Evaluating the Impact of Inaccurate Information in Utility-Based Scheduling

Alvin AuYoung, Amin Vahdat, and Alex C. Snoeren
University of California, San Diego

ABSTRACT

Proponents of utility-based scheduling policies have shown the potential for a 100–1400% increase in value-delivered to users when used in lieu of traditional approaches such as FCFS, backfill or priority queues. However, perhaps due to concerns about their potential fragility, these policies are rarely used in practice. We present an evaluation of a utility-based scheduling policy based upon *real* workload data from both an auction-based resource infrastructure, and a supercomputing cluster. We model potential sources of imperfect operating conditions for a utility-based policy: *user uncertainty* and *wealth inequity*. Through simulation, we find that while the value delivered by a utility-based policy can degrade to *half* that of traditional approaches in the worst case, the policy we study provides 20–100% *improvement* under realistic operating conditions. We conclude that future efforts in designing utility-based allocation mechanisms and policies must explicitly consider the fidelity of elicited job value information from users.

1. INTRODUCTION

The number of consumers who require on-demand computation and off-site storage is growing rapidly. Companies like Amazon, Sun and Microsoft have invested in a cloud-based services model, whereupon internal resources are provisioned and leased – on-demand – to consumers. Most providers offer flat-rate pricing with soft resource guarantees, but experience has shown that in the presence of excess or unknown demand, such schemes often squander significant value for all parties: value in terms of unrealized revenue for the service provider, and diminished satisfaction for the consumer.

Grid and supercomputing providers have long faced similar challenges, and typically deal with over-subscription through either a *first-come-first-served + backfill* (FCFS+backfill) allocation policy, or some form of *fixed-price priority-based* scheduling. The fundamental shortcoming of a FCFS policy is that it does not take into account the value of a job when allocating resources. The fixed-priced scheme used in traditional priority-based schedulers elicits some information about a job's value, but is frequently insufficient when overall resource demand fluctuates significantly—a common occurrence in cloud-computing environments. Studies have demonstrated a potential for increased performance in these settings by using more sophisticated allocation techniques that consider additional information about a job's value.

Specifically, *utility-based* policies have been shown to dramatically outperform existing FCFS+backfill or priority-based approaches. These policies employ market-inspired mechanisms to infer information about resource demand in order to determine socially desirable allocations—which is to say, allocating scarce resources to the users who derive the most value, or *utility*, from their jobs. The added information inferred by a utility-based approach allows it to make a more informed scheduling decision than traditional approaches and adapt to resource demand as it changes. The magnitude of improvement depends greatly on the details of a particular environment, but researchers have claimed improvements ranging from 4–20% [23] to 100–1400% [12]; with such significant potential for improvement, a utility-based scheduling policy appears attractive for use in cloud-computing environments.

However, despite such lofty claims, utility-based techniques are rarely deployed in practice. Opponents of market-inspired techniques often dismiss these mechanisms as too burdensome for end-users, inequitable in their allocations, or too fragile to use in production environments [30]. While proponents cite the results of numerous independent simulation and theoretical studies showing significant performance improvement, these studies assume perfect operating conditions. Operators argue that it is unclear how utility-based systems will perform in practice. Not only might they not improve performance, but the potential exists to significantly harm throughput and fairness in existing systems should users provide *inaccurate information*. Lack of deployment—and subsequent analysis—makes it difficult to advance this long-standing debate.

In order to help resolve this impasse, we use an extensive trace-based simulation to evaluate the robustness of a utility-based scheduling approach when subjected to imperfect conditions likely to exist in a real deployment. We begin by addressing the fundamental concern that the additional information gathered by utility-based schedulers, namely users' expressed job utility, may be inaccurate with respect to their actual (internal) value for jobs. Our evaluation considers two distinct sources of this inaccuracy, which stem from imperfections in user knowledge and a utility-based scheduler's ability to extract this knowledge, respectively. We term these sources of inaccuracy *user uncertainty* and *wealth inequity*.

Uncertainty exists, for example, when a user is unsure how to assign a value to a job (e.g., due to insufficient information about future demand or job importance [21, 28, 30]), or is otherwise unable to accurately determine job-specific characteristics prior to submission, such as estimated running time [16, 22, 32, 33]. Market-inspired utility schedulers use various pricing mechanisms to ex-

SC09 November 14-20, 2009, Portland, Oregon, USA

tract truthful user valuations. Wealth inequity among users may decrease the effectiveness of such approaches; for example, wealthy users may consistently overstate the value of their jobs (i.e., overpay for resources), and therefore consistently receive a larger share of resources than relatively less wealthy users, regardless of need.

For both sources of inaccuracy, we develop a parameterized model that allows us to vary the level of inaccuracy from zero—meaning all user-provided information is perfectly accurate—to one, where user-provided utility is entirely uncorrelated with true values. We use this model to perform trace-based simulations on two distinct workloads. We are unaware of any publicly available cloud-computing workloads. Instead, we consider 15 months of workload data from the *Mirage* sensor-network testbed at Intel Research Berkeley [11]. Resources on the Mirage testbed are allocated through an auction-based utility scheduler. Hence, the workload trace provides a job utility information from real users, along with real workload data. We also study a data set taken from the *SDSC-SP2* supercomputer cluster, and use a tool to generate utility information for this trace [23]. Based on simulations driven by these two workloads, we present the following main results:

- We confirm practitioners' fears: faced with extreme levels of uncertainty or wealth imbalance, a utility-based scheduler can *under-perform* traditional approaches, delivering roughly *half* the utility of a traditional FCFS+backfilling scheduler in the worst case.

- Significantly, however, we find that utility-based scheduling is robust to levels of inaccurate information that occur in existing computing environments. Specifically, if we expect user utility information to be at least as accurate as observed inaccuracies in run-time estimates, we can expect an increase in value of at least 20-100%.

- Finally, we argue that in general, the effectiveness of any utility-based allocation mechanism is dictated by its ability to accurately extract user valuation. Specifically, we demonstrate that a fixed-price priority queue allocation mechanism can degrade to the performance of a simpler, FCFS+backfill approach under a variety of conditions. Therefore, we argue that the design of future pricing mechanisms (e.g., fixed prices, auctions) or monetary policies (e.g., virtual currency distribution, spending limits) must explicitly consider the expected fidelity of elicited job value information from users.

To our knowledge, ours is the first study of a utility-based scheduling system to analyze the potential impact of the imperfect information likely to be submitted to the scheduler in a live deployment using real workloads. Quantifying the effect of inaccurate information is vital to understand how robust the underlying market mechanisms [17, 26] must be in order to effectively support a utility-based scheduling approach, or, conversely, to understand when the inability for users to provide sufficiently accurate information might prevent the effective deployment of a utility-based mechanism. Our results indicate that despite the potential for poor performance in extreme cases, utility-based scheduling is robust to reasonable levels of inaccuracy and seems likely to outperform traditional scheduling techniques in practice.

2. BACKGROUND

In this section we present a summary of the state of the art in job scheduling algorithms and survey the related literature on characterizing various forms of inaccurate information on these systems.

2.1 Job scheduling techniques

We separate job scheduling algorithms into two categories: traditional (non-utility-based) and utility-based approaches, which we briefly summarize here. See Feitelson *et al.* [15] for a more complete summary of non utility-based job scheduling algorithms and Yeo and Buyya [37] for utility-based allocation approaches.

2.1.1 Traditional approaches

Perhaps the most natural resource allocation scheme in *time-shared* systems is proportional share, which provides equal, simultaneous access to time-shared resources to all users. In this model, multiple applications can run on a machine simultaneously. It is well known, however, that proportional-share scheduling alone does not function well in systems where over-demand for resources is common. In particular, as demand for resources increases, the time-share per resource received by each user decreases, while overheads remain constant (or, in some cases, increase due to thrashing). In the end, the utility delivered to each user goes to zero as the number of competing users increases.

Batch scheduling is commonly used to schedule jobs in *space-shared* systems, such as supercomputers and grid computing environments. As opposed to time-shared resources, space-shared systems are well suited for applications which require a large share of a system's resources (e.g., a CPU-intensive job), and therefore, run a single application on a machine at a time. First-come first-served (FCFS) scheduling is among the earliest and simplest techniques used in batch computing environments. In this scheduling discipline, users submit a job, along with the desired degree of parallelism (henceforth referred to as *size*). The jobs are executed in order of arrival. A limitation of this technique occurs if a job at the head of the queue requires more processors than are available, and blocks all other jobs behind it in the queue – even if there are enough processors to satisfy those job requests. This problem is referred to as head-of-line blocking.

