Analyzing and Minimizing the Impact of Opportunity Cost in QoS-aware Job Scheduling*

M. Islam† P. Balaji‡ G. Sabin† P. Sadayappan†

†Computer Science and Engg., Ohio State University
{islammo, sabin, saday}@cse.ohio-state.edu

‡Math. and Comp. Science, Argonne National Lab.
balaji@mcs.anl.gov

Abstract

Quality of service (QoS) mechanisms that allow users to request turn-around time guarantees for their jobs have recently generated much interest. In our previous work we had designed a framework, *QoPS*, to allow for such QoS. This framework provides an admission control mechanism that only accepts jobs whose requested deadlines can be met, and guarantees that these deadlines are met. However, the framework is completely *blind* to the revenue these jobs can fetch for the supercomputer center. By accepting a job, the supercomputer center might relinquish its capability to accept some future arriving (and potentially more expensive) jobs. In other words, while each job pays an explicit price to the system for running it, the system may also be viewed as paying an implicit *opportunity cost* by accepting the job. Thus, accepting the job is profitable when the job's price is higher than its opportunity cost. In this paper we analyze the impact such opportunity cost can have on the overall revenue of the supercomputer center and attempt to minimize it through predictive techniques. Specifically, we propose two extensions to QoPS, *Value-aware QoPS (VQoPS)* and *Dynamic Value-aware QoPS (DVQoPS)*, to provide such capabilities, and present detailed analysis of these schemes. Simulation based results with these schemes demonstrate that we not only achieve several factors improvement in system revenue, but also good service differentiation as a much desired side-effect.

Keywords: QoS, Parallel Job Scheduling, Service Differentiation, Opportunity Cost

1 Introduction

Batch job schedulers are commonly used to schedule parallel jobs at shared-resource supercomputer centers such as the Ohio Supercomputer Center (OSC) [4], San Diego Supercomputer Center (SDSC) [5], Cornell Theory Center (CTC) [1], etc. The standard working model for these supercomputer centers is to allocate resources (e.g., processors, memory) to a job on submission, if available. If the requested resources are not currently available, the job is queued and scheduled to be started at a later time (when the resources become available). The turnaround time or response time of a job is the sum of

*This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

the time for which it has to wait in the job queue (for resources to be available) and the actual runtime after the job starts running. Users are typically charged as a function of the total resources used (resources $\times$ runtime).

Together with the standard working model described above, there has also been a lot of recent interest in Quality of Service (QoS) capabilities in job scheduling in terms of guarantees in the job's turn-around time. Such QoS capabilities are useful in several instances. For example, a scientist can submit a job before leaving work in the evening and request a deadline in the job's turn-around time for 8:00am the next morning, i.e., she needs the job to complete and the results to be ready by the time she is back the next morning. Currently, the only mechanism most supercomputer centers provide to achieve this is based on *advance reservations*.

With advanced reservations, when the user submits a job, she can request the needed resources during a specific time window (within her deadline). If the requested resources are available in this time window, the job is accepted and is statically assigned to be executed in that time window. If the resources are not available in this time window, the job is rejected and the user (or some software agent on behalf of the user) can try a different time window within the deadline period. While *advance reservation* capabilities are widely available in supercomputing centers, utilizing them to achieve QoS results in significant under-utilization of resources, making them not the best choice as a QoS aware scheme. Hence, in our previous work [9], we proposed a new scheme, *QoPS*, to provide QoS guarantees without statically assigning a time window to execute the job, i.e., QoPS allows jobs to be moved in the schedule while being bound by the requested deadline constraint. This results in significant improvements in resource efficiency as well as the number of accepted jobs.

The basic QoPS implementation attempts to accept as many jobs as possible and does not analyze the costs of the jobs that are being submitted. This issue is illustrated in figure 1. Let us consider a situation where there are a set of jobs that are already running in the system and there are four idle processors available. Now, let us say a 4-processor job J_4 arrives, requests a deadline in 4 hours and offers to pay \$100. In this situation, QoPS checks that it can accommodate this job into the system and accepts it. Immediately after this job is accepted, another 4-processor job J_5 arrives, requests the same deadline (4 hours) and offers to pay \$200 (a higher price than J_4). Since J_4 has already been accepted, the system cannot accept J_5 and hence has to forgo the more profitable job. This example demonstrates that it may not always be beneficial to admit all revenue generating jobs, because of consequent potential future loss of revenue, i.e., *opportunity cost*. In other words, while each job pays a explicit price to the system for running it, the system may also be viewed as paying an implicit opportunity cost by accepting the job. Accordingly, accepting the job is profitable for the system when the job's price is higher than its opportunity cost.

Formally, the opportunity cost of a job is defined as the difference between the highest revenue possible for the entire workload, with and without accepting the job. If the opportunity cost of a job is known up front, the system can easily derive the potential benefits in accepting the job. However, knowing the opportunity cost of a job up front is impossible. Thus, this paper addresses the analysis of impact such opportunity cost can have on the overall revenue of the supercomputer center

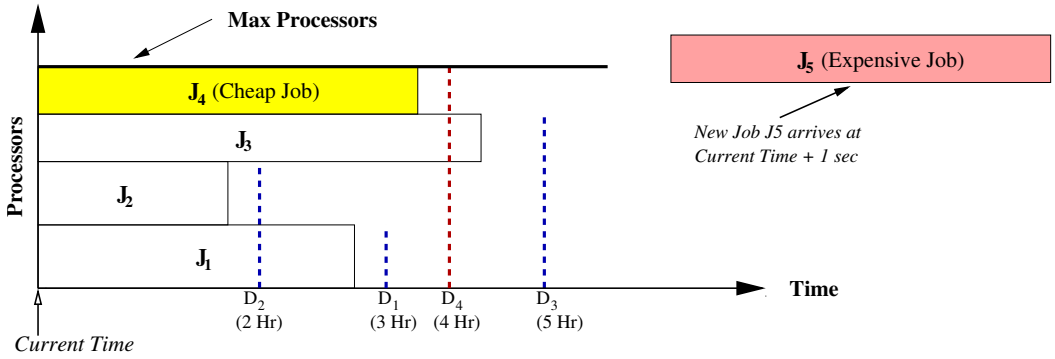


Figure 1: Expensive job J_5 can not be scheduled because identical but cheaper job J_4 is already accepted.

and attempting to minimize it through predictive techniques. Specifically, we first present an extension of QoPS, named Value-aware QoPS (VQoPS), that is aware of the different prices of different jobs and analyzes it with various statically assumed opportunity cost values for the jobs in the system. As we will demonstrate in the later sections, no single statically assumed opportunity cost value does well for all kinds of job mixes.

In the second part of the paper, we introduce Dynamic Value-aware QoPS (DVQoPS), which is a self learning variant of VQoPS to analyze past jobs and predict the opportunity costs for future arriving jobs. However, a simple history-based scheme has several disadvantages. For example, if the opportunity cost is decided based on a very long history, the mechanism will lose its sensitivity with respect to adapting to small changes in the pattern of the jobs being submitted. At the same time, if the opportunity cost is decided based on a very short history, the sample set of previous jobs might be too small and the results obtained might be noisy and unstable. To add to the complexity, the optimal history length to be considered might be different for different days (e.g., there are more jobs when there is a paper deadline, and hence short histories might be sufficient) or for different parts of the day (e.g., there are more jobs in the day time as compared to nights or weekends). Thus, the length of the history also needs to be dynamically adapted to balance sensitivity (not too long a history) and stability (not too short a history).

It is to be noted that analyzing the opportunity costs of different jobs and trying to maximize revenue also has a much desired side-effect of service differentiation, i.e., if an expensive job arrives slightly after a cheaper job arrives, a good scheme can provide appropriate service differentiation between these two jobs and give a higher chance of acceptance to the more expensive job though both jobs do not arrive at the same time.

We present the detailed analysis of both VQoPS and DVQoPS with simulation based on different real workloads of jobs from Feitelson's workload archive (San Diego Supercomputing Center (SDSC) trace from 2000 and Cornell Theory Center (CTC) trace from 1997. Our results provide various insights into the impact of opportunity costs in QoS-aware job schedulers and demonstrate that DVQoPS can provide several factors higher system revenue capabilities as compared to QoPS. Further, we also demonstrate that DVQoPS can achieve good service differentiation between high-paying urgent and low-paying

non-urgent jobs.

