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

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

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

[‡]Mathematics and Computer Science,
Argonne National Laboratory
balaji@mcs.anl.gov

Abstract

Quality of service (QoS) mechanisms allowing users to request for 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, once accepted, guarantees these deadlines. 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 a job is profitable only 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. We present detailed analysis of these schemes and demonstrate using simulation that they not only achieve several factors improvement in system revenue, but also good service differentiation as a much desired side-effect.*

1 Introduction

Batch job schedulers are commonly used to schedule parallel jobs at shared-resource supercomputer centers. The typical model for these supercomputer centers is to allocate resources (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 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 [8, 13] in Quality of Service (QoS) in job scheduling in terms of guarantees in the job's turn-around time. Such QoS capability is 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 supercomputer centers provide to achieve this is based on *advance reservations*, whereby at job submission time, the user can request the needed resources during a specific time window (within the desired 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 can try a different time window within the deadline period. However, using *advance reservation* to achieve QoS results in significant under-utilization of resources. In our previous work [5], 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, thus resulting in significant improvements in resource efficiency and the number of accepted jobs.

QoPS attempts to accept as many jobs as possible and does not analyze the costs of the jobs that are being submitted. 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. Suppose a 4-processor job J_4 arrives, requests for 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 forgo the more profitable job. This 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 only 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,

[*]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.

knowing the opportunity cost of a job up front is impossible. Thus, this paper analyzes the impact such opportunity cost can have on the overall revenue of the supercomputer center and attempts to minimize it through predictive techniques. Specifically, we first present an extension of QoPS, named *Value-aware QoPS (VQoPS)*, that analyzes job prices with various statically assumed opportunity cost values for the jobs. 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)* – a self learning variant of VQoPS to analyze past jobs and predict opportunity costs for future jobs. However, if the opportunity cost is decided based on a very long history, the mechanism will lose sensitivity to small pattern changes in the jobs. Similarly, 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 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 service differentiation between the two jobs and give a higher acceptance chance to the more expensive job.

We present detailed analysis of VQoPS and DVQoPS with simulation based on real job workloads from the San Diego Supercomputer Center. Our results provide various insights into the impact of opportunity costs in QoS-aware job schedulers and demonstrate that DVQoPS can provide significantly higher system revenue as well as better service differentiation between high-paying urgent and low-paying non-urgent jobs as compared to QoPS.

2 Related Work

There has been some prior work in providing differentiated service in job scheduling, where different classes of jobs get different statically or dynamically calculated priorities [1, 10]. These approaches provide *best effort* prioritization for jobs or classes of jobs. While such *best effort* prioritization is useful, it is of considerable interest to users to receive *hard* QoS guarantees. In this context, it has been shown that dynamic systems with more than one processor, a polynomial-time optimal scheduling algorithm for real-time deadlines does not exist [7, 6]. In our previous work, QoPS [5], we developed a heuristic based approach to provide hard deadline guarantees to end users in terms of turnaround time. Buyya et. al. [8, 12, 13] have addressed a similar scheme for time shared systems (multiple jobs are scheduled and time sliced on a single node).

There has also been previous work that examines market based approaches, where users might have different goals and preferences that are used to express their desire for service in a common way (e.g., currency) [11, 11, 9]. These follow an auction-based resource allocation mechanism to choose the winner from the bids of different users. Two recent publications [2, 4] have addressed this concept for supercomputer job scheduling. Both papers rely on a per-job specific utility or value function that provides an explicit mapping of service quality to value using a piece-wise linear function that decays with time. The magnitude of the function reflects the job value and a rate of decay reflects the urgency or sensitivity to delay.

Our current work extends previously proposed hard QoS-based scheduling mechanisms by incorporating a market-based approach for revenue, and analyzes this model to understand the impact of opportunity cost.

3 Evaluation Approach

The strategies in this paper are simulated using real workloads (10,000 job subset of the SDSC SP-2 workload) [3] that includes information such as job runtime, number of nodes requested, submission time, and user estimated runtime limit.