EASY backfilling [24] addresses this problem by allowing other jobs to jump ahead in the queue, provided their executions do not delay the projected start time of the job at the head of the queue. EASY requires each job to be supplied with an estimated running time. Using these estimates, the scheduler can create a reservation time for the first waiting job, and also determine which jobs can be "backfilled". Variants of this technique, such as conservative backfilling or selective reservation have been studied, where the number of jobs with reservations varies [16, 31].

2.1.2 Utility-based approaches

The fundamental shortcoming of traditional approaches is rooted in the fact that they do not explicitly consider a user's utility for a job when making a scheduling decision. Job priorities were introduced as a way for users to distinguish important jobs from others [19, 31]. Priority-based systems typically employ between four and six priority queues, which restricts the amount of utility information a user can express for a job. Utility-based approaches extend the idea of priority by allowing a scheduler to prioritize jobs based on finer-grained information, such as per-job utility functions [7, 8, 12, 18, 23, 34].

Both priority and utility-based approaches assume the existence of a mechanism to prevent self-interested users from artificially overstating their job priority or utility. A special class of utility-based approaches, called *market-based approaches* directly incorporate such a mechanism. A well-designed market-based approach to resource allocation can provide a natural framework for users to communicate their job utility functions, and for the system to elicit accurate utility functions from users [6, 11, 36]. The basic idea is

that resources are priced by looking to balance global supply and demand, with users required to purchase resources using a personal budget of currency. In this case, resource prices force users to self-select a resource allocation based on their utility for jobs, with the implication being that the resources are allocated to the jobs that will generate the most value for the user.

We are concerned with a particular class of market-based approach which elicits utility information from the users rather than explicitly setting a price. In these systems, users are not price taking – instead, they provide utility functions as a statement for the valuation for resources, and based on these utility functions, the system (which controls access to resources) determines a resource schedule based on the users' utility functions. In order to create proper incentives for *truthful* utility information, the allocation algorithm can use incentive-compatible bidding rules [27] or pricing rules [2]. The various schemes for establishing these prices is outside the scope of this paper. Instead, we are concerned with the effect of uncertainty or wealth inequity on the allocation outcome.

2.2 User information error

A common theme among sophisticated job scheduling techniques is the use of additional job information to improve performance. A specific job property, such as estimated run time [33], priority [19] or utility function [3, 12, 23], is leveraged by the scheduler to optimize scheduling decisions for a particular performance criterion. However, it is well known that frequently such job information is either not immediately available to the scheduler, or in some cases, simply noisy [22]. Bailey Lee *et al.* find that even with an incentive (a raffle) for users to improve their estimates of job run-times, most users are *inherently* unable to do so [22]. Tsafrir *et al.* develop a model for characterizing the error in supercomputing users' job run-time estimates [32], and others have characterized the impact of this error in space-shared environments through simulation [10, 16, 25, 33], or in simpler environments using a theoretical approach [35].

3. MODELS

In this section, we describe how we model the entities in our experiments: the scheduling environment, job utility functions, and the information inaccuracy.

3.1 Job scheduling algorithm

Developing an effective and deployable utility-based scheduler is challenging and the subject of much prior work [3, 4, 12, 18, 23, 29]. In these algorithms, users submit a *utility function* along with each job, which conveys the utility, or value, the user will receive from the job, as a function of the job schedule (e.g., turnaround time for the job). The goal of the scheduler is to determine a schedule that maximizes the aggregate utility delivered to all users. In these systems, utility typically represents not only the value the user will extract from the job, but also the price she must pay to the system upon completion. Hence, rational users have an incentive to truthfully reveal their utilities.

However, even if resource demand (e.g., all jobs and their utility functions) is known a priori, calculating a utility-maximizing allocation is computationally intractable [20]. If resource demand is *not known* a priori, no on-line scheduling algorithm is guaranteed to perform within a factor v of the optimal offline algorithm (where v is the *value density* between the most important and least important job [4]). As a result, we expect any practical approach to rely on a heuristic scheduling algorithm.

In our simulations, we use the *FirstPrice* scheduling heuristic developed by Chun *et al.* [12], which is also the basis of the algorithm

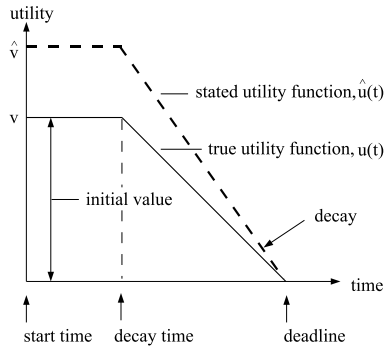


Figure 1: Job utility function (linear decay) as a function of v,$\hat{v}$.

used in Mirage allocation [11]. The basic idea of the heuristic is to calculate the value density for each job, which is the initial value of the utility function (i.e., value at time 0 in Figure 1), divided by its size and length. The heuristic performs a greedy "first-fit" for each job, ordered by value density. Our choice is motivated by the fact that FirstPrice performance has been analyzed thoroughly through a workload parameter sensitivity analysis [12] and verified in other studies [18, 29]. We defer considering extensions to the FirstPrice heuristic, such as a variable cost model like that of Popovici *et al.* [29], pre-emptible jobs [3], or a scenario where future gains are time-discounted, as in the study by Irwin *et al.* [18], to future work.

3.2 Utility functions

Prior work characterizes each job utility function by its initial value and its decay over time (see Figure 1). A study by Bailey Lee *et al.* [22] suggests that these two components are sufficient to characterize the time-varying utility of jobs from real users. However, lacking deployments from which to observe the values and decays of utility functions belonging to real system users, previous studies have relied on generating these components based on artificial values to drive their simulations [3, 12, 18, 29]. In contrast, we use workload data from the Mirage testbed [11] to create a distribution for initial values and job deadlines (both of which are contained in the job logs), and explore the effect of different types of utility decay (not contained in the job logs) in between the supplied initial value and job deadline information. In particular, for utility function decay, we examine the impact of the decay types observed by Bailey Lee *et al.* in their study of job utility functions among real users at the San Diego Supercomputer Center (SDSC) [22]. Note, for the remainder of the paper, we will use the terms *value*, *valuation* and *utility* interchangeably.

3.3 Information inaccuracy

We capture the potential fragility of a utility-based scheduling approach by considering the impact of inaccuracies in the user-provided utility information. Specifically, we model two types of inaccuracy commonly associated with utility information: valuation uncertainty, and wealth inequity. These types of inaccuracy are challenging to model because there is very little data available that describes how these inaccuracies may manifest in practice. Therefore, we create a parameterized model that can capture a range of possibilities associated with each type of inaccuracy. Namely, for a given utility function $u(t)$, we create a perturbed utility function $\hat{u}(t)$ for each of these error types, and parameterize the *magnitude* of perturbation by the variable k. For the purposes of this study, we ignore possible inaccuracies in job deadlines as, in many cases, job

deadlines are externally imposed upon the user. We defer investigation of misstated deadlines to future work.

3.3.1 Uncertainty

A first source of inaccuracy has to do with uncertainty. In many cases, the output (or utility) of a job depends upon the completion of other jobs [1, 3] and therefore a user is unable to determine its value prior to submission. More generally, a user simply may not be able to express her utility for a job due to cognitive or computational complexity [21, 28]. Therefore, we expect many honest and well-intentioned users to express job valuations that are weakly correlated or uncorrelated with their true valuations.

We assume that each job's true utility function has a start value v that is drawn from a fixed distribution V. In our simulations, V is the distribution of per-node-hour valuations from the workload trace. To model uncertainty we generate a perturbed value, $\hat{v}$, which represents the start value of a user's *stated* utility function (Figure 1). We define the parameter $k \in [0, 1]$ as the level of uncertainty, with larger values of k representing less certainty. For example, $k = 0$ indicates absolute certainty (i.e., *no* uncertainty), and $k = 1$ implies absolute uncertainty. If $k = 0$, we have the property that $\hat{v} = v$, and for $k = 1$, $\hat{v}$ will be uncorrelated with v.