The remaining part of the paper is organized as follows. In section 2, we discuss other work that is related to our research. In section 3, we describe the experimental setup and simulation approach that will be utilized in subsequent sections. In section 4, we present a value-aware modification of the QoPS scheduling algorithm, targeted at a job submission environment where users provide time-varying pricing offers along with their jobs. We also develop the dynamic, adaptive, value-aware scheduling scheme that uses recent history in determining key parameters. Simulation results based on the SDSC workload are provided in section 5. Section 6 concludes the paper.

2 Related Work

While approaches to provide deadline guarantees have been widely studied in the real-time and networking communities, there has been very little work in job scheduling that addresses this issue. It has been shown that for dynamic systems with more than one processor, a polynomial-time optimal scheduling algorithm for real-time deadlines does not exist [16, 13, 12, 14]. In our previous work, QoPS (**Qo**S for **P**arallel Job **S**cheduling) [9], we proposed a heuristic based approach to provide hard deadline guarantees to end users in terms of maximum turnaround time. In this approach, together with the resources required and an estimate of the runtime of the job, users also provide a request for a deadline by which they need the job to be completed. The QoPS scheduler attempts various schedules to find a valid schedule while preserving the deadline constraints of all the previously accepted jobs. If the new job can be accommodated while maintaining all the previous constraints, it is accepted. Buyya et. al. [17, 19, 20] have addressed deadline based scheduling for a time shared system where multiple jobs can be scheduled and time sliced on a single node. The work presented in this paper, on the other hand, addresses space shared systems that are applicable to many supercomputing centers.

There has also been some recent work in the direction of providing differentiated service to various classes of jobs using statically or dynamically calculated priorities [3, 18]. These approaches provide *best effort* relative prioritization for individual jobs or different classes of jobs. Such priority may either be assigned statically to the jobs where, for example, jobs from a group of users might be given a higher priority compared to others, or may dynamically vary during the queue time of the job where, e.g., if a job has been waiting in the queue for a long time, its priority is increased. Others have examined a market based approach. The fundamental idea a market-based approach relies on, is that different users might having different goals and preferences. These goals and preferences are used to express their desire for service in a common way (e.g., currency). Two recent publications [6, 8] have addressed this concept in the field of job scheduling. Both papers rely on a per-job specific utility or value function that provides an explicit mapping of service quality to value. Generally, the value function is a piece-wise linear function which decays as a function of the job completion time. This type of function has a magnitude that shows the value of the job and a rate of decay that reflects the urgency or sensitivity to delay.

3 Evaluation Approach

The strategies evaluated in this paper are simulated using real workload traces, such as those available at the Parallel Workload Archive [7]. These traces include information such as the jobs runtime, the number of nodes each job used, the submission time, and a user estimated runtime limit (wall-clock limit). In this work, we used two such real workload trace of 10,000 jobs subset of the SDSC SP-2 and CTC SP-2 workload trace.

Deadline Information: None of the workload traces available contain any information about requested job deadlines, as hard deadline guarantees are not supported by any supercomputer center. Therefore, we synthetically generate these deadlines. For this work we assume that users will assign deadlines based on the runtime of their job, i.e., longer jobs have longer deadlines and shorter jobs have shorter deadlines. Each job's deadline is assigned to be 5 times the users estimated runtime ($deadline_j = 5 * user\_runtime\_estimate_j$).

Runtime Estimates: Runtime estimates are critical when evaluating parallel job schedulers. Therefore, two sets of simulations are performed. The first set of simulations takes an idealistic view of the traces and assumes that users are able to perfectly estimate their job's runtime; this allows us to concentrate on the capabilities of our algorithms without being affected by other noise in the traces. The second set of simulations uses the actual runtime estimates given in the workload trace; this allows us to evaluate our algorithms in more realistic environments.

Job Submission Load: In most supercomputer centers, the number of jobs submitted within a given time varies from hour-to-hour (high in the day time and low in the nights), between days (high during the week days and low in the weekends) or even between weeks (high during submission deadlines when researchers are looking for measurement results). In order to capture such different scenarios, we carry out simulations with two different kinds of offered loads for each traces. The first is the original load as measured in the SDSC trace subset (total processor seconds submitted correspond to about 89% of the overall system capacity). The second is a high load emulation that is achieved by randomly duplicating roughly 40% of the jobs (total processor seconds submitted correspond to about 125% of the overall system capacity). The same technique is used for CTC trace with original load (74%) and high load (104%). Duplicated jobs have the same submission time, runtime estimate, runtime, and deadline as the original job.

Urgency and Job Cost: Like deadline information, none of the workload traces available contains any information about urgency requirements for the job or the amount the user is willing to pay for the job. Hence, for our evaluations we randomly mark a fraction (*U*) of the jobs as *urgent*. The cost of non-urgent jobs is fixed at 0.1 units per processor-second of the job. The urgent jobs' cost is set to be higher than that of non-urgent jobs by a factor (*C*). In our experiments, we used values of 20%, 50% and 80% for *U* and values of 10, 5 and 2 for *C*. Further, as shown in figure 2, we assume that each job specifies a maximum cost the user is willing to pay for the job, and a linearly decaying cost function. The value of the job becomes *zero* at the requested deadline of the job. This model is quite similar to that used in previous papers [6, 8], except for deadline

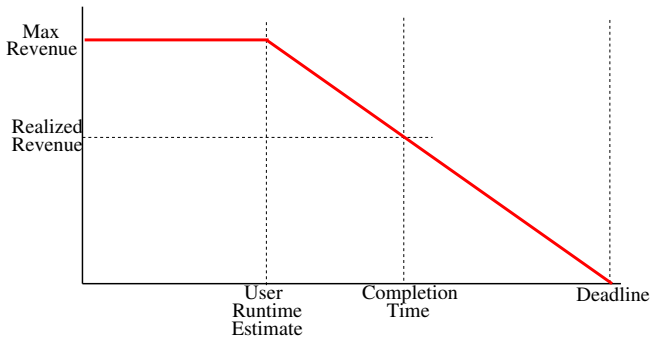


Figure 2: The realized revenue

guarantees, which were not previously considered.

4 Opportunity Cost in Job Scheduling

As described in section 1, each accepted job has the potential to force the system to reject future arriving more expensive jobs, i.e., with each accepted job the system pays an opportunity cost. However, the opportunity cost of a job depends on a number of parameters, both local to each job as well as global to the entire workload of jobs. In section 4.1, we describe the various job and workload characteristics that impact opportunity cost. In section 4.2, we design a scheme that utilizes some of the local job information to minimize the impact of opportunity cost. In section 4.3, we extend this design to monitor the global workload and predict the characteristics of future arriving jobs, thus allowing for improved performance.

4.1 Impact of Job and Workload Characteristics on Opportunity Cost

Opportunity cost for a given job identifies the amount of potential future revenue that a job loses by accepting this job. This potential future revenue depends on a number of aspects, which can be classified into two broad categories: local characteristics of the job and global characteristics of the workload.

Local Characteristics of the Job: Two primary characteristics of a job play a role in impacting its opportunity cost, viz., job shape and QoS requirements of the job.

1. *Job shape (processors and wall-clock time requested):* The shape of a job can affect how many later jobs must be dropped. For example, if a large job (which requests many processors and/or a long wall-clock time) is accepted, it uses up more of the available resources in a schedule, potentially requiring more later arriving jobs to be rejected. If on the other hand, a smaller job is accepted, it might be possible to accept more later arriving jobs. Thus, the opportunity cost of large jobs is likely higher then a small job.

2. *QoS requirements of the job:* The QoS requirement of the job can determine how stringent the schedule is, i.e., if a job has a tight deadline, it cannot be moved at all. On the other hand, if a job has a looser deadline, it might be possible to move in the schedule and thus allow other jobs to be accepted more easily. Thus, the opportunity cost of

stringent jobs is likely higher.

Global Characteristics of the Workload: Three primary characteristics of the workload play a role in impacting a job's opportunity cost, viz., offered load, job mix and job pricing.

