

On the User–Scheduler Dialogue: Studies of User-Provided Runtime Estimates and Utility Functions

Cynthia Bailey Lee and Allan Snavelly
San Diego Supercomputer Center
March 2006

Abstract

Effective communication between user and scheduler is an important prerequisite to achieving a successful scheduling outcome from both parties' perspectives. In a grid or stand-alone high-performance computing (HPC) environment, this communication typically takes the form of a user-provided job script containing essential configuration information, including processors/resources required, a requested runtime, and a priority. User requested runtimes are notoriously inaccurate as a predictor of actual runtimes. This study examines whether users can improve their runtime estimates if a tangible reward is provided for accuracy. We show that under these conditions, about half of users provide an improved estimate, but there is not a substantial improvement in the overall average accuracy. Priority, as implemented in many production schedulers, is a very crude approximation of the value users may attach to timely job completion. We show users are capable of providing richer utility functions than most schedulers elicit. Thus we explore two elements of the user/scheduler dialogue to understand if accuracy and completeness of information conveyed could be improved.

1 Introduction

Resource scheduling typically involves a dialogue between a prospective user of a resource and a scheduler to determine when the resource can be used, and for how long. Part of this dialogue may also determine how urgently the user needs the resource, or in other words how the value of the resource may change depending upon when it is available. Intuitively, schedules formed in the absence of exact information in each of these categories may be suboptimal.

To understand what we mean by a "dialogue" between the user and scheduler, it will be helpful to examine the very first supercomputer "scheduler," the scheme of *Tennis Court Scheduling*. This was the first scheduler on San Diego Supercomputer Center's Touchstone Delta [1] (a contemporary Delta system, belonging to the Concurrent Supercomputing Consortium is documented in [2]). This "scheduler" was in fact a set of human system operators who were responsible for managing phoned-in job requests from users.

While on one level, Tennis Court scheduling is the height of unsophistication, and clearly cannot scale to the large and busy systems and grids of today, we claim that it can still provide an important perspective from which to judge software-based schedulers. Human beings possess a creativity, flexibility and nuance of analysis that outclass any proposed scheduling algorithm, not just in terms of bin-packing algorithmics, but also in terms of the total user interface. One can imagine a lengthy negotiation between user and operator over job parameters and schedule availability to achieve the most satisfying result for all parties concerned. Operators, knowing the habits, personalities and relative importance of their users could assess the urgency of each job and act accordingly: backfilling, rearranging already scheduled jobs and perhaps even stopping already running jobs. It is this complex and nuanced negotiation process that we hold as the gold standard of a dialogue between user and scheduler.

Current (software) schedulers in both grid and massively parallel processor (MPP) systems have not re-attained the state of the art in the Tennis Court scheduler times, at least in terms of dialogue with the user. Modern scheduler communication takes the form of a user-provided job script that typically contains a requested runtime, a priority, the number of processors and other resources needed and essential information for executing the job.

This is crude communication and is also only one-way. The scheduler will provide no feedback, such as suggested modifications to the job to improve its wait time, which could be provided in Tennis Court scheduling. Often the only communication the user will receive is a notification that the job has started running, and that it has ended.

Most systems do allow the user to inquire about the state of the queue, for example how many other jobs are waiting and running, their sizes and their priorities. But the listings returned from these inquiries are invariably overwhelming and confusing. Even relatively savvy users are unable to use this data to determine an expected wait time for their job, or how they might alter their job to better fit the schedule. A few systems offer a best-guess expected time until a job can be started, but accuracy of this figure is still an issue.

In this paper, we will focus on two specific aspects of the user–scheduler dialogue, user-provided runtime estimates and job utility functions. Users are always required to provide a runtime estimate, but it is well known that the accuracy of the estimates they provide is extremely poor. We investigate whether they can be improved, specifically seeking to affirm or falsify the common hypothesis that users could be accurate, but pad their estimates to avoid having their jobs killed. In contrast to runtime estimates, users are not required, and indeed are not offered an opportunity, to provide a complex expression of the value of their job in the form of a *utility function*, the job's value as a function of turnaround time. Ironically, users seem able and interested in providing this information.

2 User-Provided Runtime Estimates

2.1 Inaccuracy: Characterization, Effects and Causes

It is a well-documented fact that user-provided runtime estimates are inaccurate. Characterizations of this error in various real workload traces can be found in several classic and recent papers. Cirne and Berman [21] showed that in four different traces, 50 to 60% of jobs use less than 20% of their requested time. Ward, Mahood and West [7] report that jobs on a Cray T3E used on average only 29% of their requested time. Chiang, Arpaci-Dusseau and Vernon [4] studied the workload of a system where there is a 1-hour grace period before jobs are killed, but found that users still grossly overestimate their jobs' runtime, with 35% of jobs using less than 10% of their requested time (includes only jobs requesting more than one minute). Similar patterns are seen in other workload analyses [22,3,5].

One might ask what impact user inaccuracy has on scheduler performance—why worry if user estimates are inaccurate? Indeed, Mu'alem and Feitelson have shown the surprising result that if workloads are modified by setting the requested times to $R * \text{actual runtime}$, average slowdown for the EASY and conservative backfilling algorithms actually *improves* when $R = 2$ or $R = 4$, compared to $R = 1$ (total accuracy) [3,14]. Similar results have been shown when R is a random number with uniform distribution between 1 and 2, or between 1 and 4, etc. [22,14]. But simply taking the accurate time and multiplying it by a factor does not mimic the "full badness of *real* user estimates" [22, emphasis added]. Using real user-provided times [22,3], some scheduling algorithms did still perform equivalently or slightly better, compared to the same workload with completely accurate times. However, other algorithms experienced significant performance degradation as a result of user inaccuracy [4,5]. *The key point here is that Mu'alem and Feitelson's result only applies to the specific algorithms they studied, and it is necessary to re-prove (or disprove) their result for each new algorithm individually.*

Also, even for an algorithm such as conservative backfilling, which shows some mild improvement of average slowdown with inaccurate estimates, it is at the cost of less useful wait time guarantees at the time of job submittal, and causing an increased tendency to favor small jobs over large jobs (which may or may not be desirable) [4,14].