The key challenge in defining a model for uncertainty is remaining independent of the distribution V; this is important in case V exhibits significant skew, as is the case for the value distribution in Mirage. Our procedure for doing this is as follows. Define the function $CDF : V \rightarrow [0, 1]$ as map from a value in v to its percentile within the distribution V, and its inverse function, $CDF^{-1} : [0, 1] \rightarrow V$. Therefore, if we draw $v \in V$, and v is in the 75th percentile of all values in V, then $CDF(v) = 0.75$, and $CDF^{-1}(0.75) = v$.

For a given value v, we find its percentile in V (as a fraction) $f = CDF(v)$. From f, we draw $\hat{f} \in Gaussian(\mu = f, \sigma = k/2)$. We determine our perturbed value $\hat{v}$ from the inverse CDF as $\hat{v} = CDF^{-1}(\hat{f})$. By using a Gaussian distribution with mean f, and standard deviation $k/2$, we have an intuitive way to understand how the value $\hat{v}$ deviates from v. For example, if $CDF(v) = 0.5$, this means that there is a "good chance" (from the definition of a Gaussian distribution, this means 68%) that $\hat{v}$ will be within k standard deviations of the percentile of v.

Figure 2 illustrates a sampled distribution when choosing $CDF(\hat{v})$ from $CDF(v)$. We see that when $k = 0$, $CDF(\hat{v})$ is exactly $CDF(v)$ with probability 1, and as k approaches 1, $CDF(\hat{v})$ is drawn from an increasingly noisy distribution centered at $CDF(v)$. Note that since a Gaussian distribution is unbounded, we clip $\hat{f}$ such that $\hat{f} \in [0, 1]$. As we can see from Figure 2 ($CDF(v) = 0.5$), the end points are raised for values 0 and 1 due to the effects of clipping. However, since clipping is significant only as k approaches 1, we do not expect it to adversely affect our results.

3.3.2 Wealth inequity

A second source of inaccuracy has to do with a user's wealth. All utility-based scheduling systems assume the existence of a currency – either real or virtual – with which users will express their value, or willingness to pay for a job. In either scenario, we assume that a user with a larger budget of currency will receive a larger share of resources, regardless of the true value derived from that job. For example, a user with a disproportionately large share of wealth has more "disposable income", and is therefore, more likely to forgo an additional dollar than a user with comparatively less wealth (this a consequence of the Diminishing Marginal Utility of Wealth). For systems that use an *artificial currency* (e.g., super-

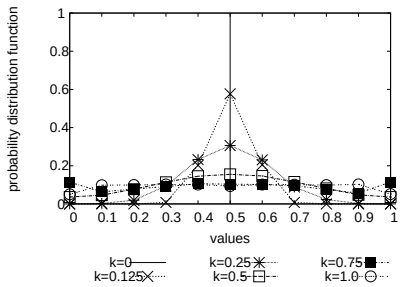


Figure 2: Generating $\hat{v}$ from v for uncertain users.

$\{w_1, w_2, w_3, w_4, w_5\}$	Gini coefficient (k)
$\{1, 1, 1, 1, 1\}$	0
$\{1, 1, 1, 1, \epsilon\}$	0.2
$\{1, 1, 1, \epsilon, \epsilon\}$	0.4
$\{1, 1, \epsilon, \epsilon, \epsilon\}$	0.6
$\{1, \epsilon, \epsilon, \epsilon, \epsilon\}$	0.8
$\{1, \epsilon, \epsilon, ...\epsilon, \epsilon, ...\}$	1

Table 1: User wealth inequity. Here ϵ is approximately zero. The last table row corresponds to a population larger than 5 to illustrate how wealth inequity can approach 1.

computing service units, or virtual currency in Mirage), this effect is exaggerated since *all* users will be motivated to spend their remaining balance on their remaining workloads, since the currency can only be redeemed for this purpose.

To model this behavior, we define a wealth parameter, $w_i > 0$, for each user i, with the property that $w_i > w_j$ indicates that user i has more wealth, or ability to pay, than user j. For any user i, we generate $\hat{v}$ by scaling v by w_i (i.e., $\hat{v} = w_i \cdot v$).

Similar to the case with uncertain users, we use parameter k to indicate the level of wealth inequity across users in the system. For $k = 0$, we have $w_i = w_j$ for all users i, j, and for $k = 1$, nearly all wealth is held by a single user. There is a formal economic metric used to describe such wealth (im)balance in a society: the *Gini* coefficient [5]. This coefficient is defined as a fraction between 0 and 1 where 0 represents perfect wealth *equality* and 1 indicates perfect *inequality*. Therefore, we simply define k as the Gini coefficient of the wealth coefficients in the population. Table 1 contains the Gini coefficient for various values of w_i in a given population.

Finally, given $\hat{v}$ as a result of user uncertainty or wealth inequity, we generate a user's stated utility function, $\hat{u}(t)$, by scaling the non-decaying portion of a user's utility function by $\hat{v}/v$, and decay the job from $\hat{v}$ to the original deadline in the same manner as the original decay. Figure 1 illustrates such an example for a linear-decay utility function.

4. SIMULATIONS

We quantify the performance of a utility-based scheduling approach *relative* to traditional, non-utility-based approaches under various operating conditions—particularly in the face of inaccurate utility information. We are primarily interested in two metrics: the *aggregate utility* delivered to users, and the *distribution of utility* across its users. Our primary goal is to identify how much uncertainty or wealth inequity a utility-based approach can tolerate before degenerating. Secondarily, we seek to determine whether such levels of uncertainty arise in real workloads.

	Light			Loaded		
Source,Category	**Mean**	**Minimum**	**Maximum**	**Mean**	**Minimum**	**Maximum**
Mirage,Value (per node-hour)	0.16	0.0025	2.5	1.5	3e-4	111
Mirage,InterArrival (hr)	18	0	44	8.6	0	24
Mirage,Deadline (hr)	81	0	359	133	0	359
SDSC,Value (per node-hour)	0.002	4.7e-5	1.84	0.001	1e-9	1.84
SDSC,InterArrival (hr)	0.317	0	22	0.312	0	22
SDSC,Deadline (hr)	231.8	0	40209	274	0	43238

Table 2: Workload characteristics for different demand regimes. Deadline represents a relative offset from job submission time.

4.1 Simulator

We have constructed a concurrent simulation environment to mimic a generic parallel computing cluster and use it to compare the performance of the FirstPrice utility-based scheduling policy against that of two traditional scheduling policies: first-come, first-served (FCFS) ordering with EASY backfilling [24], and a priority queue that uses four levels of priority. While neither the FCFS+backfill nor the priority scheduler explicitly considers job deadlines, we assume that jobs that pass their deadlines are canceled or otherwise removed from the queue such that neither algorithm will consider scheduling a job that cannot complete before its deadline passes.

4.2 Workloads

Our simulation experiments are driven by workload traces from the Mirage sensor-network testbed at Intel Research Berkeley [11], and the SDSC-SP2 cluster [14].

The Mirage workload contains fifteen months of usage data (July 2006 to October 2007). The testbed consists of 150 nodes, and is shared among dozens of academic users primarily as a research platform for conducting sensor-network experiments. Since early 2005, node allocations are performed by auction, where users bid for time slots on particular node configurations using a supplied budget of virtual currency. As such, Mirage presents a unique case study since it contains data about utility functions of real users.

The simulated workload uses job characteristics—e.g., size, length, utility and deadlines, as well as arrival characteristics (e.g., demand, inter-arrival times)—from the distributions observed in the 15-month usage trace. Table 2 contains example values of these job characteristics.

Workloads from previous simulation studies of similar utility-based scheduling algorithms often use a bi-modal distribution of valuable and less-valuable jobs, and a bi-modal distribution of urgent and less-urgent jobs [3, 12, 18, 29]. In comparison, the workload data in Mirage exhibits an even wider distribution of job valuations and deadlines than those used in previous studies.

Job arrival patterns are highly variable in the Mirage trace, making it difficult to draw conclusions from the trace as a whole. Instead, we partition the trace data into two operating regimes based upon short-term demand: *light* and *loaded*. We define the light regime to be any time period where almost all incoming requests can be completely satisfied by available system capacity; in other words, total resource demand can satisfied on average. (This regime includes many instances where the system is completely idle.) In the loaded regime, incoming resource demand exceeds system capacity. For each operating regime, we extract all job requests and calculate the distribution of inter-arrival times, bid sizes, lengths, patience and values and simulate the results separately. Table 2 contains a few workload values for each demand regime. Note that bid values and arrival rates are larger in the loaded regime than the light regime. However, jobs appear to be less urgent during the loaded regime. The increased patience seems to indicate additional planning by users in order to use the system.