Deadline Information: None of the workloads available contains any job deadline information, as hard deadline guarantees are not supported by any supercomputer center. Hence, we synthetically generate them by assuming that users will assign deadlines based on job runtimes, i.e., a job's deadline is assigned to be 5 times the user estimated runtime ($deadline_j = 5 * user\_runtime\_estimate_j$).

Runtime Estimates: We performed two sets of simulations. The first set takes an ideal view of the workload and assumes that users are able to perfectly estimate their job's runtime; this removes the impact of estimation noise in the workload. The second set uses the actual runtime estimates given in the workload allowing evaluation using more realistic environments.

Job Submission Load: In supercomputer centers, the arrival rate of job requests varies with time (in a day, between days, between weeks). To capture this, we carry out simulations with two different offered loads for each workload: (i) the original load as measured in the workload and (ii) a high load emulation achieved by randomly duplicating roughly 40% of the jobs. Duplicated jobs retain the submission time, runtime estimate, runtime, and deadline.

Urgency and Job Cost: Like deadline information, none of the workloads available contains information about job urgency or the amount the user is willing to pay for it. Hence, 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 and that of the urgent jobs' 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 10, 5 and 2 for *C*. 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 similar to that used in previous papers [2, 4], except for deadline guarantees, which were not previously considered.

4 Opportunity Cost in Job Scheduling

As described earlier, each accepted job can force the system to reject future arriving more expensive jobs, i.e., with each accepted job the system pays an opportunity cost. Opportunity cost of a job depends both on the characteristics of the job as well as the workload, as we will describe in section 4.1. In section 4.2, we present a new design that utilizes job characteristics to minimize the impact of opportunity cost and extend this design in section 4.3 to monitor workload characteristics and predict the characteristics of future jobs, thus allowing for improved performance.

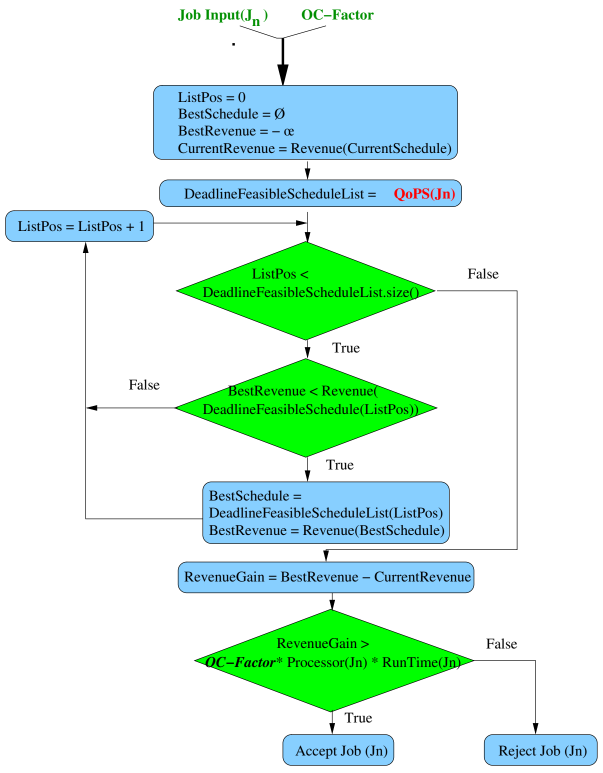
4.1 Job/Workload Characteristics

Opportunity cost of a job depends on two broad aspects: job characteristics and workload characteristics.
Job Characteristics: Two primary characteristics of a job impact its opportunity cost – job shape and job urgency. Job shape determines how many later jobs must be dropped. Large jobs (processor-seconds) typically cause more later arriving jobs to be dropped; thus, their opportunity cost will likely be high. Job urgency determines how stringent the schedule is, and how easier it is to accommodate other jobs. Thus, the opportunity cost of stringent jobs is likely high.
Workload Characteristics: Three primary workload characteristics impact a job's opportunity cost – offered load, job mix and job pricing. When there are few jobs in the system (i.e., low offered load), acceptance of a non-urgent job is less likely to prevent admittance of future urgent jobs. Thus, the opportunity cost would typically be low. Similarly, if all jobs had the same urgency and pricing, opportunity cost of every job will be 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; opportunity cost of admitting a non-urgent job increases with the percentage of urgent high-paying jobs since there is a high probability that its admittance could prevent admission of a future high-paying job. Finally, the higher the relative premium paid by urgent jobs compared to non-urgent jobs, the greater the cost of losing an urgent job, and greater the opportunity cost of non-urgent jobs.