Many factors contribute to the inaccuracy of user estimates. All workloads show a significant portion of jobs that crash immediately upon loading. This is likely more indicative of users' difficulties with configuring their job to run correctly, than difficulties with providing accurate runtime estimate [22]. However, a job's runtime may also vary from run to run due to load conditions on the system. In an extreme example, Nitzberg and Jones [9] found that on an Origin system where different jobs on the same node share memory resources, job runtime varied 30% on a lightly loaded system, to 300% on a heavily loaded system.

Mu'alem and Feitelson [22] note that because many systems kill jobs after the estimated time has elapsed, users may be influenced to "pad" their estimates, to avoid any possibility of having their job killed. Users may also be insufficiently motivated to provide accurate runtime estimates. Many users are likely unaware of the potential benefits of providing an accurate request, such as higher probability of receiving quicker turnaround (because of an increased likelihood of backfilling), or this motivation may not be strong enough to elicit maximum accuracy.

2.2 Requested Runtimes vs. Estimated Runtimes and The Padding Hypothesis

With regard to this padding, we believe that it is important to be precise about what users are typically asked to provide, which is a time after which they would be willing to have their jobs killed, and to distinguish this from the abstract notion of an estimate of their jobs' runtime. This leads us to prefer the term, *requested runtime* for the former, reserving the term *estimated runtime* for a best guess the user can make without any kill penalty (and possibly even with an incentive for accuracy).

In our experiment, we sought to affirm or falsify the following hypothesis, which we have observed is implicitly or explicitly assumed in many of the papers in the literature:

The Padding Hypothesis

Users know the runtime of their jobs; the error observed in requested runtimes is a result of users adding padding to an accurate runtime estimate they have in mind.

This study tests this hypothesis by asking users of the Blue Horizon system at the San Diego Supercomputer Center (SDSC) [8] for a non-kill-time estimate of their jobs' runtime, and offering rewards for accuracy.

2.3 Survey Experiment Design

Users of the Blue Horizon system submit jobs by using the command *lsubmit*, passing as an argument the name of a file called the job *script*. The script contains vital job information such as the location and name of the *executable*, the number of *nodes* and *processors* required, and a *requested runtime*. An analysis of the requested runtimes from the period prior to the experiment shows that the error has a similar distribution to that observed in other workloads, as seen in Figure 3.

During the survey period, users were prompted for a non-kill-time estimate of their jobs' runtime by the *lsubmit* program, randomly one of every five times they submit. We asked, at the moment of job submission, hoping that this will be the most timely and realistic moment to measure the user's forecasting abilities. The traditional requested runtime is not modified in the job script, we merely reflect that value back to the user and ask them to reconsider it, with the assurance that their response in no way affects this job.

Users were notified of the study, by email and newsletter, a week prior to the start of the survey period. The notification included information about prizes to reward the most accurate users (with consideration given also to frequency of participation), specifically one MP3 player and several USB pen drives. The prizes were intended to provide a tangible motivation for accuracy and thus to elicit the most accurate estimates users are capable of providing.

The text of the survey is as follows. First, the user is reminded of the requested runtime (kill time) provided in their script. The user is then queried for a better estimate. Finally, the user is asked to rate their confidence in the new estimate they provided, on a scale from 0 to 5 (5 being the highest). This question was designed to test if users could self-identify as good or poor estimators. The survey does not provide default values. A sample of the survey output is shown below in Figure 1.

```
% lsubmit job_script
#####
# You have been randomly selected to participate in a two-question survey #
# about job scheduling <as posted on www.npaci.edu/News>. Your #
# participation is greatly appreciated. If you do not wish to participate #
# again, type NEVER at the prompt and you will be added to a #
# do-not-disturb list. #
#####
In the submission script for this job you requested a 01:00:00 wall-clock limit.

We understand this may be an overestimate of the wall clock time you expect the job to take. To the
best of your ability, please provide a guess as to how long you think your job will actually run.
**NOTE: Your response to this survey will in no way affect your job's scheduling or execution on
Blue Horizon.
Your guess (HH:MM:SS)? 00:10:00
Please rate(0-5) your confidence in your guess: (0 = no confidence, 5= most confident): 3
Thank you for your participation.
Your Blue Horizon job will now be submitted as usual.
```

Figure 1. Sample user survey and response.

2.4 User Accuracy

Over the 9-week period of the survey, 2,870 jobs ran on the system (note that there were more job submissions than this, since many jobs are withdrawn while still waiting in the queue). We did not survey automated submissions (81) or jobs that requested less than 20 minutes of runtime (172). There were also 21 timeouts, where there was no response to the survey for more than 90 seconds; and 59 jobs that were submitted by the 11 people that decided not to take part in the survey. We only surveyed one in five job submissions (selected pseudorandomly), leaving us with a sample of 143 surveys.

Of these 143 surveys, 20 had actual runtimes that were equal to the requested runtime. This situation could possibly indicate that the user was precisely accurate or, more likely, that the scheduler killed the job once it reached its requested runtime. We decided to discard these survey entries since it was not possible to determine whether the job was completed or killed from the information we collected. In 16 other responses, the estimated runtime given in response to the survey was *higher* than the original requested runtime in the script. Taken at face value, this means that upon further reflection, the user thought the job would need *more* time than they had requested for it, in which case the job is certain to be killed before completing. Some of these responses appeared to be garbage (e.g. “99:99:99”) from users who perhaps did not really want to participate in the study or just hoped a random response had some chance of winning a prize. In our analysis, all of these higher responses were discarded, as well as a survey response indicating an expected runtime of 0

seconds. Henceforth when discussing the survey results, we will be considering and referring to these 107 complete and valid surveys only.