To account for the possibility of hidden demand or future peak usage, we create a synthetic operating regime that we call *extreme*; it maintains the same job distribution as the loaded regime, but with double the number of requests (i.e., inter-arrival times are halved).

Although the Mirage trace reports each user's job valuation and deadline, it contains no information about how the utility of the job decays over time. We consider the following scenarios to capture the possibilities of job decay (inspired by study of SDSC users from Bailey *et al.* [22]):

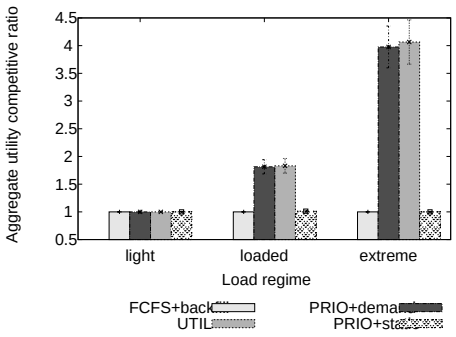
- *convex*: Job utility decays as a convex curve from arrival time until the deadline.
- *linear*: Job utility decays linearly from arrival time+run time until the job deadline (Figure 1). This model is used frequently in previous work [3, 12, 18, 29].
- *flat*: Job utility does not decay until the deadline, at which point it drops instantaneously to zero.
- *mix*: Job utility decays as either *flat*, *linear*, or *convex*. Each type is equally probable.

The SDSC-SP2 trace contains 24 months of job submissions (April 1998-2000). We partition load and extract job distribution data from the trace in the same way as the Mirage data. Unlike the Mirage workload, the SDSC-SP2 cluster does not contain fine-grained job valuation information, so we rely on the tool used by Bailey *et al.* [22] to generate utility functions for the simulations using the SDSC-SP2 trace. This tool generates a utility function for each job in a given workload log based upon the priority queue it was submitted to. More details about this tool can be found in the authors' paper. Table 2 contains example workload values for each demand regime extracted from the SDSC-SP2 trace.

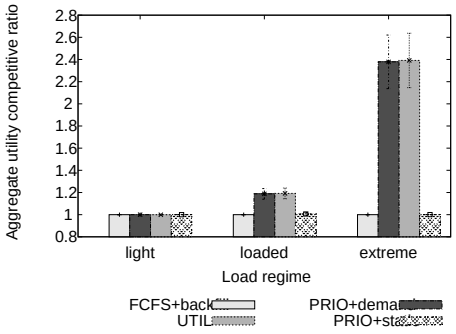
4.3 Schedulers

In addition to the Mirage FirstPrice scheduler, we consider two alternatives. As a baseline, we simulate FCFS+backfill, a traditional, non-utility based scheduler. In addition, we study a simple four-level priority scheduler modeled on the SDSC scheduler, which can be viewed as a simplistic form of utility-based scheduling (that captures starting value, but ignores decay). At SDSC, users are charged for their jobs in accordance with the priority level they assign, so there is motivation to accurately classify job priority.

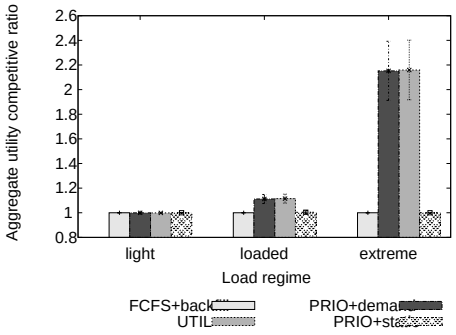
In our baseline experiments, we consider two configurations of the priority scheduler: static (*PRIO+static*) and dynamic (*PRIO+demand*). The former uses the statically defined priorities observed in SDSC trace logs. Specifically, rates are fixed at $0.5x$, $1x$, $2x$, and $1.8x$ service units per CPU hour, respectively, for each of the four increasing priority levels. (Note the highest priority is reserved for "express" jobs which are restricted in their duration,



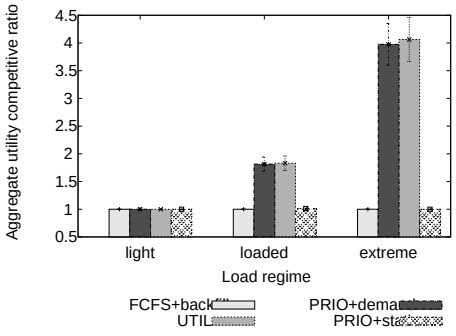
(a) Convex utility function decay.



(b) Linear utility function decay.

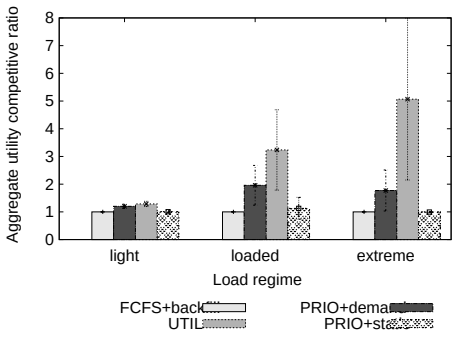


(c) Flat utility function decay.

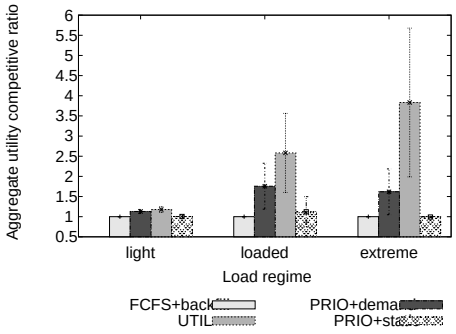


(d) Mixed utility function decay.

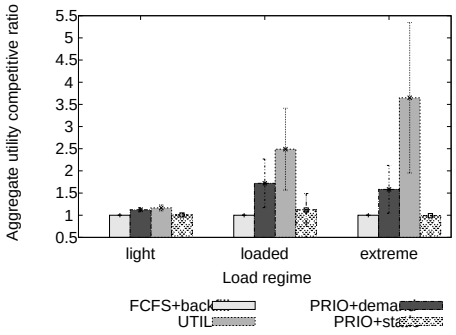
Figure 3: Mirage: Baseline competitive ratio.



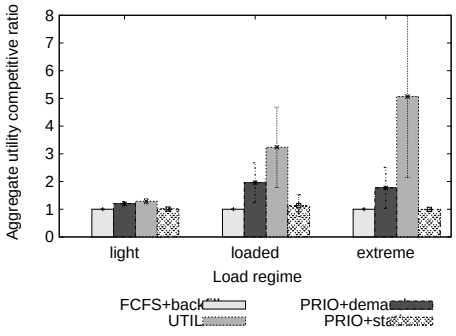
(a) Convex utility function decay.



(b) Linear utility function decay.



(c) Flat utility function decay.



(d) Mixed utility function decay.

Figure 4: SDSC-SP2: Baseline competitive ratio.

hence the lower cost.) If we define priority level $1x$ to be the median job value in our workload, then jobs with values twice the value of the median have priority level $2x$, and sufficiently small jobs with at least $1.8x$ the value of the median qualify for the express queue, or highest priority.

In both workloads however, most of the important jobs are several orders of magnitude more valuable than less important jobs, so the static priority settings do not appropriately segregate jobs of different values. Therefore, we also consider a dynamic approach (PRIO+demand), which classifies job priority based upon

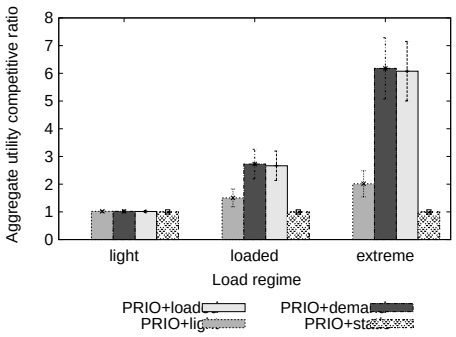


Figure 5: Mirage: Competitive ratio using different priority-based schedulers (mix utility function decay).

observed demand. Specifically, we determine the appropriate priority level for each job based on its value and the total distribution of job values in the workload. We use the Expectation Maximization (EM) algorithm to decompose the distribution of observed values into four separate Gaussian distributions, each with its own weight, mean and standard deviation. Each of the component distributions represents a single priority level (e.g., the distribution with the highest mean represents jobs with the highest priority level), and based on this decomposition, the *PRIO+demand* algorithm determines to which priority level a job corresponds.