4.2 Value-aware QoPS (VQoPS)

The QoPS algorithm does not perform any differentiation between jobs based on their price. However, in an environment that allows users to offer price based on responsiveness, some jobs 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, it 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. It should maximize overall system revenue during job acceptance by considering both the explicit revenue benefit and the implicit loss of opportunity for the system.



Job Input(J_n)
OC–Factor
ListPos = 0
BestSchedule = Ø
BestRevenue = – ∞
CurrentRevenue = Revenue(CurrentSchedule)
DeadlineFeasibleScheduleList = QoPS(Jn)
ListPos = ListPos + 1
ListPos < DeadlineFeasibleScheduleList.size()
False
True
BestRevenue < Revenue(DeadlineFeasibleSchedule(ListPos))
False
True
BestSchedule = DeadlineFeasibleScheduleList(ListPos)
BestRevenue = Revenue(BestSchedule)
RevenueGain = BestRevenue – CurrentRevenue
RevenueGain > OC–Factor* Processor(Jn) * RunTime(Jn)
False
True
Accept Job (Jn)
Reject Job (Jn)

Figure 1: VQoPS Algorithm Flow Chart

4.2.1 Design of VQoPS

Value-aware QoPS (VQoPS) utilizes some of the characteristics of the job (job shape), to identify its opportunity cost. It uses a static system parameter (OC-Factor), assumes the opportunity cost of the job to be OC-Factor $\times$ job size and analyzes the impact of different OC-Factors on system revenue. Figure 1 shows the high-level VQoPS flowchart. VQoPS utilizes QoPS as a pluggable module to verify when a new job can be accepted. Since QoPS just provides a recommendation on whether a job should be accepted or rejected, we adapted it to instead provide the list of all the acceptable schedules that it finds. 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 is expected to be defined by an administrator as a system parameter. The time complexity of our approach for N jobs is $\theta(K.N^2.log(N))$, where K specifies maximum number of violations allowed in QoPS.

Table 1: Revenue improvement for varying OC-Factors

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

4.2.2 Impact of Workload Characteristics on VQoPS

Table 1 shows the revenue improvements generated by VQoPS (compared to QoPS) for different workload characteristics and different OC-Factors (0.0, 0.05, 0.1, and 0.4). Several insights can be drawn from these results. Primarily, though VQoPS outperforms QoPS by up to 160% in some cases, the improvement is highly dependent on the workload characteristics. That is, no single OC-Factor can consistently provide a good performance for all workloads.

From table 1, 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. Accordingly, the OC-Factor providing the best performance decreases as well. Also, as the number of urgent jobs decreases, smaller OC-Factor values perform better. This is again expected 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. Finally, 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), 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: Self-learning VQoPS

Revenue improvement achieved by VQoPS heavily depends on the workload characteristics. Thus, it is desirable to develop a more sophisticated algorithm that has all the benefits of VQoPS, and can adapt the OC-Factor based on workload characteristics. In this section, we describe a variant of VQoPS which aims to achieve this.

4.3.1 Dynamic Value-aware QoPS (DVQoPS) In VQoPS, to properly tune the OC-Factor, various dynamically changing variables must be considered. 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 cheaper jobs. As the expected revenue difference increases, the OC-Factor should be increased to be more selective in jobs accepted. 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 increase. It is very difficult to manually take these considerations and their interactions into account, especially as they change over time. Thus, it is desirable for the scheduler to automatically generate and adjust the OC-Factor.