1. *Offered load:* When the number of jobs in the system is very less (i.e., offered load is low), the acceptance of a non-urgent job is less likely to prevent the admittance of a future urgent job. Thus, the opportunity cost would typically be lower than when the number of jobs in the system is high (i.e., offered load is high).
2. *Job mix (% of Urgent Jobs)*: If all jobs had the same urgency and pricing, the opportunity cost of a job is essentially zero, since it is not possible for a later job to have better pricing. But when there are some urgent (and high-paying) and some non-urgent (and low-paying) jobs in the system, opportunity cost is no longer zero. Further, if the percentage of urgent high-paying jobs in the job trace is very high, the opportunity cost of admitting a non-urgent job is high, since there is a high probability that its admittance could prevent admission of a future high-paying job.
3. *Job pricing*: The relative premium paid by urgent jobs is also an important factor. The higher the premium paid by an urgent job relative to a non-urgent job, the greater the cost of losing an urgent job, and thus greater the opportunity cost of non-urgent jobs.

4.2 Value-aware QoPS (VQoPS)

The QoPS algorithm does not perform any differentiation between jobs. It assumes that all jobs have the same price value assigned to them. However, in an environment that allows users to offer price based on responsiveness, some jobs in the system will offer more revenue than others. Similarly, some jobs will have tighter deadlines than others. For an algorithm that is expected to improve the overall revenue of the system, the following two properties are desirable:

1. During backfilling, the new algorithm should reorder the job queue so as to give a higher priority for more urgent jobs and attempt to reduce their turnaround time, thus increasing revenue.
2. The new algorithm should attempt to maximize the overall revenue of the system during job acceptance by considering both the explicit revenue benefit to the system and the implicit loss of opportunity for the system.

4.2.1 Design of VQoPS

This section presents a new design known as Value-aware QoPS (VQoPS) that provides the above two features. Specifically, it utilizes some of the local information about a job, i.e., the shape of the job, to identify whether the job is likely to have a high opportunity cost or not. Accordingly, an opportunity cost that is proportional to the size of the job (number of processor-seconds of the job) is statically assumed, i.e., the opportunity cost is assumed to be OC-Factor $\times$ job size, where OC-Factor is a static system parameter.

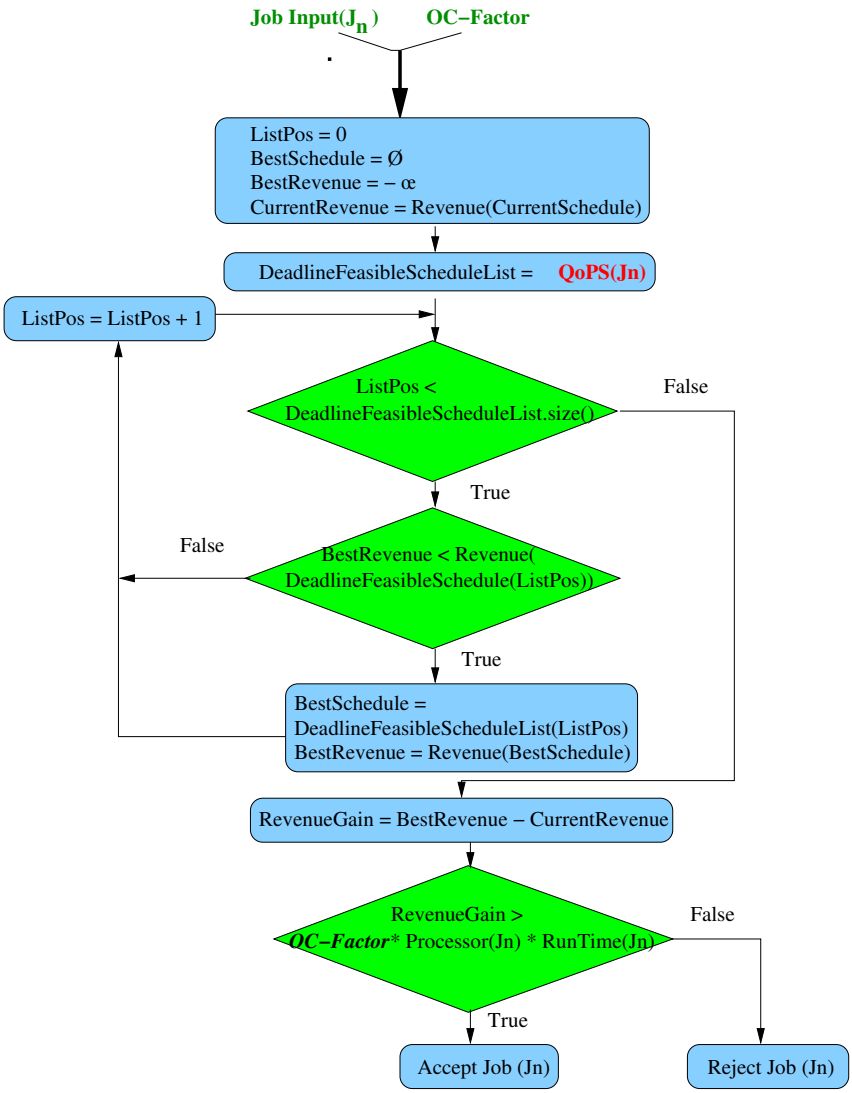


Figure 3: VQoPS Algorithm Flow Chart

Figure 3 shows the high-level VQoPS flowchart based on the properties described above. VQoPS utilizes QoPS as a pluggable module to check the acceptance of a new job (passed as an argument) while maintaining the hard deadline restriction for all accepted jobs. Since the original QoPS design just provided a recommendation about whether a job should be accepted or rejected, we adapted it to instead provide the entire list of acceptable schedules. Particularly, when a new job arrives in an existing schedule of *N* accepted jobs, QoPS tries to push the new job in $log_2(N)$ different positions $\{0, N/2, 3N/4, ...\}$ of the queue resulting in as many as $log_2(N)$ different schedules. Of these, the list of feasible schedules, where no job misses its requested deadline, is returned by the QoPS module.

When a new job arrives, this job together with the OC-Factor is passed as input to VQoPS. Admission control for this job is done in two stages. In the first stage, VQoPS uses, as mentioned above, the QoPS module to check for feasible schedules such that the newly arriving jobs meets its deadline without violating the deadline constraint of any of the already accepted

Table 1: Revenue improvement for varying OC-Factors

Relative Urgent Cost	% Urgent Jobs	Offered Load	OC-Factor 0.00	0.05	0.10	0.20	0.40
10X	80%	Orig	21%	26%	37%	37%	**39%**
5X	80%	Orig	20%	25%	34%	**35%**	30%
2X	80%	Orig	19%	26%	**27%**	-47%	-100%
10X	80%	Orig	21%	26%	37%	37%	**39%**
10X	50%	Orig	23%	34%	**46%**	45%	45%
10X	20%	Orig	26%	**38%**	22%	22%	22%
10X	80%	Orig	21%	26%	37%	37%	**39%**
10X	80%	High	63%	90%	135%	144%	**160%**

jobs. In the second stage, VQoPS weighs the statically assumed opportunity cost of the new job with its price and decides whether the job should be accepted. In particular, if the revenue gain obtainable can offset the opportunity cost, VQoPS accepts the job. It is worthy to note that the OC-Factor input in VQoPS is constant (static) throughout the job processing and expected to be defined by an administrator as a system parameter. The time complexity of our approach is $\theta(K.N^2.log(N))$.

4.2.2 Impact of Workload Characteristics on VQoPS

In this section, we analyze VQoPS by varying the different characteristics of the workload, such as the relative urgency cost, percentage of urgent jobs and the offered load on the system.

Table 1 shows the improvement in revenue generated (compared to QoPS) for the different workload characteristics and different OC-Factors (0.0, 0.05, 0.1, 0.2, and 0.4). Several insights can be drawn from these results. Primarily, though the VQoPS algorithm outperforms QoPS by up to 160% in some cases, we notice that the improvement achieved is highly dependent on the trace characteristics. That is, no single OC-Factor can consistently provide a good performance for all kinds of workloads. In fact, the same OC-Factor that provided a 160% improvement in one trace, can provide worse performance than even QoPS by almost 100% in another trace.

