared to batch usage. Thus, under batchactive scheduling, improved time metrics coupled with no monetary risk for the user to disclose speculation can encourage more users, the submission of longer chains of tasks, and larger tasks, resulting in higher billed load (increased server revenue). Our simulator is available for testing batchactive scheduling for particular domains.

Acknowledgements

We thank the members and companies of the PDL Consortium (including APC, EMC, Equallogic, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, and Sun) for their interest, insights, feedback, and support. This material is based on research sponsored in part by the Army Research Office, under agreement number DAAD19-02-1-0389.

References

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. 1990. Basic local alignment search tool. *Molecular Biology 215*.

Biowulf 2004. Using BLAST on Biowulf. `http://biowulf.nih.gov/apps/blast/index.html`.

Bubenik, R. and Zwaenepoel, W. 1989. Performance of optimistic make. *SIGMETRICS*.

Conway, R. W., Maxwell, W. L., and Miller, L. W. 1967. *Theory of Scheduling*. Addison-Wesley.

Corbató, F. J., Merwin-Daggett, M., and Daley, R. C. 1962. An experimental time-sharing system. *AFIPS*.

Crovella, M. E. and Bestavros, A. 1995. Explaining world wide web traffic self-similarity. Tech. Rep. BUCS-TR-1995-015, Dept. of Comp. Sci., Boston Univ.

DeGroot, D. 1990. Throttling and speculating on parallel architectures. *Parbase '90*.

Epps, D. 2004. Personal comm. R&D at Tippett Studio.

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. 1997. Theory and practice in parallel job scheduling. *IPPS / SPDP*.

Gibson, G., Nagle, D., Amiri, K., Butler, J., Chang, F., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., and Zelenka, J. 1998. A cost-effective, high-bandwidth storage architecture. *ASPLOS*.

Hennessy, J. L., Patterson, D. A., and Goldberg, D. 2002. *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann.

Hildebrand, D. and Honeyman, P. 2004. NFSv4 and high performance file systems: Positioning to scale. Tech. Rep. CITI-04-02, Univ. of Michigan.

Hillner, J. 2003. The wall of fame. *Wired Magazine 11,* 12.

Holliman, D. 2003. Personal comm. Past sys. admin. for the Berkeley Phylogenomics Group.

Lokovic, T. 2004. Personal comm. Engineer at Pixar.

NAS 2002. NAS system documentation. `http://www.nas.nasa.gov/User/Systemsdocs/systemsdocs.html`.

Osborne, R. B. 1990. Speculative computation in Multilisp. *Conf. on LISP and Functional Programming*.

Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. 1995. Informed prefetching and caching. *SOSP*.

Petrou, D. 2004. Cluster scheduling for explicitly-speculative tasks. Ph.D. thesis, Dept. of Elect. & Comp. Eng., Carnegie Mellon Univ. CMU-PDL-04-112.

Polyzotis, N. and Ioannidis, Y. 2003. Speculative query processing. *CIDR*.

Steere, D. C. 1997. Exploiting the non-determinism and asynchrony of set iterators to reduce aggregate file I/O latency. *SOSP*.