In our approach we perform a number of *what-if* simulations over a limited backward window in time, called the *rollback window*. The 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. Such simulation is feasible since the history of arrived jobs (accepted and rejected jobs) is available at any time. For each choice of OC-Factor, we utilize the revenue of each job accepted by the simulated schedule, and thereby estimate the total revenue. The OC-Factor giving the best overall revenue over the window is chosen to be used for the immediate future (until the next OC-Factor change event).

The basic premise of the adaptive OC-Factor selection procedure is that the best OC-Factor depends on time-varying workload characteristics. If DVQoPS is to be effective in adapting to such 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, if the rollback 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 *sensitivity* and *stability*. The rollback window should be short enough to be sensitive to changes in workload characteristics (e.g., load and job mix). Similarly, the rollback window should not be so short that specific jobs affect the best what-if OC-Factor, and rather be stable with respect to identifying aggregate characteristics of the jobs in the window.

For assessing the effect of the rollback window on trace variation, the average offered load is computed over segments of duration equal to the rollback window, and the variance of these averages 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

Table 2: Impact of rollback window size

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

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 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. By varying the length of the rollback window from 1 to 128 hours, the revenue varies from 414M units to 716M units. That is, rollback window size has a large impact on the overall revenue. The second column of the table shows the average of the variance of each set of 5 consecutive OC-Factor choices. With a small *rollback window*, a small change in the considered jobs will result in a very different OC-Factor. This reflects as a higher average variance for the OC-Factor. 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* is used to dynamically vary the *rollback window* size. Every *max rollback window* hours the scheduler runs historical simulations (using *rollback window* sizes of 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 and uses it for the next *max rollback window* hours.

DVQoPS (Figure 2(a)) asynchronously evaluates the rollback interval and OC-Factor after every fixed interval. Figure 2(b) demonstrates the basic steps used to determine the best rollback interval. This is used in Figure 2(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 *what-if* simulations for different values of *candidate OC-Factors*. 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 choices from a set of *T* OC-Factors and *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 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

There are three workload characteristics that affect opportunity cost: (i) urgent job mix, (ii) cost of urgent jobs and (iii) system load. In this section, we vary these characteristics to analyze the behavior of the different schemes.

Simulator Overview: Job scheduling research historically involves running an event based simulator on a synthetic or a historic workload. These workloads contain the runtime, number of nodes, queue or arrival time, and user expected runtime for a set of jobs. Simulations are used to design and test emerging scheduling algorithms before deployment in production systems. 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 [3], simulates the scheduling model and creates an output trace containing data necessary to gather metrics and perform post processing.

5.1 Impact of Urgent Job Mix

Figure 3(a) illustrates the percentage revenue improvement (compared to QoPS) for VQoPS (with different static OC-Factors) and DVQoPS. VQoPS achieves about 20%-45% improvement, with different OC-Factors performing the best for different workload characteristics (i.e., percentage of urgent jobs) – there is no consistently superior OC-Factor value. DVQoPS, on the other hand, consistently achieves within 5% of the best VQoPS implementation for all workloads.

Figure 3(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, and thus cannot maximize revenue. As OC-Factor increases, VQoPS rejects more normal jobs that would have been accepted by QoPS in order to accept urgent jobs, thus reducing the overall acceptance. DVQoPS, on the other hand, accepts more urgent jobs when the opportunity cost is high and more