The impact of the workload characteristics on VQoPS can be noticed in several aspects. First, as the relative cost of the urgent jobs decreases, we see the smaller OC-Factor values perform better. This is expected, since the opportunity cost of turning away a future urgent job decreases as the cost of the urgent job decreases. In other words, if QoPS makes a bad decision about accepting a job and ends up accepting a normal job instead of an urgent job, the opportunity cost it would pay would decrease as the cost of the urgent job decreases and hence the amount it would lose would reduce. Accordingly, the OC-Factor providing the best performance decreases as well.

Second, as the number of urgent jobs decreases, smaller OC-Factor values perform better. This is again an expected trend since as the number of urgent jobs decreases, though the worst case opportunity cost for QoPS would not reduce, the average case opportunity cost would reduce, and accordingly the OC-Factor.

Third, with load increase we see that the revenue improvement increases for all OC-Factor values. As the fraction of

dropped jobs increases with increasing load (since the system cannot accommodate all the jobs while meeting their requested deadlines), the algorithm gets very selective and picks only the high-paying jobs. QoPS, on the other hand, picks all the jobs that it can meet the deadlines for without considering their prices. This, accordingly, reflects as higher revenue improvement for all OC-Factor values at high loads.

4.3 DVQoPS: A Self-learning Variant of Value-aware QoPS

As described in section 4.2, while Value-aware QoPS is able to achieve revenue improvements, the behavior of the algorithm heavily depends on the characteristics of the trace. Thus, it is desirable to develop a more sophisticated algorithm that has all the benefits of VQoPS, but is able to adapt the OC-Factor based on the characteristics of the trace. In this section, we describe a variant of VQoPS which aims to achieve this.

4.3.1 Dynamic Value-aware QoPS (DVQoPS)

The VQoPS algorithm presented in the previous section requires that a good OC-Factor be determined - say by a system administrator. In order to properly tune the OC-Factor, various dynamically changing variables must be taken into account. The offered load must be accounted for. In lightly loaded situations more normal jobs should be accepted, and a lower OC-Factor should be used. As the load increases, the OC-Factor should be increased. If the expected price difference between urgent and non-urgent jobs is small, the scheduler should be more aggressive and accept more of the less expensive jobs. As the expected revenue difference increases, the OC-Factor should be increased to not accept jobs that only increase revenue modestly. Finally, as the percent of urgent or expensive jobs increases, the OC-Factor should also increase. When there are only a few urgent jobs, it is not desirable to reject a large number of normal jobs waiting for an urgent job to arrive. However, as it becomes more likely an urgent job will arrive soon, the OC-Factor should be increased.

It is very difficult to manually take these considerations and their interactions into account. It becomes even more difficult as these criterion may change over time. Thus, it is highly desirable for the scheduler to automatically generate and adjust the VQoPS OC-Factor. The approach we pursue is that of performing a number of *what-if* simulations over a limited backward window in time, called the *rollback window*. The essential idea is to periodically adjust the OC-Factor by looking backwards and performing simulations to estimate the revenue that would have been realized for various choices of OC-Factors. A set of such simulations is feasible since the history of arrived jobs (both actually accepted and those rejected in the real schedule) is available. For each choice of OC-Factor, simulation over the rollback window is used to estimate the revenue for each job accepted by the simulated schedule, and thereby the total estimated revenue over the rollback period is estimated. The OC-Factor giving the best estimated revenue over the window is chosen as the actual OC-Factor to be used for the immediate future (until the next OC-Factor change event).

We next discuss the issue of the choice of rollback window duration. The basic premise of the adaptive OC-Factor selection procedure is that the best OC-Factor depends on various characteristics of the trace, which vary over time. For example, the

load in supercomputer centers tends to be bursty, with some periodic patterns. At federal supercomputer centers, job arrival rate during working hours tends to be higher than during the night, and arrival rate during weekdays tends to be higher than during weekends. If the DVQoPS scheme is to be effective in adapting to such load variations, clearly the rollback window must not be too large, e.g., one month, because the time variance in load will get averaged out over the long rollback window. At the other extreme, there is a different problem if the rollback window is made extremely small in order to be very responsive to the time variation of trace characteristics. When the number of jobs submitted over the window is very small, the results of the what-if simulations may be extremely sensitive and not robust. In the next subsection we discuss this issue further.

4.3.2 Balancing Sensitivity and Stability in DVQoPS

The choice of rollback window involves a judicious balance between two competing factors:

1. Sensitivity to Trace Variation: The rollback window should be short enough to capture the effect of changes in input trace characteristics, such as load and job mix characteristics.
2. Stability: The rollback window should not be so short that specific jobs affect the best what-if OC-Factor, rather than the aggregate characteristics of the jobs in the window.

In order to assess these factors, we carried out studies using different traces. For assessing the effect of the rollback window to trace variation, the average offered load was computed over segments of duration equal to the rollback window, and the variance of these averages was computed. At one extreme, if the rollback window size is the entire trace duration, the variance is zero since a single average load is computed. At the other extreme, when the rollback window is minimized, the variance of the average loads is maximized. As the rollback window size increases, the load variance is expected to decrease.

To understand the impact of rollback window on stability of OC-Factor choice, consider the following example. Suppose that a set of N consecutively arriving jobs were considered for the what-if simulations in a rollback window, and the best OC-Factor choice determined. Let us then slightly change the front end of the window to exclude the latest of these N jobs, but move back the rear end of the rollback window to include the last arriving job outside the currently chosen window. If the best choice of OC-Factor for the two rollback windows is very different, it implies that the choice procedure is very unstable.

Table 2 shows results that study the impact of rollback window choice on the variance of OC-Factor choice for adjacent window groups, variance of the average offered load and overall revenue, for the scenario of exact user estimates, relative urgent job cost = 10X, and 50% urgent jobs at a high offered load. By varying the length of the rollback window from 1 to 128 hours, the revenue varies from 414M units to 716M units. This shows that choosing the correct rollback window size will have a large impact on the overall revenue. However, setting the *rollback window* size is not trivial. If the *rollback*

Table 2: Effects of rollback windows size for exact (perfect) user estimates, relative urgent job cost = 10X, and 50% urgent jobs at a high offered load

Rollback Window Size	Avg. OC-Factor Variance ($*10^{-5}$)	Load Variance	Revenue
1	3.15	10.60	473631718
4	6.18	2.89	508341077
16	7.84	0.62	555062813
32	2.99	0.34	692266945
48	1.36	0.24	715606095
64	1.03	0.17	715733110
128	1.13	0.04	701476009

window is too small, the OC-Factor will change very erratically. The second column of the table shows the average of the variance of each set of 5 consecutive OC-Factor choices. A higher average variance shows that the OC-Factor changes more erratically and thus will be unable to reach a good value. Because with a small *rollback window*, a small change in the exact jobs considered in the window will result in a very different OC-Factor, implying that OC-Factor for the new job was incorrect. However, if the value is too large, critical changes in the workload (e.g., load, percentage of urgent jobs, relative job values) may be missed. The third column shows the variance of the average offered load over the window size. As the variance in average offered load decreases, important variations in the load are being missed and the historical simulation will not be able to "see" the variations. The *rollback window* size needs to be large enough to have a low average OC-Factor variance (so the historical OC-Factor has meaning), but small enough to capture significant workload differences. Therefore, each scenario may require a different *rollback window* size that may vary over time.

Thus, a *max rollback window* (64 hours for this paper) is used to dynamically vary the *rollback window* size. Each *max rollback window* hours the scheduler runs historical simulations (using each of the following *rollback window* sizes: 1 hour, 2 hours, 4 hours, 8 hours, 16 hours, 32 hours, 64 hours) to determine what *rollback window* would have yielded the best revenue over the last *max rollback window* hours. For the next *max rollback window* hours, the scheduler uses the new *rollback window*.

Figure 4 shows the DVQoPS flowchart encompassing the aspects described above. Figure 4(a) illustrates the main flow of the DVQoPS algorithm, which passes the dynamically calculated OC-Factor to the VQoPS algorithm (used as a module). DVQoPS also asynchronously evaluates the rollback interval and OC-Factor after every fixed interval. The details of calculating these two system variables are presented in figures 4(b) and 4(c) respectively. Figure 4(b) demonstrates the basic steps used to determine the best rollback interval. This is used in figure 4(c) when evaluating the best OC-Factor.