4.4 Baseline performance

In order to calibrate our study with previous studies, we quantify the baseline (i.e., assuming perfect information) performance improvement of utility-based scheduling over a non-utility-based approach under each workload.

This experiment measures the *competitive ratio* (measured as a ratio of aggregate utility delivered by a scheduler to the FCFS+backfill scheduler) when user-provided information is completely accurate. The results for each workload are plotted in Figures 3 and 4 (unless otherwise noted, standard deviation bars are depicted along with the averages). As expected, the performance for all the schedulers is the same under light conditions, regardless of utility function decay or workload.

However, under *loaded* demand, the competitive ratio for the FirstPrice scheduler (*UTIL*) and *PRIO+demand* are similar, and range from 1.1 to 1.8 in the Mirage workload. In the SDSC workload, we see that the competitive ratio for *UTIL* ranges from 2.5 to 3, and *PRIO+demand* ranges from 1.75 to 2. For the *extreme* demand regime, the competitive ratios range from 2.2 to 4 in Mirage, and from 3.5 to 5 in SDSC-SP2, depending on utility decay.

In the Mirage workload, the median job size is 100 (out of 100 possible nodes), and in the SDSC-SP2 workload, the median job size is 5 (out of 128 possible nodes). Therefore, we can account for the difference in competitive ratios across workloads by recognizing that there are more scheduling decisions to make in the SDSC-SP2 workload, which provides an opportunity for the *UTIL* and *PRIO* schedulers to make a "smart" decision, and at the same time, for the *FCFS* scheduler to make an unwise decision.

Interestingly, the *PRIO+static* algorithm performs quite poorly compared to the other utility-based algorithms. In both workloads, it is unable to effectively differentiate the highest-valued jobs, and therefore performs about as well as the FCFS+backfill algorithm. In Figure 5, we plot the baseline performance of the *PRIO+static* algorithm against algorithms which take into account the distribution of value densities in job workload. For example *PRIO+light* establishes 4 levels of priority based on value-density observed during light demand, and *PRIO+demand* established priority levels based on value-density observed during the *entire* trace. We see that a priority-based algorithm *must* take into account the value density of the incoming job stream in order to approach the performance of the utility-based approach.

The preceding graphs illustrate the overall allocation efficiency in the system, but does not consider how utility is distributed among users. We define *fairness* to be a measure of the utility share received by each user. Formally, we measure the fraction of utility received by a user divided by the maximum possible utility he *could have* received from all of his submitted jobs. In Figures 6 and 7, we see the average, minimum and maximum utility share received by each user (the error bars represent minimum and maximum).

Interestingly, while all algorithms schedule roughly the same fraction of jobs for each user (not shown for space considerations, but 80% for light demand and 25% for loaded), both utility-based approaches (UTIL and PRIO+demand) schedule jobs with $2-3x$ (for SDSC-SP2) to $1.8-4.5x$ (for Mirage) more value to the average user, and $1.45-7x$ to the minimum (worst-case) user. In general, the relative improvement of utility and priority scheduling decreases (and fairness increases) as the utility function decay moves from convex to linear to flat. In the interest of space, we typically present results for the middle ground—mixed decay—moving forward. Similarly, because *PRIO+static* is unable to effectively differentiate the high-value jobs (Figure 5), it performs quite similarly to FCFS+backfill; hence, we plot only the *PRIO+demand* scheduler for the remainder of the simulations.

4.5 Information inaccuracy

In our next set of experiments, we measure the impact of imperfect utility information on the competitive ratio. We consider the impact of each type of information inaccuracy separately.

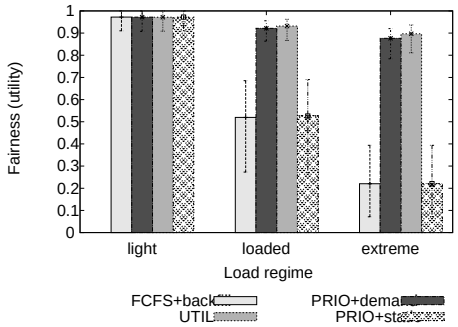
4.5.1 User uncertainty

In Figures 8 (Mirage workload) and 9 (SDSC workload), we plot the competitive ratio under user uncertainty as a function of k (degree of user uncertainty). We see that again, for a lightly loaded system, there is effectively no difference among the schedulers. However, as demand increases, the impact of uncertainty on the competitive ratio depends directly on the utility function decay. In fact, without sufficient demand, a linear or flat decay and $k = 0.75$ uncertainty renders both priority or utility-based scheduling approaches 20% *less* effective than a FCFS+backfill approach in the Mirage workload. In the SDSC workload, uncertainty of $k = 0.125$ can drop the competitive ratio of the utility-based approaches by a factor of $2-3$, depending on utility decay. Note that while the SDSC workload provided more opportunity to deliver utility to its users (i.e., higher competitive ratio than in Mirage workload for baseline experiments), it is also more vulnerable to error from user uncertainty.

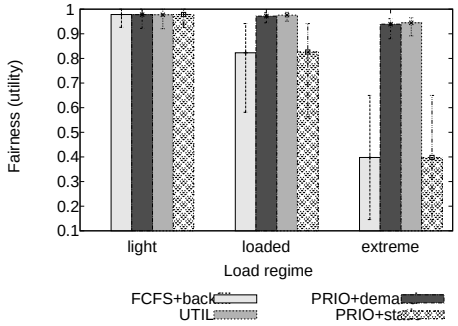
All schedulers are effectively fair under light load (light loads not shown for the remainder of this paper for space considerations).

For the loaded regime, however, we see that when k exceeds 0.5, that the utility-based approaches can be, on average (and max-min), *less fair* than the traditional approach, when utility decay is linear or flat (Figure 10(a) for Mirage and 11(a) for SDSC).

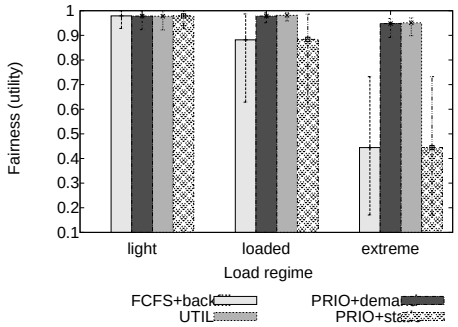
For the extreme regime, this breaking point occurs even earlier, at $k > 0.25$ (Figure 10(b) for Mirage and Figure 11(b) for SDSC). Again, as with the competitive ratio for aggregate utility, we see that generally, as user certainty decreases, the user utility-share also decreases, and that the uncertainty has a much more pronounced effect in the SDSC-SP2 workload since there are more scheduling opportunities for error.



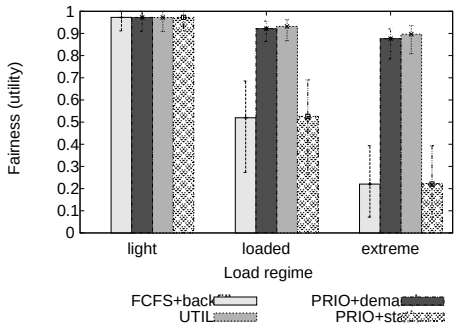
(a) Convex utility function decay.



(b) Linear utility function decay.

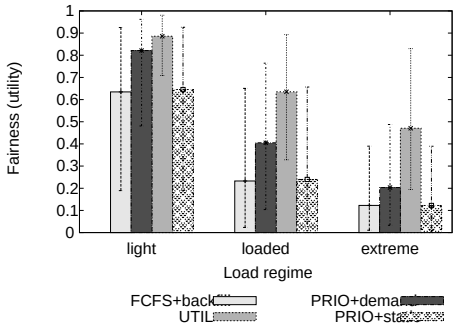


(c) Flat utility function decay.