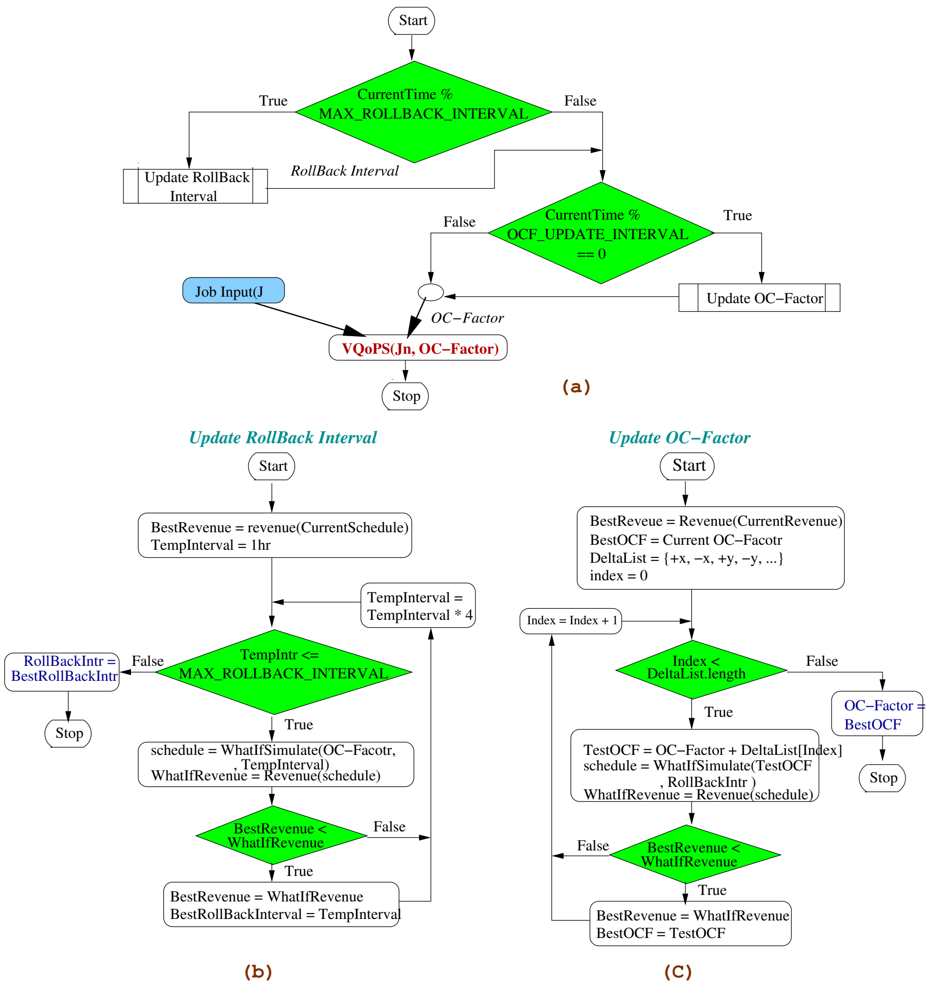
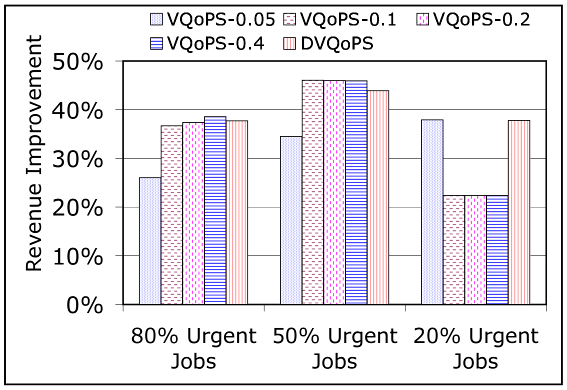

Start
CurrentTime % MAX_ROLLBACK_INTERVAL
True
False
Update RollBack Interval
RollBack Interval
CurrentTime % OCF_UPDATE_INTERVAL == 0
False
True
Job Input(J
Update OC−Factor
OC−Factor
VQoPS(Jn, OC−Factor)
Stop
(a)
Update RollBack Interval
Start
BestRevenue = revenue(CurrentSchedule)
TempInterval = 1hr
TempInterval = TempInterval * 4
RollBackIntr = BestRollBackIntr
False
TempIntr <= MAX_ROLLBACK_INTERVAL
Stop
True
schedule = WhatIfSimulate(OC−Facotr, , TempInterval)
WhatIfRevenue = Revenue(schedule)
BestRevenue < WhatIfRevenue
False
True
BestRevenue = WhatIfRevenue
BestRollBackInterval = TempInterval
(b)
Update OC−Factor
Start
BestReveue = Revenue(CurrentRevenue)
BestOCF = Current OC−Facotr
DeltaList = {+x, −x, +y, −y, ...}
index = 0
Index = Index + 1
Index < DeltaList.length
False
OC−Factor = BestOCF
True
Stop
TestOCF = OC−Factor + DeltaList[Index]
schedule = WhatIfSimulate(TestOCF , RollBackIntr)
WhatIfRevenue = Revenue(schedule)
False
BestRevenue < WhatIfRevenue
True
BestRevenue = WhatIfRevenue
BestOCF = TestOCF
(C)