We divide the 107 responses into two categories. First, those where the estimated time (the survey response) was the *same* as the requested runtime (from the job script), numbering 56, and second, those where the user provided a reduced estimate in response to the survey, numbering 51. (See Figure 2.) Of the 51 responses where users provided a tighter estimated runtime, users cut substantially—an average of 35%—from the requested time. The average *inaccuracy* in this group decreased from 68% to 60%. By inaccuracy we mean the percent of requested (or estimated) time that was unused or exceeded (in the case of estimates it is possible, though unusual, in this survey, to underestimate the runtime), as given in the following formula:

$$\text{Inaccuracy} = \text{abs}(\text{base} - \text{actual_runtime}) / \text{base}$$

Where *base* is either the requested runtime or the estimated time from the survey. So for example, an estimated time inaccuracy of 68% means either that 32% of the estimated runtime was used, or that 168% of the estimated time was used. A requested time inaccuracy of 68% means that 32% of the estimated runtime was used (overruns are not possible for requested time).

Including both of our two categories of responses (same as requested time, and different), the overall the inaccuracy decreased from an average of 61% to 57%. Those users who did not tighten their estimate were notably more accurate than those who did revise it; their initial inaccuracy was 55%. To fully understand our two metrics it is helpful to understand an example: A not atypical user requested their job to run for 120 minutes, revised (estimated) the runtime at 60 minutes in response to the survey, and the job actually ran for 50 seconds (!). In this example the user tightened their estimate by 50%. But the inaccuracy of the request is 99%, and the inaccuracy of the estimate is improved only 1% down to 98%. Intuitively, many users are substantially improving extreme overestimates, still without making the bounds very tight.

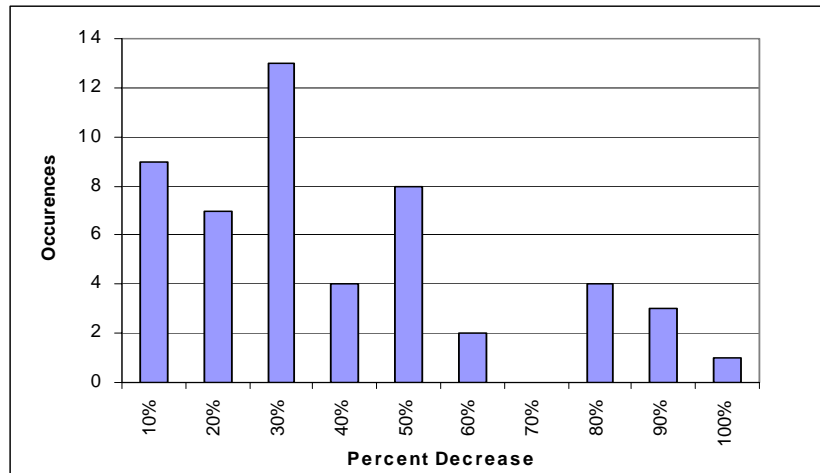


Figure 2. Histogram of percent decrease from the requested time to the estimate provided in response to the survey (includes only responses that were different from the requested time—56 responses had a 0% decrease). Categories represent a number of respondents up to the label, e.g. 20% represents 7 responses that were between 10% (exclusive) and 20% (inclusive) decreased from the requested time in the script.

In Figure 4 we show the comparison between the requested runtime in the script, and the actual runtime for the survey entries. The results are similar to those seen in Figure 3, where we see the same information but for the entire workload during the survey period, suggesting that the survey entries collected are a representative population sample. The results are also similar to those seen in the literature, in particular see [22]. Figure 5 shows the results if the estimate provided in the survey is used, instead of the requested runtime in the script. Note that no job’s actual runtime can exceed the requested runtime, but because the

survey responses were unconstrained in terms of being a kill time, the actual runtime can be either more or less than this estimate. The great majority of survey responses were still overestimates of the actual runtime. We cannot be sure why this is so, but it may be a lingering tendency due to users having been conditioned to overestimate by system kill-time policies.

Some degree of improvement can be seen in the pattern of error, for example a cluster of points on the right to right-bottom area of Figure 4 is largely dissipated in Figure 5. We can see that users still tend to round their times to 12, 24 and 36 hours in the survey, but not quite as heavily.

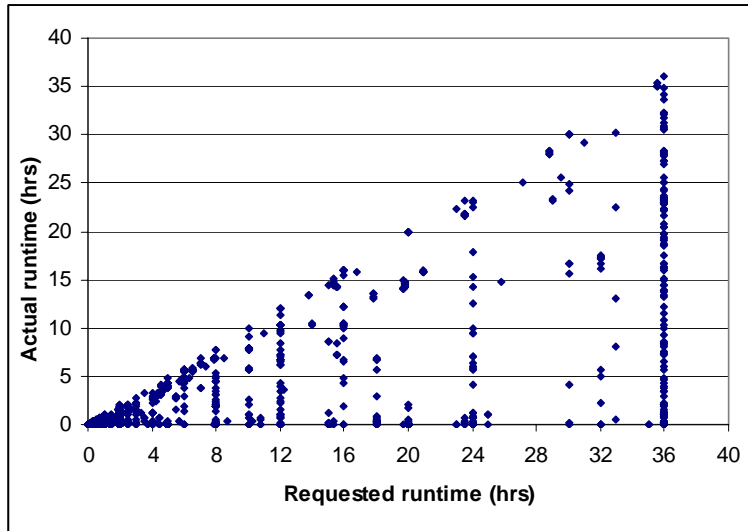


Figure 3. Comparison of actual runtime and requested runtime for all jobs on Blue Horizon during the duration of the survey period. Figure 4 shows the same data but only for jobs in survey sample. (N=2,870; note that some data points may be overlapping.)