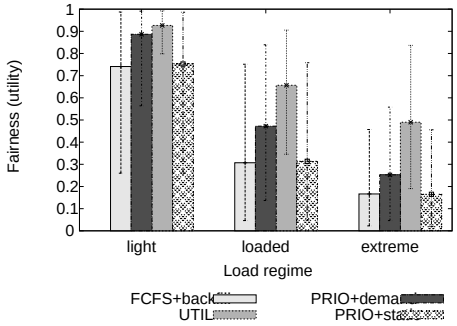


(d) Mixed utility function decay.

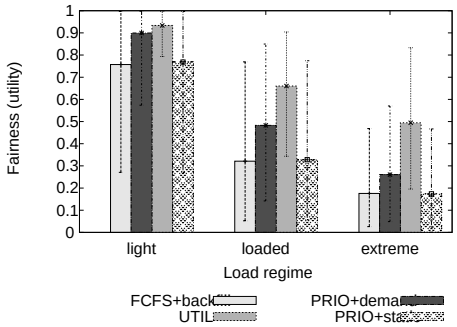
Figure 6: Mirage: Baseline *utility* fairness.



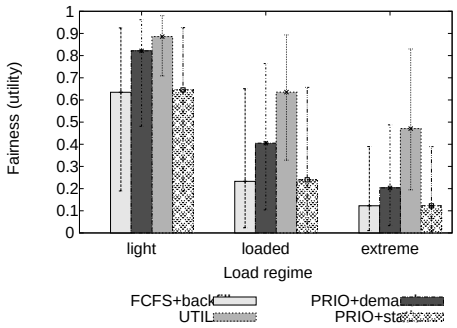
(a) Convex utility function decay.



(b) Linear utility function decay.



(c) Flat utility function decay.



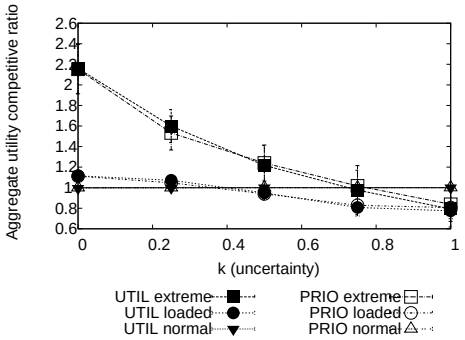
(d) Mixed utility function decay.

Figure 7: SDSC-SP2: Baseline *utility* fairness.

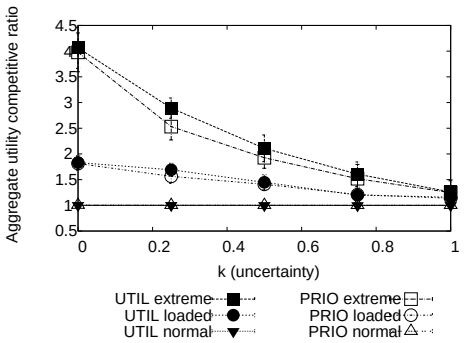
4.5.2 Wealth inequity

This section considers the impact of wealth inequity between users on a utility-based approach. In these experiments, the degree to which users have different levels of wealth is parameterized by the variable k. Figures 12(a) (Mirage) and 12(b) (SDSC) show the aggregate utility competitive ratio for each utility-based scheduler as a function of k .

Similar to the scenario with uncertain users, there is effectively no difference in *aggregate utility* among the scheduling approach for a lightly loaded system, but as demand increases, the com-

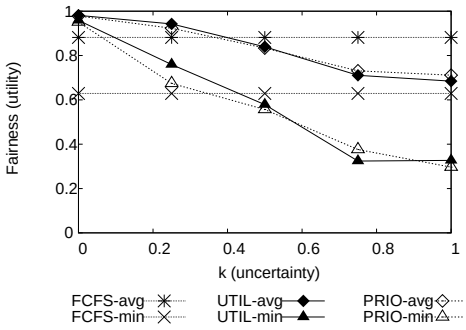


(a) Flat utility function decay.

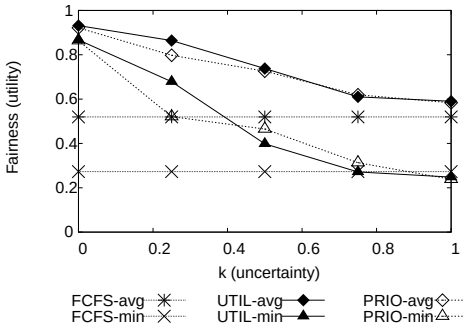


(b) Mix utility function decay.

Figure 8: Mirage: Competitive ratio with uncertain users.

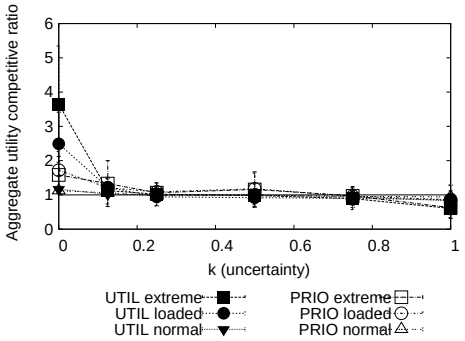


(a) Loaded demand and mix utility function decay.

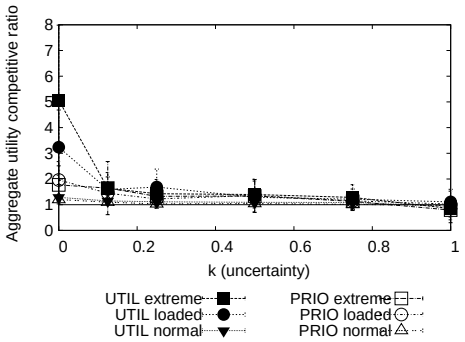


(b) Extreme demand and mix utility function decay.

Figure 10: Mirage: Fairness with user uncertainty.

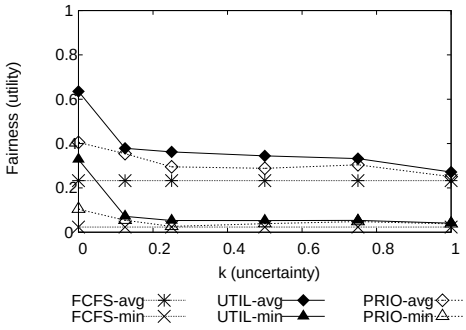


(a) Flat utility function decay.

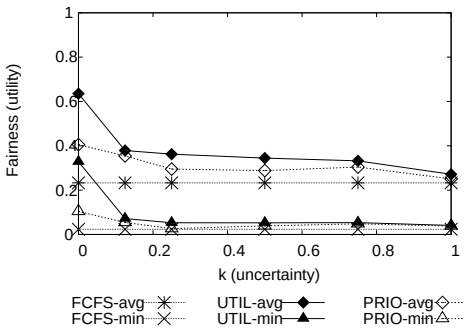


(b) Mix utility function decay.

Figure 9: SDSC-SP2: Competitive ratio with uncertain users.

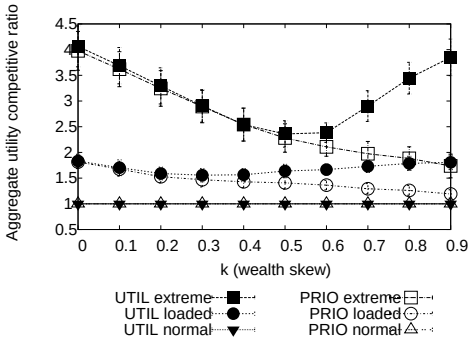


(a) Loaded demand and mix utility function decay.

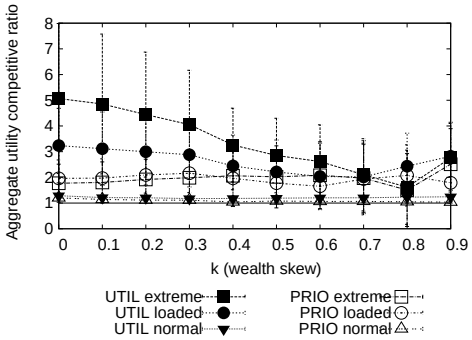


(b) Extreme demand and mix utility function decay.

Figure 11: SDSC-SP2: Fairness with user uncertainty.