Figure 2: DVQoPS Algorithm Flow Chart



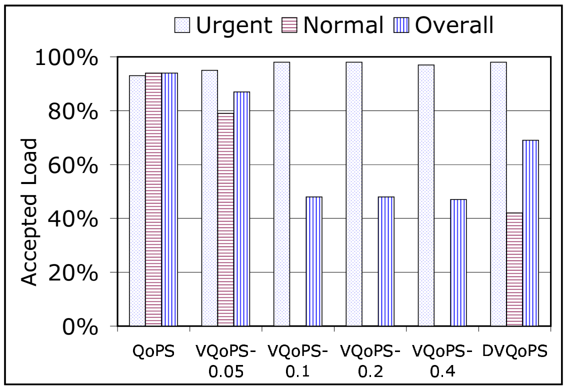
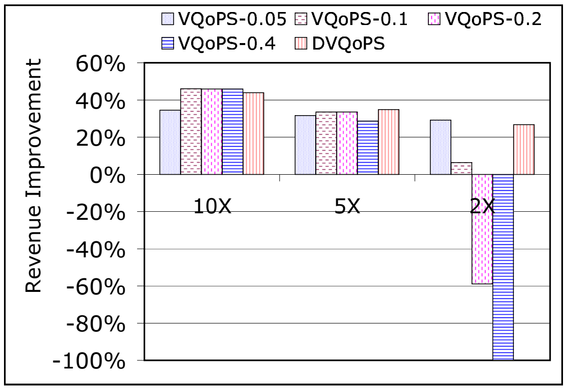


Figure 3: Revenue improvement and Accepted load. Cost of urgent jobs is 10X of normal jobs. In (b), 50% urgent jobs are used.

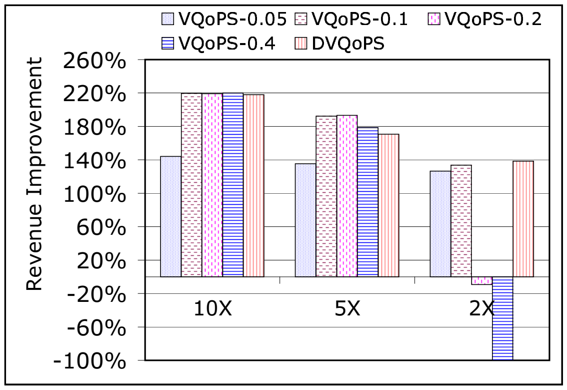
normal jobs when it is low. Thus, it dynamically adapt itself based on the workload characteristics.

5.2 Cost of Urgent Jobs and System Load

Figure 4(a) illustrates the performance of the different schemes as we vary the cost of the urgent jobs. 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 *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 hurt revenue as shown. Again, DVQoPS shows a consistent performance with less than 5% difference as compared to the best static OC-Factor.



(a) Original offered load (89%)



(b) High offered load (125%)

Figure 4: Revenue improvement for original and high offered load for SDSC trace.

Figure 4(b) illustrates a similar trend, but for a high-load scenario. In this case, we observe that the revenue improvements are higher 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 this 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. This shows that our schemes can earn further benefit from extra offered load.