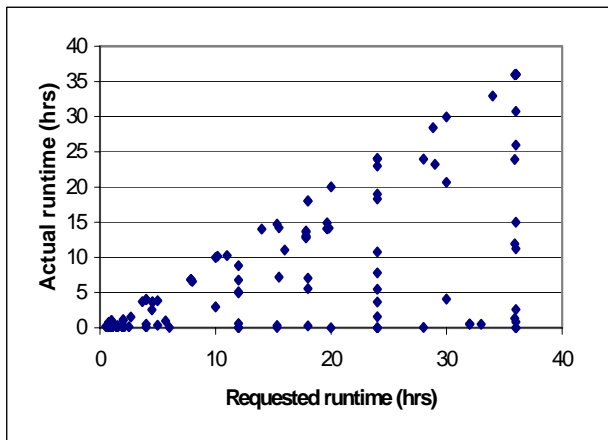


Figure 4. Correlation between requested runtime and actual runtime for jobs in the survey sample. (N=107; note that some data points may be overlapping.)

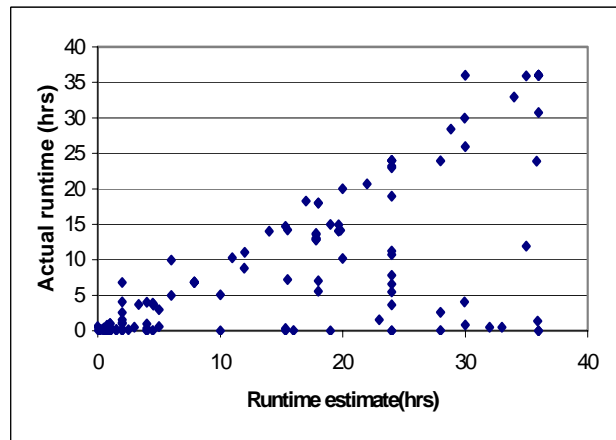


Figure 5. Correlation between users' survey runtime estimates and actual job duration. (N=107; note that some data points may be overlapping.)

2.5 User Confidence

It is likely that even the most motivated of users will not always be *able* to provide an accurate runtime request or estimate. But it may be useful if users can at least self-identify when they are unsure of their forecast. In our study, we asked users to rate their confidence in the runtime estimate they provided in response to the survey on a scale from 0 (least confident) to 5 (most confident). Figure 6, below, shows the

distribution of responses. In a majority (70%) of the responses, users rated themselves as most confident or very confident (5 or 4 rating) in the estimate. This is in spite of the fact that, overall, the accuracy of the requested runtimes and runtime estimates was poor (though typical, as observed in other workloads). It may be that users did not significantly adjust their forecasts of their jobs' runtime to account for possible crashes and other problems [11,12].

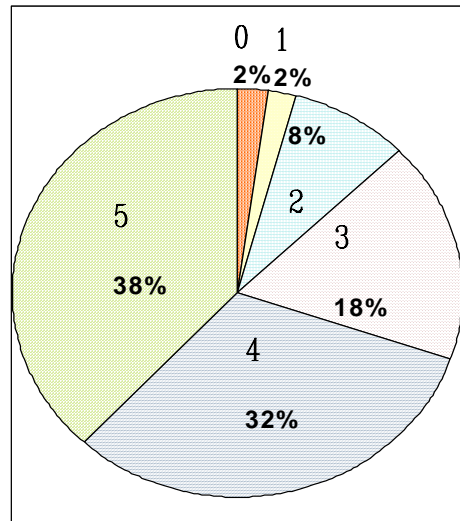


Figure 6. Distribution of user accuracy self-assessments (i.e. confidence) (N=107).

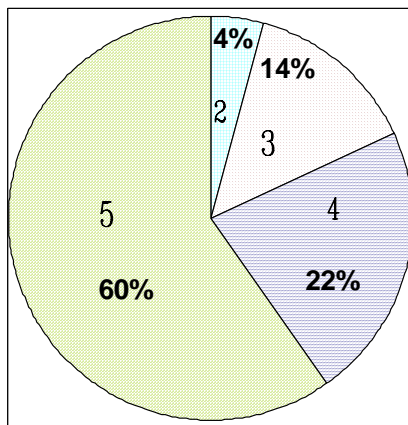


Figure 7a. Distribution of user accuracy self-assessments in users who *did not* change their requested runtime in response to the survey (N=56).

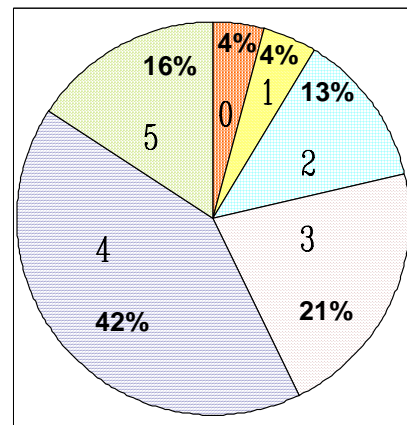


Figure 7b. Distribution of user accuracy self-assessments in users *did* change their requested runtime in response to the survey (N=51).

The responses are divided by category—those users who provided a revised estimate in response to the survey, and those who reiterated the requested runtime in their script. In Figure 7a, we see that in 60% of responses that were the same as the requested runtime, users rated themselves as most confident (5), with another 22% rated very confident (4). No users rated themselves as low or very low confidence (1 or 0). In contrast, of those responses that were a different estimate (Figure 7b), most users rated themselves somewhere in the middle (4 or 3).

Psychologists Kruger and Dunning [11] have observed that people who are most ignorant of a subject area are *more* likely to overestimate their own abilities than those who are knowledgeable. We wondered if our results were an instance of the same phenomenon. In other words, perhaps users reiterated the same requested runtime out of ignorance, and were then very self-confident, as predicted by Kruger and Dunning.

However, it appears that users who did not change in response to the survey, and had high confidence, did on average have more accurate estimates (as seen in Figure 8). For the unchanged responses, there is a clear pattern of decreasing average inaccuracy as the confidence increases. The same pattern is not seen in for those survey responses that were different from the requested runtime in the script. There does not seem to be a strong correlation between these users’ confidence and the accuracy of the estimates they gave in the survey.