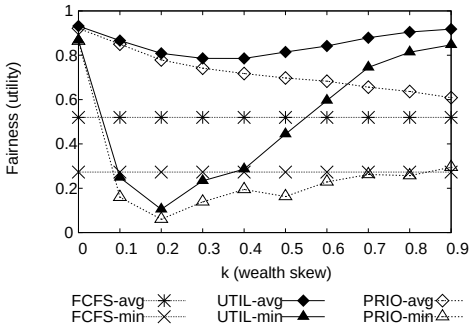


(a) **Mirage:** Mix utility function decay.

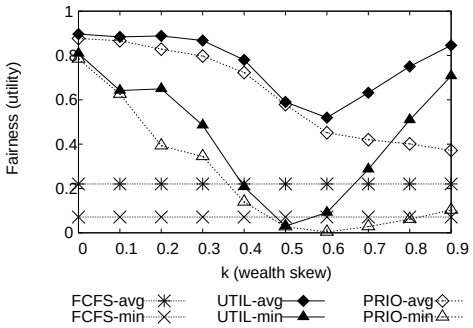


(b) **SDSC-SP2:** Mix utility function decay.

Figure 12: Competitive ratio under wealth inequity.



(a) Loaded demand and mix utility function decay.



(b) Extreme demand and mix utility function decay.

Figure 13: Mirage: Fairness with wealth inequity.

petitive ratio for both utility-based schedulers increase, depending upon the steepness of utility function decay. For the Mirage workload, the competitive ratio for *UTIL* incurs its minimum at $k = 0.3$ (loaded) and $k = 0.5$ (extreme), while *PRIO* incurs its minimum as k approaches 1. To understand why the worst-case performance for *UTIL* occurs at a different value of k from that of *PRIO*, consider the difference between a scenario with $k = 0.5$ and $k = 0.9$. With $k = 0.5$, half of the users hold most of the wealth in the system, and they are able to consume most of the available resources. However, as k increases to 0.9, more of the wealth is held by a single user, in which case he does not consume most of the resources. The remaining resources are scheduled among the remaining users who are of the same wealth level. Therefore with *UTIL*, the jobs of these users will be prioritized based upon expressed utility, however with *PRIO*, all of the jobs are classified as having the same (low) priority, and thus scheduled in order of arrival, without regards to job value. For the SDSC workload, the competitive ratio for *UTIL* incurs its minimum at $k = 0.7$ (loaded) or $k = 0.8$ (extreme).

In Figures 13(a) through 14(b), we plot the minimum and average utility share for different demands. Generally, we see that the *average* utility share received by each user is no worse with either utility-based scheduler. However, we also see that the *minimum* utility share may be lower depending on k. Specifically, in Figure 13(a) (Mirage), we see that for $k = 0.2$, the minimum utility share received by the worst-case user is less than half of that received by the worst-case user in the FCFS approach. In the corresponding graph for the SDSC workload (Figure 14(a)), we see that for $k < 0.9$, the worst-case user gets *no* utility share. As mentioned in the user uncertainty simulations, the SDSC-SP2 workload presents more scheduling opportunities for a given load, and therefore more opportunities for error. Therefore, we expect that the effect of wealth inequity is more pronounced than in Mirage.

4.5.3 Simultaneous information inaccuracy

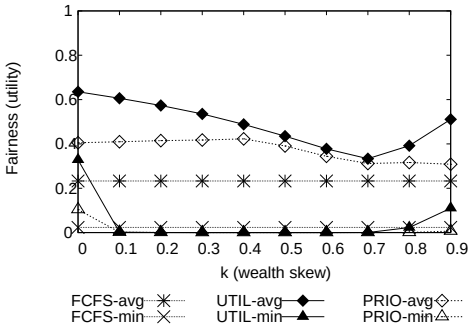
Finally, we examine the impact of simultaneous user uncertainty and wealth inequity. In Figure 15, we observe that user uncertainty has a larger impact on the competitive ratio than wealth inequality. For steep utility function decays (only *mix* shown), *UTIL* is robust to both forms of inaccurate information, and *PRIO*, while robust to most forms of inaccuracy, exhibits performance that is sensitive to the incoming resource demand. Only in the scenario with little or no decay (not shown for space considerations) and uncertainty with $k > 0.5$ does a FCFS+backfill approach deliver more value than a utility-based or priority approach.

4.6 Discussion

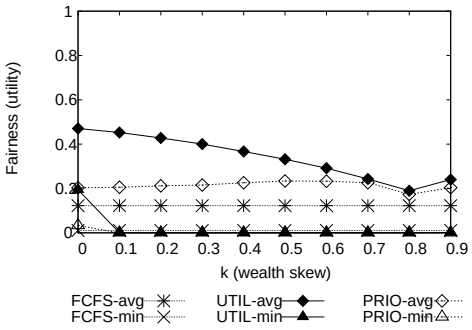
Our simulation results suggest that the robustness of a utility-scheduling approach depends significantly upon the distribution of utility function decay and the assumed value of k for information inaccuracy. In this section, we consider which decay model and values of k might be expected in practice. The study of SDSC users by Bailey Lee *et al.* [22] suggests that the *mix* decay most accurately reflects reality, particularly for the SDSC workload. Unfortunately, there is little available data to suggest what values of k occur in production systems.

For user uncertainty in particular, it is not possible to determine k from available workload logs. However, accuracy information about other user-specified job characteristics can provide a comparison point. For example, studies show that user-provided job *run-time estimates* used for backfill algorithms are inherently inaccurate [22].[1] Using our definition of uncertainty from Section

[1]All of our presented simulations assume perfect run-time estimates. However, repeating all of the experiments using the actual run-time estimates from users does not significantly change the results; we omit these graphs for space considerations.



(a) Loaded demand and mix utility function decay.



(b) Extreme demand and mix utility function decay.

Figure 14: SDSC-SP2: Fairness with wealth inequity.

Cluster name	Number of jobs	k
SDSC SP2 [14]	54,006	**0.107**
SDSC BLUEHORIZON [14]	224,065	**0.165**
SDSC DATASTAR [14]	71,484	**0.111**

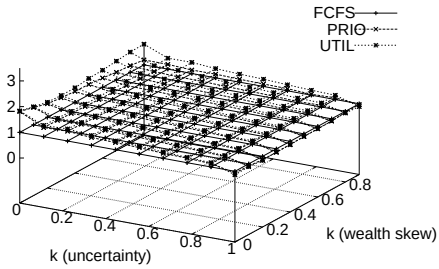
Table 3: Uncertainty in job run-time estimates.

3 we list the corresponding values of k for job run-time estimates for both the SDSC-SP2 workload considered here, as well as two other supercomputing workloads, in Table 3. The magnitude of k in all instances is within the threshold for user uncertainty that a utility-based scheduler can tolerate.

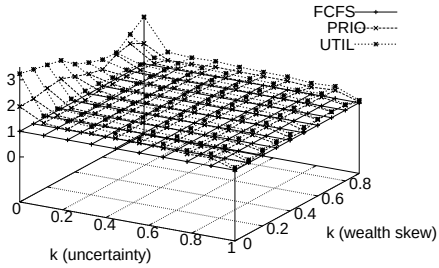
For the case of wealth inequity, we can use information about the distribution of virtual currency endowments to directly determine k. In Mirage, most users receive the same endowment of virtual currency, with a single group receiving twice as much as the others. Since the user population is restricted to fewer than 20 groups, this distribution results in an inequity level of $k = 0.09$, which is within the threshold for wealth inequity that a utility-based scheduler can tolerate. While we lack similar data for SDSC-SP2, we have data from other production clusters at SDSC which show that k ranges from 0.75 to 0.85 [13]. From Figure 12, we see that these values of k in SDSC-SP2 may adversely impact the performance of a utility-based scheduler. One way to mitigate the impact would be to limit the quantity of high-priority jobs across groups, or limit the accrual of service units by any group, as is done by the savings tax policy in Mirage [11]. This policy, in effect, lowers the average k over tax-collection periods.

5. CONCLUSIONS

Ours is the first study of which we are aware that not only quantifies the benefits of a utility-based scheduler under a real workload,



(a) **Mirage:** loaded demand.



(b) **SDSC-SP2:** loaded demand.

Figure 15: Competitive ratio with both uncertainty and wealth inequity, and users with *mix* decay.

but also considers the imperfect conditions likely to be present in real-world deployments. This study's primary goal is to advance the debate surrounding the practicality of using a utility-based scheduling approach in both current computing environments, and future environments, such as cloud computing systems.