For each candidate rollback interval, DVQoPS runs the simulation starting *candidate rollback interval* hours in the past. The *rollback interval* is set to the *candidate rollback interval* that would have produced the best revenue. This best rollback interval is used for the next *max rollback window* hours. The OC-Factor is set by running the what-if simulation for different values of *candidate OC-Factor*s. For each candidate OC-Factor, DVQoPS runs a what-if simulation starting

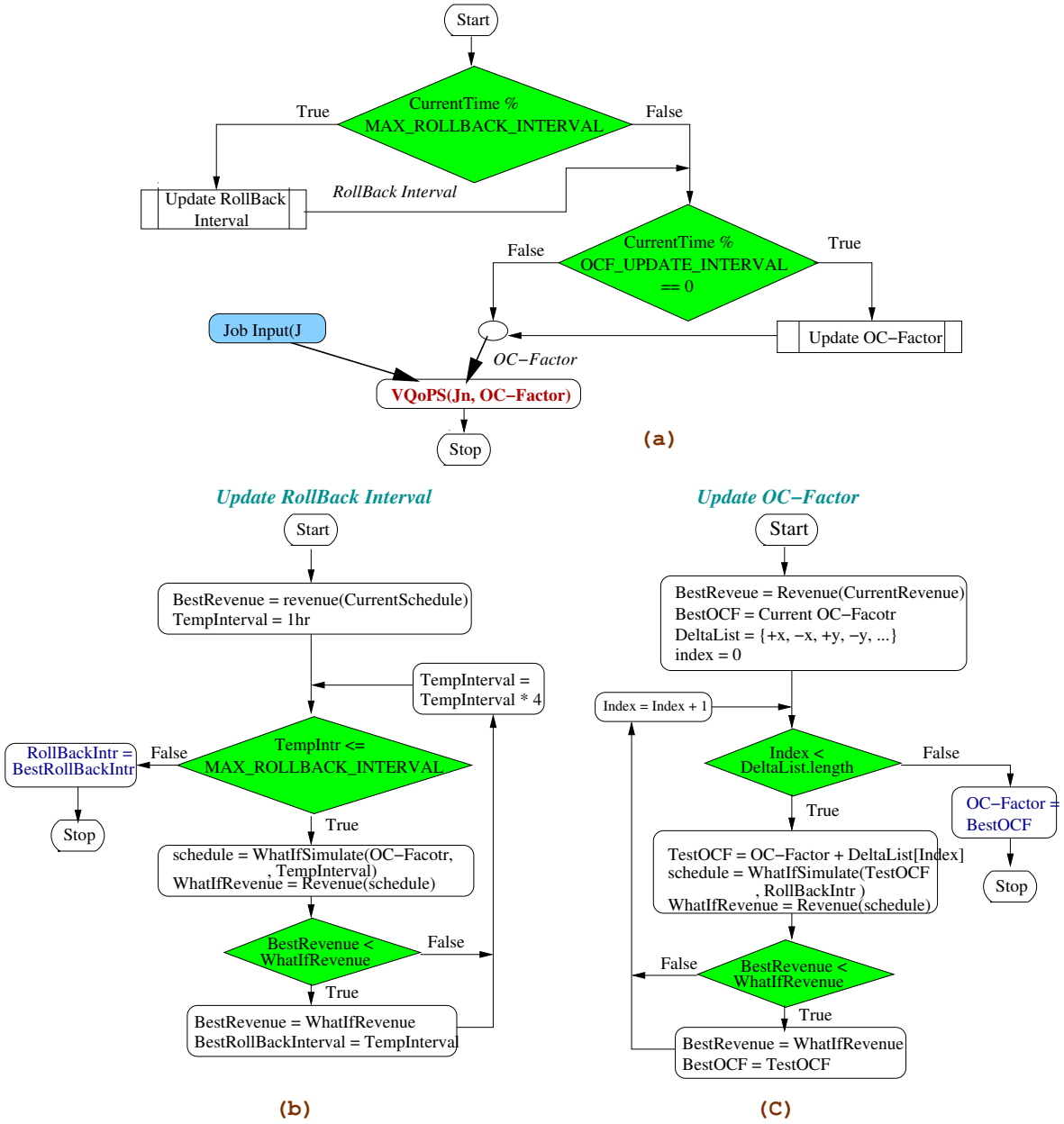


Figure 4: DVQoPS Algorithm Flow Chart

rollback window hours in the past. The current *OC-Factor* is set to the *candidate OC-Factor* that yields the maximum revenue. DVQoPS uses the new OC-Factor for the next *rollback window* hours.

Since DVQoPS dynamically determines the best OC-Factor out of a set of T OC-Factors and the best rollback window out of a set of R rollback windows, its time complexity would increase to $\theta(T.R.K.N^2.log(N))$. While the time complexity of the scheme seems to be high, in practice the time taken for a scheduling event is not too much. For example, during our experiments, the scheduling event took an average of 0.9 seconds for each job. Given that job arrival times are typically in the order of several minutes in most supercomputer centers, this is not a concern.

5 Experimental Results

In this section, we use a simulation approach to evaluate the capabilities of the proposed schemes. Specifically, we compare the performance of VQoPS and DVQoPS against QoPS to study the benefits in overall revenue and service differentiation that each of these schemes can achieve.

As described in section 4, there are three workload characteristics that affect the opportunity cost of jobs and hence the system revenue: (i) urgent job mix, (ii) cost of urgent jobs and (iii) the load on the system. In this section, we vary the above characteristics to analyze the behavior of the different schemes. Further, we also measure the impact of user inaccuracy in runtime estimates on the performance of our schemes. As mentioned in sections 4.2.1 and 4.3.2, the time complexities of the VQoPS and DVQoPS schemes are $\theta(K.N^2.log(N))$ and $\theta(T.R.K.N^2.log(N))$ respectively. In practice the time taken for a scheduling event using DVQoPS is less than a second (as measured in our experiments) on a 2.4GHz system. Further, most of the processing of these scheduling systems can easily be processed with near linear speedup on a parallel machine, with up to (T $\times$ R) processors using a master-slave setup.

5.1 Overview of the Simulator

Job scheduling research historically involves running an event based simulator on a synthetic or a historic input trace. These input traces normally must contain the runtime, number of nodes, and queue or arrival time for a set of jobs. The traces can also contain optional information that the scheduler may or may not need. One such piece of information is the user expected runtime (or wall-clock limit). The user expected wall-clock limit, is often important information, there have been multiple studies showing the importance of user estimates [15, 11, 10].

Simulations are used to design and test emerging scheduling algorithms before development in a production system. Simulations provide an environment where varying scheduling algorithms can be compared in reproducible scenarios, providing comparable results. In addition, months of data can be simulated in seconds or minutes, in comparison to the months or years it would require to run multiple algorithms in a live environment.

DVQoPS and VQoPS are evaluated using an event-based simulator. The simulator takes data in the standard workload format version 2.0 [7]. The simulator then performs the necessary simulation and creates an output trace containing data necessary to gather metrics and perform post processing. This data includes an entry for every job. This allows us to easily obtain statistics via post processing.

5.2 Impact of Urgent Job Mix on Performance

Figure 5(a) illustrates the percentage improvement in revenue (as compared to QoPS) for VQoPS (with different static OC-Factors) and DVQoPS for SDSC trace. The figure shows that VQoPS achieves about 20%-45% improvement in performance for all OC-Factor values. Also, depending on the trace characteristics (i.e., what percentage of the jobs are urgent jobs),

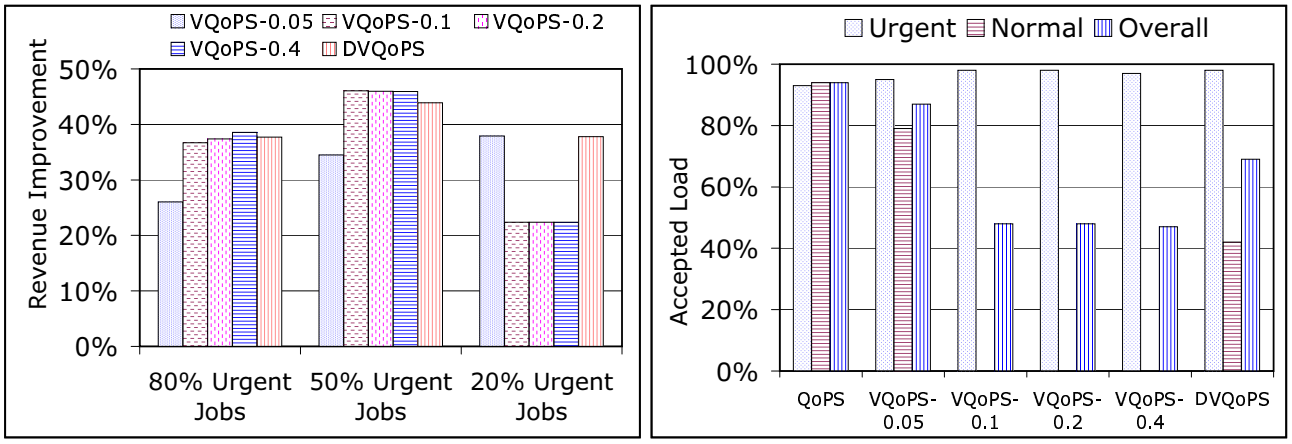


Figure 5: Revenue improvement and Accepted load realtive to QoPS assuming exact estimates at the original offered load (89%) for SDSC trace. Cost of urgent jobs is 10X compared to normal jobs. In (b), 50% urgent jobs are used.