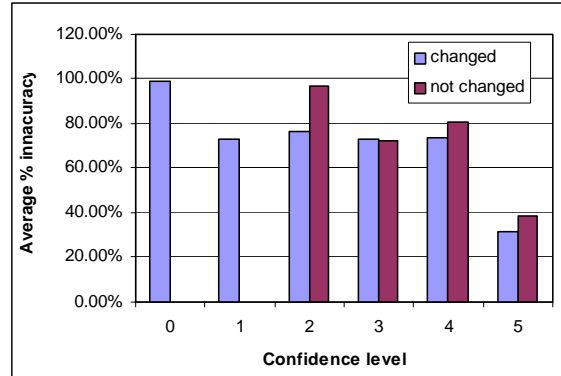


Figure 8. Average percent inaccuracy of user survey responses, separated into those responses that were changed and not changed with respect to the requested runtime in the script.

2.6 Other Approaches to Estimate Improvement

Asking the user for a more accurate time, as we have done in this study, is not the only approach to mitigating inaccuracy. One suggestion is to weed out some inaccurate jobs through speculative runs, in order to detect jobs that immediately crash [5,13]. Or, the system could generate its own estimates for jobs with a regular loop structure, via extrapolation from timings of the first few iterations [4]. Another proposal [6] is to charge users for the entire time they requested, not only the time they actually used. This idea, meant to discourage users from “padding” their estimates, may seem unfair to users because the fact that runtime may vary from run to run due to system load conditions necessitates a certain amount of padding.

3 Job Utility Functions

3.1 Utility Functions in HPC/Grid Scheduling

Commonly used metrics such as response time, bounded slowdown and expansion factor are all designed to capture the users’ desires to not be kept waiting. User-selectable priorities are one mechanism for communication between the user and scheduler, describing the relative urgency of a job. But the interplay between users and the completion of their jobs involves a richer context than is captured with these numbers—users have their own daily schedules and preferences, externally imposed schedules and preferences (e.g. conference paper deadlines), and so on. Virtually all HPC workload traces show diurnal patterns, with more small (presumably debugging) jobs during the day, and very infrequent new job submissions at night. If a scheduler sacrifices utilization in favor of very good response time for a particular job at 2 a.m., the effort is most likely wasted—the user will not even check the job until morning—but such a tradeoff could be highly desirable during peak business hours. Feitelson et al. present an example scenario to use as a starting point for thinking about the true needs and desires of users:

Assume that a job i needs approximately 3 hours of computation time. If the user submits the job in the morning (9am) he may expect to receive the results after lunch. It probably does not matter to him whether the job is started immediately or delayed for an hour as long as it is done by 1pm. Any delay beyond 1pm may cause annoyance and thus reduce user satisfaction, i.e. increase costs. This corresponds to tardiness scheduling. However, if the job is not completed before 5pm it may be sufficient if the user gets his results early next morning. Moreover, he may be able to deal with the situation easily if he is informed at the time of submittal that execution of the job by 5pm cannot be expected. Also, if the user is charged for the use of system resources, he may be willing to postpone execution of his job until nighttime when the charge is reduced. [23]

The utility function $u(t)$, or the change in user satisfaction over time, implied by this scenario is not a simple linear function, as is implicitly assumed in metrics such as response time. There are discontinuities (e.g. at 1 p.m.) and periods where the slope is zero (e.g. between 5 p.m. and the following morning). Furthermore, it is likely that every (user, job) pair will have a function with a different pattern.

In a Tennis Court scheduling scheme, the user in the example scenario would be able to convey in full all the relevant details of his schedule and preferences to the system operator during the course of their dialogue. But the user-scheduler dialogue in current schedulers either allows no discussion on this topic, or, most often, the user is allowed to choose from 2 or 3 priority categories (e.g. *High*, *Normal*, *Low*). These categories are broad and are not changeable over time, for example to handle the overnight hours when users will probably be unaware exactly when their job ran. Even in a system where only discrete categories are available, perhaps only two or three does not provide sufficient granularity. Using an analogy to mailing a package, postal patrons have a wide range of choices ranging from paying as little as 1.00 USD for slow ground shipping, to 30 USD or more for FedEx to deliver the package early the next morning.

There are proposed schedulers and scheduler evaluation schemes [15, 16, 17] that use simple job utility functions, as shown in Figure 9 (below). These functions are limited to a linear shape, but allow the user to set the maximum (starting) value to an arbitrary positive value, and the decay slope to an arbitrary negative value, allow greatly increased expressiveness from the common fixed priority categories. The form used in [15] also allows for a penalty for "late" completion of jobs. The linear decay in value only begins after an amount of time equal to the job's expected runtime has elapsed. On a fixed system, this does not affect the scheduling computation because the passage of time due to running the job is unchangeable and unavoidable. However, in a broader application, we believe it could be useful to represent the loss in value due to runtime as well. For example, system administrators could quantify the increased value to users of upgrading to a faster processor that decreases the runtime of jobs, in the same way that we analyze the increased value to users of a change in scheduling algorithms that decreases the wait time of jobs.

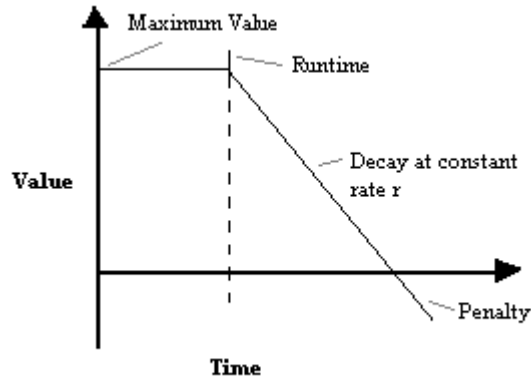


Figure 9. Job utility function as used in [15]. Despite the simple form, it is much more expressive than common "Low," "Normal," "High" priority categories. We investigate whether users can make use of even more expressive power.

We wondered whether actual users could and would express the complex job valuations available with an unconstrained utility function formulation, or whether the simple 2- or 3-category priority system or simple linear utility function was adequate. To answer this question, we undertook a survey experiment of users on SDSC's IBM Power3 system Blue Horizon [8].