In our simulations, we corroborate the findings of previous studies that a utility-based scheduler can *potentially* increase the value delivered to users when compared to a FCFS+backfill scheduler, and that the potential increase is at least 10–300% in real workloads. We also find that utility is distributed more equitably across users than in the case of a traditional approach, in both the average (80–450%) and minimum (45–700%) utility share per user.

Furthermore, we find that these results are surprisingly robust based upon projections from real world data. Despite a theoretical worst-case performance of up to 50% degradation when compared against a FCFS+backfill approach, we provide projections from information error in real systems that suggest a utility-based approach can indeed deliver more equitable values to users even under potentially imperfect operating conditions. Specifically, if we expect user utility information to be *at least* as accurate as observed inaccuracies in run-time estimates ($k < 0.2$ for all logs in Table 3), we can expect an increase in value of at least 20–100%.

The secondary goal of this study is to advance future work in this area by highlighting the important issues facing prospective deployments. While we may have assuaged concerns about the robustness of a utility-based system, concern may still exist about their usability and the possible difficulty of setting appropriate system policies [30]. Indeed, different market-inspired mechanisms have been proposed to alleviate these particular issues. For example, fixed resource prices has been proposed as an effective [36] and more usable [9] alternative to auctions. Also, virtual currency has been proposed as a mechanism to control allocation fairness

policies. However, as was demonstrated earlier, the effectiveness of these mechanisms are dictated by their ability to accurately extract user valuation: a priority-based scheduler can perform well *only if* queue charge rates reflect the incoming resource demand. Therefore any fixed-price scheme must accurately elicit user valuation information in order to appropriately set prices. Likewise, user incentives for spending and saving real currency may not be the same under virtual currency, it is unclear how effectively a virtual currency policy elicits job valuation information from real users.

Understanding the sensitivity of information-elicitation accuracy in pricing mechanisms and monetary policies for a utility-based allocation system provides important insight into designing a practical and effective allocation system for existing and future computing environments.

Acknowledgements

The authors are indebted to Cynthia Bailey Lee, Allan Snavely, and the anonymous reviewers for their feedback on earlier drafts. This work is funded in part by the UCSD Center for Network Systems (CNS), NSF CAREER grant CNS-0347949, and Qualcomm through the UC Discovery program.

6. REFERENCES

[1] E. Anderson, D. Beyer, K. Chaudhuri, T. Kelly, N. Salazar, C. Santos, R. Swaminathan, R. Tarjan, J. Wiener, and Y. Zhou. Value-maximizing deadline scheduling and its application to animation rendering. In *Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures*, July 2005.

[2] L. M. Ausubel and P. Milgrom. *The Lovely but Lonely Vickrey Auction*, chapter 1. MIT Press, 2006.

[3] A. AuYoung, L. E. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *Proc. 15th IEEE Int'l Symp. High Performance Distributed Computing*, June 2006.

[4] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the Competitiveness of On-line Real-time Task Scheduling. *Real-Time Systems*, 4(2):125–144, 1992.

[5] J. Benhabib and A. Bisin. The distribution of wealth and redistributive policies. Meeting Papers 368, Soc. for Economic Dynamics, Dec. 2006.

[6] A. Byde, M. Sallé, and C. Bartolini. Market-Based Resource Allocation for Utility Data Centers. Technical Report HPL-2003-188, Hewlett Packard Laboratories, September 2003.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. 18th ACM Symp. Operating Systems Principles*, October 2001.

[8] K. Chen and P. Muhlethaler. A Scheduling Algorithm for Tasks Described by Time Value Function. *Real-Time Systems*, 10(3):293–312, May 1996.

[9] A. Chernev. Reverse pricing and online price elicitation strategies in consumer choice. *J. Consumer Psychology*, 13(1&2):51–62, 2003.

[10] S.-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon. The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance. In *Proc. 8th Workshop Job Scheduling Strategies for Parallel Processing*, July 2002.

[11] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A Microeconomic Resource Allocation System for SensorNet Testbeds. In *Proc. IEEE Workshop Embedded Networked Sensors*, May 2005.

[12] B. N. Chun and D. E. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers. In *Proc. 2nd IEEE Int'l Symp. Cluster Computing and the Grid*, May 2002.

[13] T. Duffield, D. Hart, and N. Wolter. Data obtained through personal communication. San Diego Supercomputing Center, 2009.

[14] D. Feitelson. Parallel Workloads Archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[15] D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel Job Scheduling – A Status Report. In *Proc. 10th Workshop Job Scheduling Strategies for Parallel Processing*, June 2004.

[16] D. G. Feitelson. Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study. *IEEE Trans. Parallel and Distributed Systems*, 16(2):175–182, Feb. 2005.

[17] D. Grosu. AGORA: an architecture for strategyproof computing in grids. In *Proc. 3rd Int'l Workshop Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, July 2004.

[18] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing Risk and Reward in a Market-Based Task Service. In *Proc. 13th IEEE Int'l Symp. High Performance Distributed Computing*, pages 160–169, June 2004.

[19] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Proc. 7th Workshop Job Scheduling Strategies for Parallel Processing*, June 2001.

[20] T. Kelly. Utility-directed allocation. In *Proc. 1st Workshop Algorithms, Architectures for Self-Managing Systems*, July 2003.

[21] K. Larson and T. Sandholm. Costly Valuation Computation in Auctions. In *Proc. 8th Conf. Theoretical Aspects of Rationality and Knowledge*, pages 169–182, July 2001.

[22] C. B. Lee and A. Snavely. On the User-Scheduler Dialogue: Studies of User-Provided Runtime Estimates and Utility Functions. *Int'l J. High Performance Computing Applications*, 20(4):495–506, 2006.

[23] C. B. Lee and A. E. Snavely. Precise and Realistic Utility Functions for User-centric Performance Analysis of Schedulers. In *Proc. 16th IEEE Int'l Symp. High Performance Distributed Computing*, June 2007.

[24] D. A. Lifka. The ANL/IBM SP Scheduling System. In *Proc. 1sd Workshop Job Scheduling Strategies for Parallel Processing*, Apr. 1995.

[25] A. W. Muálem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans. Parallel and Distributed Systems*, 12(6):529–543, June 2001.

[26] C. Ng, D. Parkes, and M. Seltzer. Strategyproof Computing: Systems Infrastructures for Self-Interested Parties. In *Proc. 1st Workshop Economics of Peer-to-Peer Systems*, June 2003.

[27] D. C. Parkes, R. Cavallo, N. Elprin, A. Juda, S. Lahaie, B. Lubin, L. Michael, J. Shneidman, and H. Sultan. ICE: An iterative combinatorial exchange. In *Proc. 6th ACM Conf. Electronic Commerce (EC 05)*, pages 249–258, June 2005.

[28] D. C. Parkes, L. H. Ungar, and D. P. Foster. Accounting for Cognitive Costs in On-Line Auction Design. *Lecture Notes in Computer Science*, 1571:25–40, 1998.

[29] F. I. Popovici and J. Wilkes. Profitable Services in an Uncertain World. In *Proc. ACM/IEEE Conf. Supercomputing*, page 36, Nov. 2005.

[30] J. Shneidman, C. Ng, D. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun. Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems. In *Proc. 10th USENIX Workshop Hot Topics in Operating Systems*, June 2005.

[31] D. Talby and D. Feitelson. Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *Proc. 13th Int'l Parallel and Distributed Processing Symp.*, Apr. 1999.

[32] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling User Runtime Estimates. In *Proc. 11th Workshop Job Scheduling Strategies for Parallel Processing*, June 2005.

[33] D. Tsafrir and D. G. Feitelson. The Dynamics of Backfilling: Solving the Mystery of Why Increased Inaccuracy May Help. In *2006 IEEE Int'l Symp. Workload Characterization*, Oct. 2006.

[34] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility Functions in Autonomic Systems. In *Proc. 1st Int'l Conf. Autonomic Computing*, May 2004.

[35] A. Wierman and M. Nuyens. Scheduling despite inexact job-size information. In *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems*, June 2008.

[36] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-Commerce: Market Formulations Controlling Resource Allocation on the Computational Grid. In *Proc. 15th Int'l Parallel and Distributed Processing Symp.*, Apr. 2001.

[37] C. S. Yeo and R. Buyya. A Taxonomy of Market-based Resource Management Systems for Utility-driven Cluster Computing. *Software: Practice and Experience*, 36(13):1381–1419, Nov. 2006.