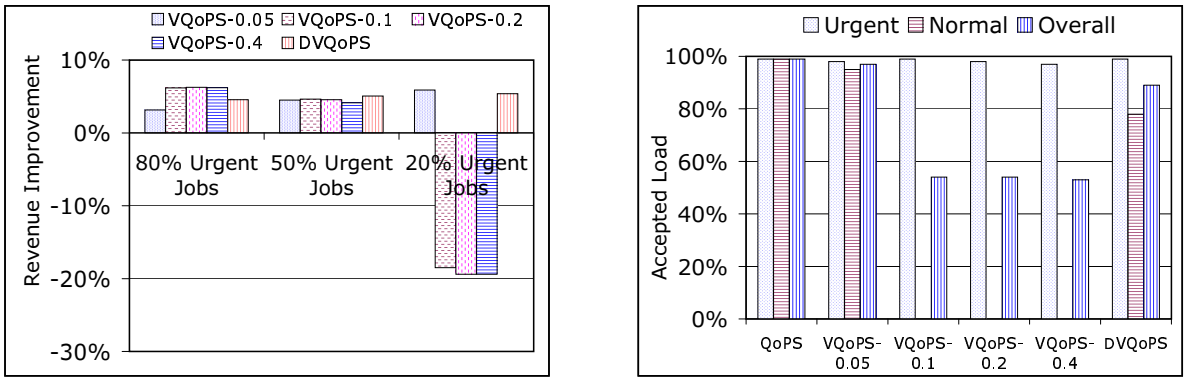
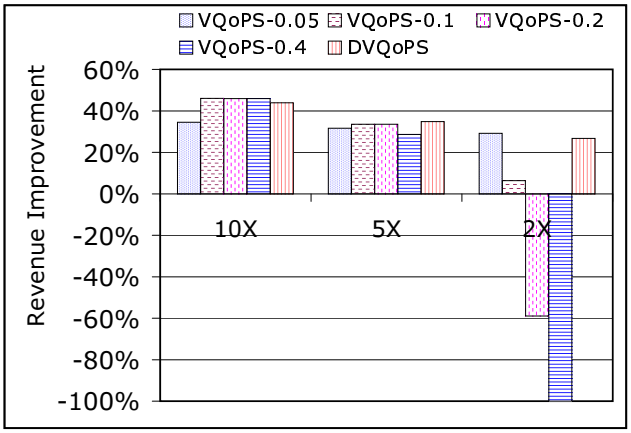


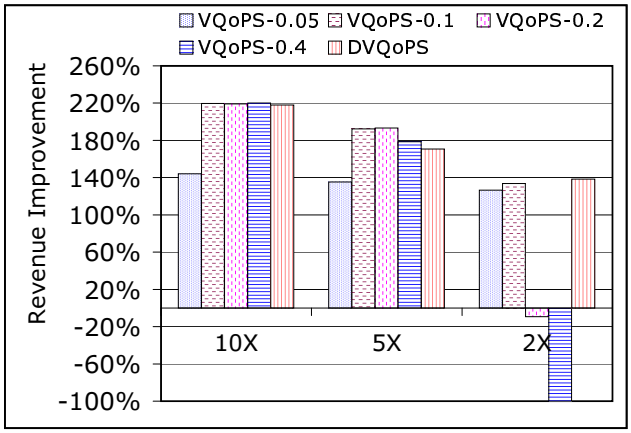
Figure 6: Revenue improvement and Accepted load realtive to QoPS assuming exact estimates at the original offered load (74%) for CTC trace. Cost of urgent jobs is 10X compared to normal jobs. In (b), 50% urgent jobs are used.

different OC-Factor values perform well – there is no consistently superior OC-Factor value. DVQoPS, on the other hand, consistently achieves within 5% of the best VQoPS implementation for all traces.

Together with revenue maximization, it is important to provide service differentiation as well. Figure 5(b) shows the service differentiation capability of the different schemes. As shown in the figure, QoPS accepts the highest overall percentage of the workload. However, it does not differentiate between urgent and normal jobs. This inability to differentiate is what allows for the increase in revenue for the VQoPS and DVQoPS scheduling policies. As the OC-Factor is increased, the VQoPS scheduler rejects more jobs that could have been accepted by the standard QoPS algorithm, reducing the overall acceptance. However, as the OC-Factor is increased (up to a certain value) the percentage of accepted urgent jobs is increased. Very few normal jobs are accepted at OC-Factors greater than 0.05, even when the revenue is high. The increase in revenue is caused by the ability to achieve a higher revenue from urgent jobs. The DVQoPS algorithm is able to achieve competitive revenues, but accepts many more normal jobs; it even accepts more urgent jobs in scenarios where the static OC-Factor is greater then 0.05, where the revenue is often the best. In summary, the DVQoPS algorithm is able to tune the percentage of urgent and non-urgent jobs it is able to accept and to dynamically adapt itself to the trace characteristics.

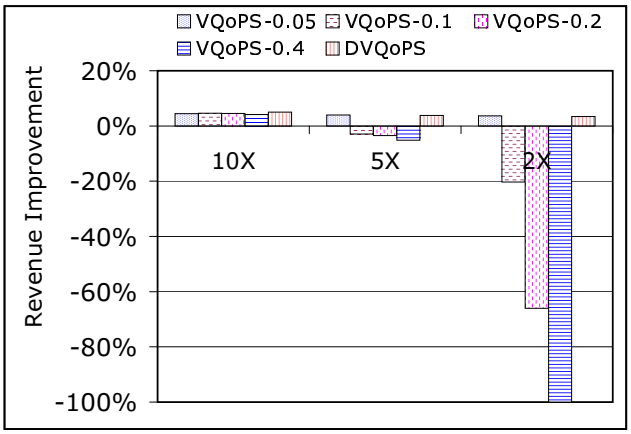


(a) Original offered load (89%)

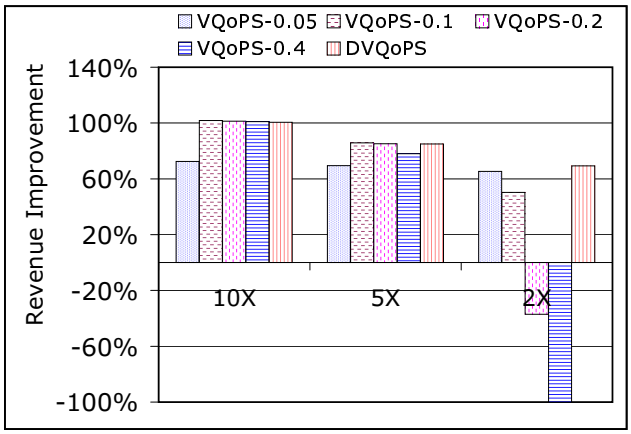


(b) High offered load (125%)

Figure 7: Revenue improvement realtive to the QoPS assuming exact estimates for original and high offered load for SDSC trace



(a) Original offered load (74%)



(b) High offered load (103%)

Figure 8: Revenue improvement realtive to the QoPS assuming exact estimates for original and high offered load of CTC trace

To demonstrate the robustness of our schemes across different workloads, we illustrate the revenue improvement using the CTC workload in Figure 6 . It is evident from the figure that, the trend of the results is similar in the CTC workload as well. However, the improvements are lesser compared to the corresponding SDSC workload. This is attributed to the low original offered load (74%) for CTC workload as compared to the SDSC workload (89%), i.e., if the number of jobs are less, opportunity cost would be less too; thus the difference between the schemes will not be as prominent.