3.2 Survey Experiment Design

The structure of the utility function survey is nearly identical to the runtime estimate survey described above. Again, we altered the script of the *llsubmit* command, so that one in five times it is run (selected randomly), the survey question is administered. Again, we think surveying users at the moment of job submission is the most timely and realistic moment to measure the user's willingness to pay, in relation to their willingness to wait. A sample of the survey output is shown below in Figure 10.

```

% llsubmit job_script
#####
# You have been randomly selected to participate in a one-question survey      #
# about job scheduling. Your participation is greatly appreciated. If you      #
# do not wish to participate again, type NEVER at the prompt and you will    #
# be added to a do-not-disturb list.                                         #
#####

Assuming your 1-node high priority job will use the full amount of time you specified in your Time
Limit, it will consume 130 SU's.
Based on some historical queue times for jobs of this size and priority on Blue Horizon, we estimate
your job will finish in 32 h 38 min.
Question:
If you could have this job finish 2 times faster (i.e. in 16 h 19 min) how many times 130 SU's is the
most you would be willing to pay for the improved turnaround? (ex: 1, 1.5, 2, 4.3, 10, times 130
SU's)? 4
Thank you for your participation.
Your Blue Horizon job will now be submitted as usual.

```

Figure 10. Sample user survey and response.

The text of the survey is as follows. First, the user is reminded of how many *service units (SU's)* their job will consume, given the processor (node) count and requested runtime in their script. Next, the user is given a rough estimate of the job's wait time, based on historical patterns for jobs of that size on the

system. The user is then presented with a hypothetical situation in which the total turnaround time (wait time plus execution time) of their job is increased or decreased by a factor of *time_factor*. Finally, the user is asked to provide a factor *cost_factor* reflecting how many times more or fewer SU's they would be willing to pay for the hypothetical turnaround time.

3.3 Results

Since the purpose of the survey is to examine the complexity of valuation expressiveness users could provide, we will only analyze the results for users who responded to the survey more than once per job. For the purposes of analysis, we say a job is the same job being run again when we observe a job script containing the same processor count and requested runtime as a previous script from the same user (admittedly this is only an approximation of whether a job is truly the same job). Obviously many users do not have a workflow that entails repeatedly running the same job with the same processor/time configuration many times, and thus we unfortunately discard a majority of the survey responses because they lacked additional survey data points. One way to avoid this would have been to collect several data points from the user at one time, however in order to secure permission to collect this information, we were severely constrained in the amount of time a instance of the survey could consume. However, as a result of the positive user response to this survey, anecdotally and in the results, we do now believe many users would be receptive to providing more than one data point at a time.

Each survey response represents a different (*time_factor*, *cost_factor*) pair. Recall that the *time_factor* is a value provided by the survey to create the hypothetical scenario, and *cost_factor* is the user's response to the survey, the premium or discount they would expect for *time_factor* faster or slower turnaround. Using the job's original parameters, we translate each (*time_factor*, *cost_factor*) pair into a (*time*, *utility*), or (*t*, *u(t)*) pair according to the following formulas:

$$\text{Turnaround time } t = \text{time_factor} * \text{original_turnaround}$$

$$\text{Utility } u(t) = \text{cost_factor} * \text{original_cost}$$

$$\text{original_turnaround} = \text{requested runtime} + \text{predicted wait time}$$

$$\text{original_cost} = \text{requested processors} * \text{requested runtime} * \text{priority weight}$$

$$\text{priority_weight}^1 = \begin{cases} 1 & \text{for Normal priority} \\ 2 & \text{for High priority, and} \\ 1.8 & \text{for Express priority} \end{cases}$$

An additional data point is created using the job's actual parameters, (*original_turnaround*, *original_cost*), since presumably the user is willing to pay for the job they are submitting, under the conditions they are submitting it. Then all the (*t*, *u(t)*) pairs for a given job are aggregated together into a single utility function. We do this knowing that it is an imprecise interpretation of the data. Indeed, we expect that users will have different needs and desires at different times, even for the same job. Thus, because the responses were elicited on different occasions, possibly spanning days or weeks during the month-long duration of the survey, it is possible to have different values of *u(t)* for the same value of *t*. It is also possible to have points where the value of the job seems to have *increased* over time, which is also not allowable in a strict definition of utility functions. The resulting utility functions are shown below in Figure 11a–g, one user per graph (some users have more than one job).

¹ These categories and associated cost values are particular to SDSC's policy at the time we conducted the survey.

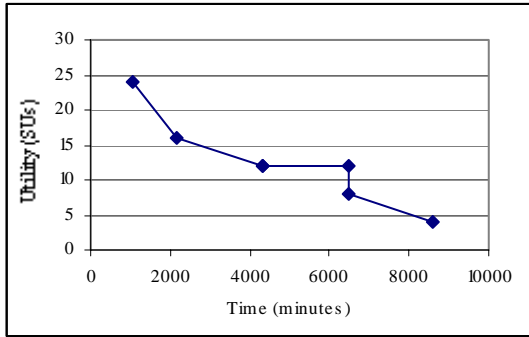


Figure 11a.

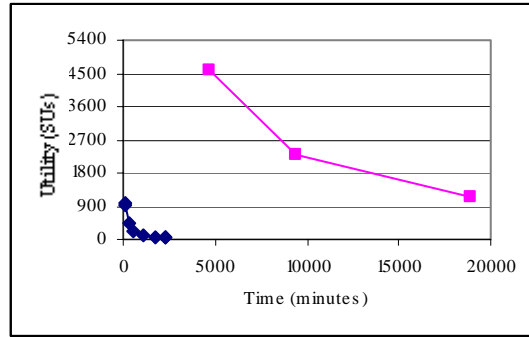


Figure 11b.