5.3 Impact of Cost of Urgent Jobs and System Load

Figure 7(a) illustrates the performance of the different schemes as we vary the cost of the urgent jobs SDSC workload. The first thing we observe is that when the cost of the urgent jobs is very low, high static OC-Factors actually perform worse as

compared to QoPS. This is expected, since high static OC-Factors aim to be extremely *picky* about jobs by anticipating that picking cheap jobs might result in a high opportunity cost and hence a loss of revenue. However, when the urgent jobs are not very expensive, the potential opportunity cost is low; thus not accepting the cheaper jobs would in fact hurt revenue as it does in the above figure. Again, DVQoPS shows a consistent performance with less than 5% difference as compared to the best static OC-Factor.

Figure 7(b) illustrates a similar trend, but for a high-load scenario. In this case, we can observe that the revenue improvements are much higher as compared to the original load scenario. This is because, though the job mix is the same as the original load case, the absolute number of urgent jobs is higher in the high-load case. Since all the schemes shown tend to pick the urgent jobs and drop the non-urgent jobs, this allows them to improve their revenue further. The overall trend, however, is still the same, with high static OC-Factors performing worse than QoPS when the cost of urgent jobs is not too high.

We illustrate the revenue improvement using the CTC trace in Figure 8 relative to QoPS. The revenue improvement is around 5% for original offered load (74%) and as high as 100% for higher offered load (103%). Though it is obvious from the graphs that the performance trend is same for the proposed schemes for both CTC and SDSC trace, but the performance improvement for SDSC trace is better due to higher available load in SDSC trace. It also substantiates that our schemes can earn further benefit from extra offered load.

5.4 Impact of User Runtime Estimate Inaccuracy

Figure 9(a) shows the performance of the schemes for different urgent job mixes in the case where the original inaccurate user estimates of jobs are used with SDSC workload. Compared to a scenario with exact user estimates, we notice that the overall revenues are lower in all cases (especially when the percentage of urgent jobs is low). The main reason for this is the functional difference between a service differentiation scheme and a scheme that maintains deadline guarantees. Specifically, a scheme that maintains deadline guarantees has to be conservative with respect to its estimates about a job's runtime. For example, if a one-minute job estimates its runtime as one-hour, the scheme has to assume that its going to run for the entire one-hour and try to find a feasible schedule before it can accept the job. Because of this conservative nature of the deadline guarantee scheme, the number of jobs that are accepted are much lesser than what the system could potentially accept. Further, for all the accepted jobs, if most jobs terminate early and thus pay the maximum amount they had promised (figure 2), there is no real differentiation that can be achieved for these jobs. Only for the jobs that are accepted and have to wait in the queue, can we provide efficient mechanisms to improve the overall revenue of the system. Since these kind of jobs are lesser with inaccurate estimates, we see that the overall revenue is lesser as well. In general, with inexact user estimates, it may appear that little profit can be made, but due to jobs completing early, more profit can actually be made by running jobs earlier and the effect of the opportunity cost is reduced. Therefore, it is more often better to be more lax with the OC-Factor and accept jobs that appear to only modestly increase revenue.

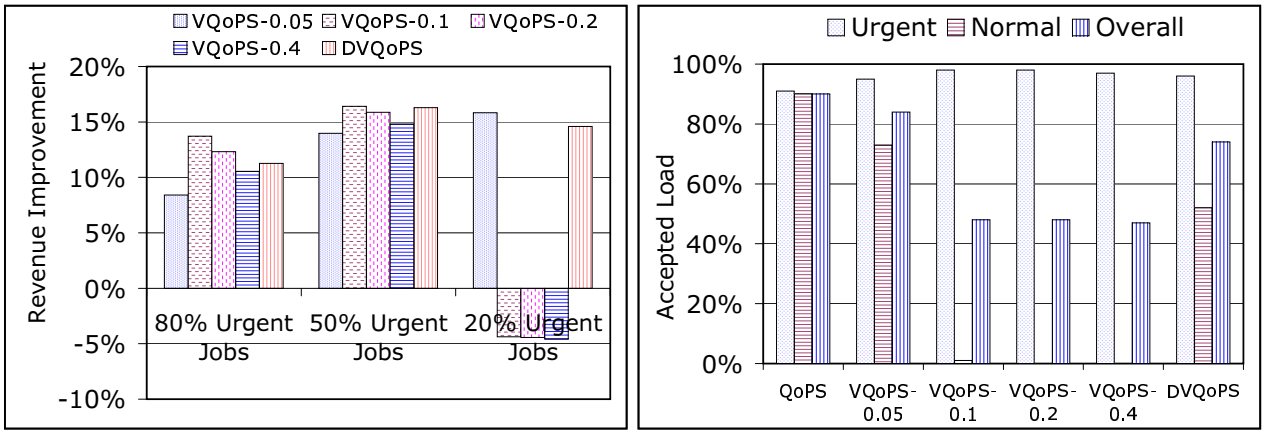


Figure 9: Revenue improvement and Accepted load realtive to QoPS assuming actual inexact estimates at the original offered load (89%) for SDSC trace. Cost of urgent jobs is 10X compared to normal jobs. In (b), 50% urgent jobs are used.

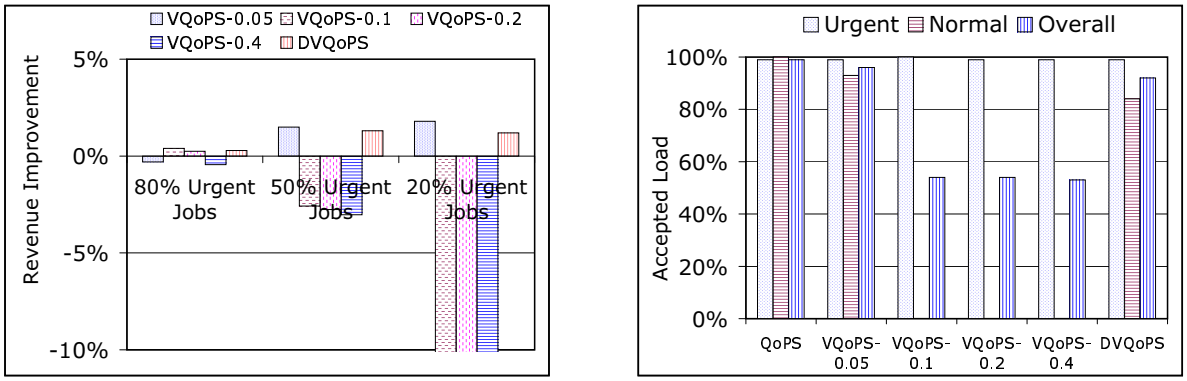
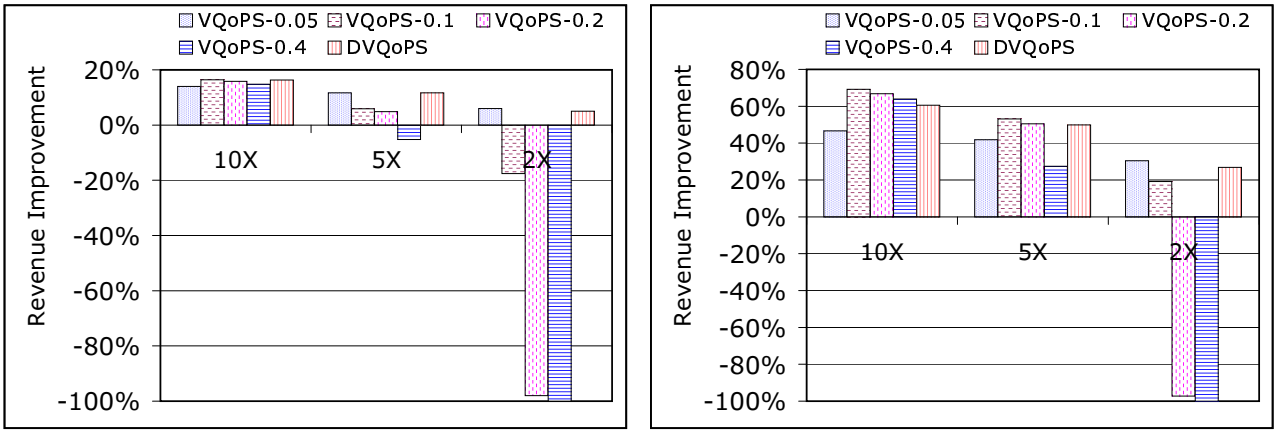


Figure 10: Revenue improvement and Accepted load realtive to QoPS assuming actual inexact estimates at the original offered load (74%) for CTC trace. Cost of urgent jobs is 10X compared to normal jobs. In (b), 50% urgent jobs are used.

Also, for low percent of urgent jobs, high OC-Factor values actually perform worse than QoPS. The reason for this is, when the static OC-Factor is high, the scheme is rejecting all the normal jobs; since the number of urgent jobs is very low, the system is being left highly under-utilized as compared to schemes with lower static OC-Factor values. This reflects in a lower revenue than even basic QoPS in some cases.

Figure 9(b) shows the service differentiation capabilities of the different schemes. The trends for the inexact case are pretty similar to that of the exact case. This is expected since the inaccuracy in estimation only affects the overall revenue of the system, but not the kind of jobs each scheme can accept. Like performance for SDSC trace in Figure 9, Figure 10 exhibits the same overall trend for CTC trace. Again, the improvement is less compare to SDSC due to less overall load in CTC trace.

Figures 11(a) and 11(b) illustrate the revenue improvement capability of the different schemes while varying the cost of urgent jobs for original offered load as well as high offered load for SDSC workload. The overall drop in revenue improvement is especially noticeable when the cost of urgent jobs is less. Since there are not many jobs to achieve a revenue improvement from in the inaccurate estimate case, if the urgent job cost is low, the overall revenue would get hurt as well.



(a) Original offered load (89%)

(b) High offered load (125%)

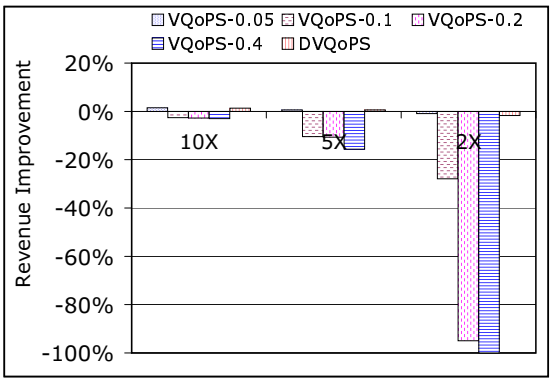
Figure 11: Revenue improvement realtive to the QoPS assuming inexact estimates for actual and high offered load for SDSC trace

For a higher load, though the improvement is a little better, compared to exact estimates, there is a drastic degradation. Figure 12 presents the performance of our proposed schemes for CTC trace for the same scenarios. The general tendency of improvement remains the same as SDSC in Figure 11 and thus demonstrates the robustness of our schemes independent of traces.

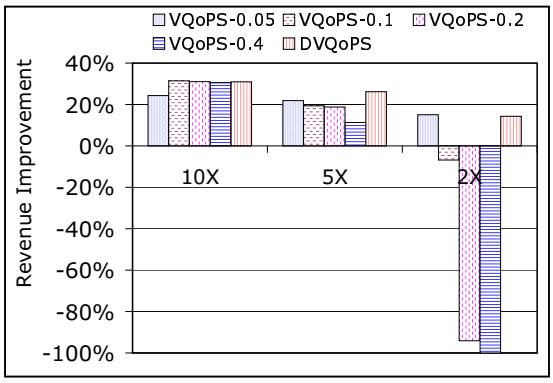
Overall, it appears that inaccuracy in user estimates has a significant impact on the revenue improvements VQoPS and DVQoPS can achieve. The service differentiation aspect is not affected, as expected. However, for a deadline guarantee scheme, we expect the supercomputer centers to provide a dual charging model, i.e., resource usage cost and deadline guarantee cost. Resource usage cost could be the same as what we currently have (resources $\times$ runtime). For the deadline guarantee cost, since requesting for a deadline guarantee means that potential later jobs could be rejected, we expect supercomputer centers to charge the user based on the estimated runtime rather than the actual runtime. This, hopefully, would require users to improve their runtime estimates and in turn improve the revenue gains VQoPS and DVQoPS can achieve.

6 Concluding Remarks and Future Work

In this paper, we proposed two extensions to our previous QoS-aware job scheduling mechanism, *QoPS*, in order to analyze and minimize the impact of *opportunity cost* in job scheduling. Specifically, for each job accepted, the supercomputer center might relinquish its capability to accept some future arriving (and potentially more expensive) jobs, i.e., it can viewed as paying an implicit opportunity cost. Accordingly, in this paper, we used predictive techniques to identify such opportunity costs and attempted to minimize their impact on the overall system revenue. We analyzed our designs and evaluated them in a simulation environment with different real workloads of jobs that had run on the San Diego Supercomputing Center (SDSC) in 2000 and the Cornell Theory Center (CTC) in 1997. Our results demonstrated that we not only achieve several



(a) Original offered load (74%)



(b) High offered load (103%)

Figure 12: Revenue improvement realtive to the QoPS assuming inexact estimates for actual and high offered load for CTC trace

factors improvement in system revenue, but also good service differentiation as a much desired side-effect.

As a part of future work, we would like to integrate these services into production job schedulers such as Moab [2]. Further, we would also like to study interactions of other schemes such as advanced reservations with our scheme.

References

[1] Cornell Theory Center (CTC). http://www.tc.cornell.edu/.

[2] Moab. http://www.supercluster.org/moab.

[3] National Energy Research Scientific Computing Center. http://www.nersc.gov/nusers/accounts/charging/sp-charging.php.

[4] Ohio Supercomputer Center (OSC). http://www.osc.edu/.

[5] San Diego Supercomputer Center (SDSC). http://www.sdsc.edu/.

[6] B. Chun and D. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers . In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2002.

[7] D. G. Feitelson. Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/.

[8] D. Irwin, L. Grit, and J. Chase. Balancing risk and reward in a market-based task service. In *Proceedings of the IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, 2004.

[9] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for Parallel Job Scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP); held in conjunction with the IEEE International Conference on High Performance Distributed Computing (HPDC)*, 2003.

[10] G. Kochhar. Characterization and enhancements of backfill scheduling strategies. Master's thesis, Ohio State University, 2003.

[11] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *10th Job Scheduling Strategies for Parallel Processing*, June 2004.

[12] A. K. Mok. *Fundamental design problems of distributed systems for the hard real-time environment*. PhD thesis, Massachussetts Institute of Technology, 1983.

[13] A. K. Mok. The design of real-time programming systems based on process models. In *Real Time Systems Symposium*, 1984.

[14] A. K. Mok and M. L. Dertouzos. Multi-Processor Scheduling in a Hard Real-Time Environment. In *Texas Conference on Computing Systems*, 1978.

[15] A. W. Mualem and D. G. Feitelson. Utilization, Predictability, Workloads and User Estimated Runtime Esti mates in Scheduling the IBM SP2 with Backfilling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 12, pages 529–543, 2001.

[16] Krithi Ramamritham, John A. Stankovic, and Perng-Fei Shiah. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. In *IEEE Transactions on Parallel and Distributed Systems*, 1990.

[17] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An Economy driven Job Scheduling System for Clusters. In *Proceedings of the International Conference on High Performance Computing and Grid in Asia*, 2002.

[18] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP2 scheduler using Slack Based Backfilling. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 513–517, April 1997.

[19] C. S. Yeo and R. Buyya. Pricing for Utility-driven Resource Management and Allocation in Clusters. In *Proceedings of the International Conference on Advanced Computing and Communications*, 2004.

[20] C. S. Yeo and R. Buyya. Managing Risk of Inaccurate Runtime Estimates for Deadline Constrained Job Admission Control in Clusters. In *Proceedings of the International Conference on Parallel Processing*, 2006.