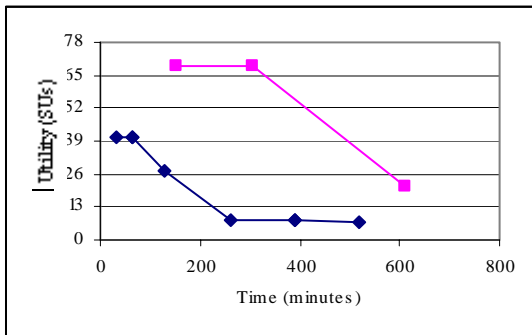


Figure 11c.

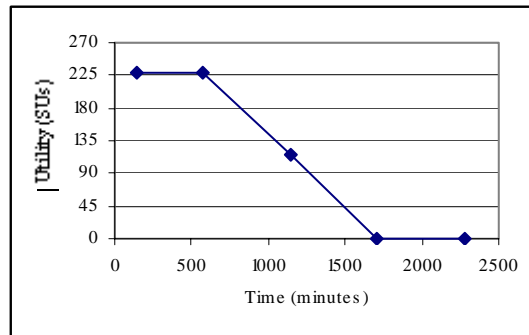


Figure 11d.

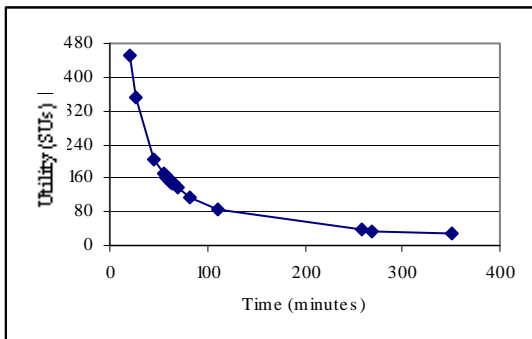


Figure 11e.

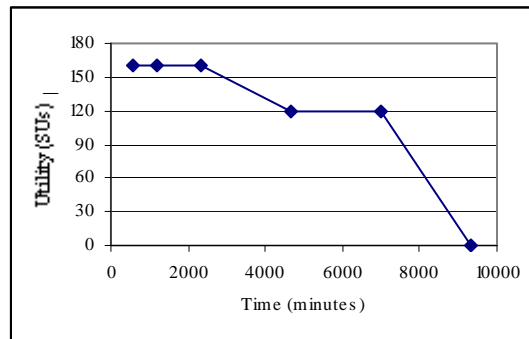


Figure 11f.

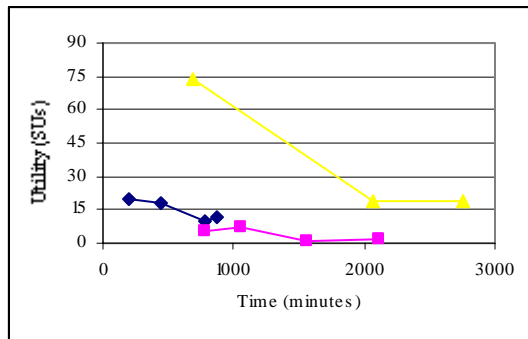


Figure 11g.

The first significant finding from looking at these utility functions is that none of them is linear. Further, many of the complex traits listed above in connection with the example narrative scenario are indeed observed in these actual user-provided functions. There are periods of time where the slope is zero (Figure 11 a, c, d, e and g), a very steep drop in value in the moments after job submission with a leveling

off later (Figure 11 b, c and f), and a deadline-like utility function where the value is constant until the deadline nears, causing a sharp decline in utility (Figure 11 c, d and e).

While these represent just a small sampling of users and jobs, they provide clear evidence that users not only have complex needs and desires regarding the scheduling of their jobs, but also that they are able to express themselves when given the opportunity. It is significant that these functions were elicited on a purely voluntary basis, from users who had little to gain by participating, and whose jobs were unaffected by their responses. Perhaps users would be even more willing to provide even greater thoughtful detail if they were engaged in an actual dialogue with the scheduler and their input would have an impact on the scheduling of their job.

4 Conclusion

We believe that sound design of any software—including schedulers—must not ignore the human-computer interaction (HCI) component of the system. In the case of schedulers, a successful submit-time dialogue between the user and the scheduler lays the foundation for optimal scheduling outcomes. In this paper, we present studies of two key elements of this dialogue: user-provided runtime estimates and job utility functions.

We asked the question, are users are capable of providing more accurate runtime estimates? To answer this question, we surveyed users upon job submittal, asking them to provide the best estimate they can of their job's runtime, with the assurance that their job will not be killed after that amount of time has elapsed. We have demonstrated that some users will provide a substantially revised estimate but that, on average, the accuracy of their new estimates was only slightly better than their original requested runtime. Thus, we find that the Padding Hypothesis is false—even absent a reason to pad, users do not demonstrate knowledge of a highly accurate estimate, so padding cannot be the sole cause of estimate inaccuracy. However, it is clearly one of several major causes, as many users showed an awareness that they do pad by reducing their estimates. Another useful outcome of the survey was the observation that many users were able to correctly identify themselves as more or less accurate in their estimating than other users.

Utility functions are widely used in economics as a means of expressing possibly complex changes in the value of a completed job over time. We wondered if users of HPC systems had more complex feelings about their jobs than can be expressed in the simple and discrete "High," "Normal" and "Low" priority choices available on most systems. We surveyed users upon job submittal, asking how much more or less they would be willing to pay for earlier or later job completion, respectively. We found that users do indeed have a wide variety of valuations and function shapes. Even for the same user, valuations vary from project to project, and from day to day on the same project.

5 Future Work

In future work, we plan to continue studies of the user-scheduler dialogue, including quantifying the impact of both robust dialogue and poor dialogue on scheduling outcomes.

We will measure the impact that better user estimates have on scheduler performance where "performance" may be defined (as traditionally) by response time, but also other metrics of interest to users, for example predictability and fairness. As to the somewhat negative finding of this study that users cannot significantly improve accuracy; we are not yet convinced this will always be the case. Even in this study users could tighten the bounds somewhat, and self-identify as to confidence. Education, feedback

and software tools might enable users to be much more accurate. It is clear that users could benefit from tools for performance evaluation and prediction of their own codes, something our research group has done extensive research on [18]. We also developed a prototype web-based tool Blue View to visually present the Blue Horizon scheduler's current plans for running and queued jobs. The tool shows "holes" in the schedule, where, if the user knows their job will fit, they have a chance to run early. Thus they might be willing to take measures, including providing highly accurate runtime estimates, to make the fit.

As to user utility functions, we propose a scheduler that explicitly optimizes to minimize the delay cost embodied in the functions. This delay-cost scheduler, similar to the time-sharing delay-cost scheduler proposed by Franaszek and Nelson [19], would charge users according to the loss in value that the running of their job inflicts on other jobs through delay. Calculating this optimization based on job utility functions is a mixed-integer program (a linear programming problem with integer constraints on some variables), which is NP-hard. Notwithstanding, commercial solver software packages can achieve good solutions in very reasonable timeframes (on the order of seconds), on problem sizes that reflect most current systems and workloads. [20]. The key observation is that metrics such as throughput and average turnaround are *machine centric*. We prefer metrics for evaluating schedulers related to user satisfaction—interpreted broadly. Opening up the user-scheduler dialogue enables these by eliciting user information and (potentially) improving scheduling algorithms accordingly.

6 Acknowledgements

This work was sponsored in part by NSF cooperative agreement STI-00230925 "Data Intensive Grid Benchmarks," DOE SciDAC award titled "Performance Evaluation Research Center," DOE award titled "HPCS Petascale Development Time Analysis," DoD HPCMP award titled "HPC Performance Evaluation," and NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the San Diego Supercomputer Center. We gratefully acknowledge all the users of Blue Horizon who participated in the surveys, and the staff who supported the administration of the surveys.

References

- [1] Pfeiffer, Wayne. Personal Interview. San Diego Supercomputer Center, at the University of California, San Diego. La Jolla, CA. April 9, 2004.
- [2] Messina, Paul C., "The Concurrent Supercomputing Consortium: year 1." *Parallel & Distributed Technology*, 1(1). February 1993.
- [3] Srinivasan, Srividya, Rajkumar Kettimuthu, Vijay Subramani and P. Sadayappan. "Characterization of Backfilling Strategies for Parallel Job Scheduling," *Proceedings of 2002 International Workshops on Parallel Processing*, August 2002.
- [4] Chiang, Su-Hui, Andrea Arpaci-Dusseau and Mary K. Vernon. "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance," *Proceedings of the 4th Workshop on Workload Characterization*, Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, eds. July 2002.
- [5] Lawson, Barry G. and Evgenia Smirni. "Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," *Proceedings of 8th Workshop on Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, eds. July 2002.
- [6] Stoica, Ian, Hussein Abdel-Wahab and Alex Pothen. "A Microeconomic Scheduler for Parallel Computers," *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph, eds. April 1995.

- [7] Ward, William A. Jr., Carrie L. Mahood and John E. West. "Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy." *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, eds. July 2002.
- [8] Blue Horizon, National Partner for Advanced Computing Infrastructure (NPACI).
www.npaci.edu/Horizon/
- [9] Jones, James Patton and Bill Nitzberg. "Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization." *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson, Larry Rudolph, eds. April 1999.
- [10] Kruger, Justin and David Dunning. "Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments," *Journal of Personality & Social Psychology*, 77(6). December 1999.
- [11] Lovallo, Dan and Daniel Kahneman. "Delusions of Success." *Harvard Business Review*, 81(7). July 2003.
- [12] Buehler, Roger. "Planning, personality, and prediction: The role of future focus in optimistic time predictions." *Organizational Behavior & Human Decision Processes*, 92(1/2). September/November 2003.
- [13] Perkovic, Dejan and Peter Keleher. "Randomization, Speculation, and Adaptation in Batch Schedulers." *Proceedings of Supercomputing 2000*. November 2000.
- [14] Zotkin, Dmitry and Peter Keleher. "Sloppiness as a Virtue: Job-Length Estimation and Performance in Backfilling Schedulers" *8th IEEE International Symposium on High Performance Distributed Computing*, Redondo Beach, CA. August 1999.
- [15] Irwin, David E., Laura E. Grit, Jeffrey S. Chase. "Balancing Risk and Reward in a Market-Based Task Service," *13th IEEE International Symposium on High Performance Distributed Computing*, June 2004.
- [16] Chun, Brent. N. *Market-based Cluster Resource Management*. PhD thesis. University of California, Berkeley. November 2001.
- [17] Chun, Brent. N. and David E. Culler. "User-centric performance analysis of market-based cluster batch schedulers." *2nd IEEE International Symposium on Cluster Computing and the Grid*, May 2002.
- [18] Snaveley, Allan, Laura Carrington, Nicole Wolter, Jesus Labarta, Rosa Badia, and Avi Purkayastha. "A framework for performance modeling and prediction." *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. November 2002.
- [19] Franaszek, Peter A. and Randolph D. Nelson. "Properties of delay-cost scheduling in time-sharing systems," *Journal of Research and Development*, 39(3). 1995.
- [20] Lee, Cynthia, Allan Snaveley, Robert H. Leary, Laura Carrington, Henri Casanova, Roger Bohn, Richard Carson, Jennifer Hardy and Yael Schwartzman. "Towards High-Order Performance Objectives for HPC System Scheduling." San Diego Supercomputer Center Technical Report. March 9, 2004.
- [21] Cirne, Walfredo and Fran Berman. "A comprehensive model of the supercomputer workload." *Proceedings of IEEE 4th Annual Workshop on Job Scheduling Strategies for Parallel Processing*. Cambridge, MA. 2001.
- [22] Mu'alem, Ahuva W. and Dror G. Feitelson. "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Trans. Parallel & Distributed Systems*, 12(6). June 2001.
- [23] Feitelson, Dror G., Larry Rudolph, Uwe Schwiegelsohn, Kenneth C. Sevcik and Parkson Wong. "Theory and Practice in Parallel Job Scheduling." *Proc. of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, eds. April 1997.