5.3 User Runtime Estimate Inaccuracy

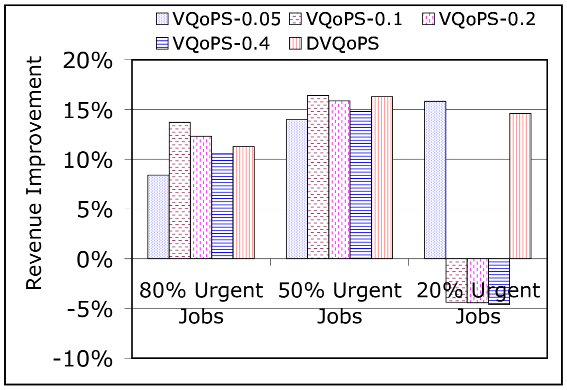
Figure 5(a) shows the performance of the schemes for different urgent job mixes when the original inaccurate user estimates of jobs are used. Compared to a scenario with exact user estimates, we notice that the overall revenues are lower in all cases (especially when the percentage of urgent jobs is 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 find a feasible schedule before accepting the job. Because of this conservative nature of deadline guarantee schemes, the number of accepted jobs is 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, 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 often better to be more lax with the OC-Factor and accept jobs that appear to only modestly increase revenue.

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, VQoPS rejects all the normal jobs; since the number of urgent jobs is very low, the system is left 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 5(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.

Figures 6(a) and 6(b) illustrate the revenue improvement of the different schemes while varying the cost of urgent jobs for original as well as the high offered load. The overall drop in revenue improvement is especially noticeable when the cost of urgent jobs is low. 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. For a higher load, though the improvement is better, compared to exact estimates, there is a drastic degradation.

Overall, 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. Thus, to handle this issue, 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 could potentially require users to improve their runtime estimates and in turn improve the revenue gains VQoPS and DVQoPS can achieve.



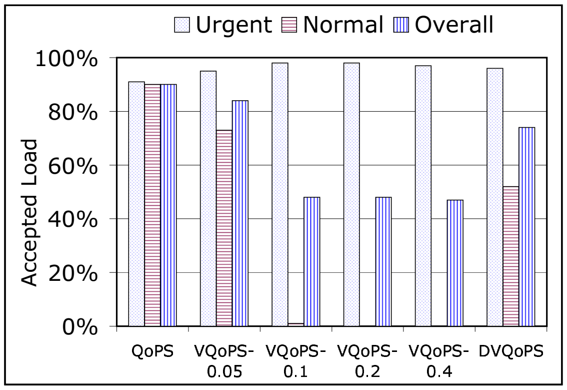
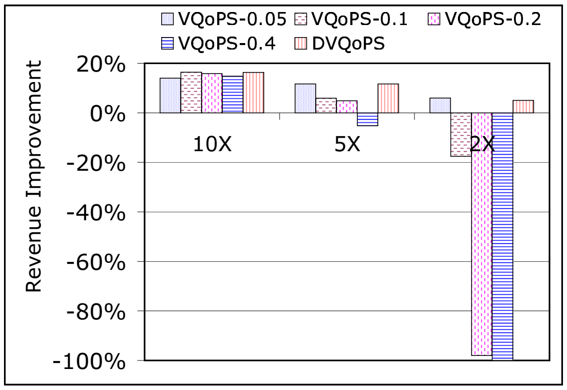
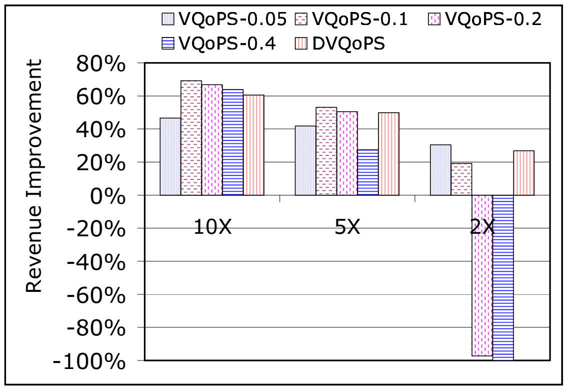


Figure 5: Revenue improvement and Accepted load for inexact estimates. Urgent jobs cost 10X of normal jobs. In (b), 50% urgent jobs are used.



(a) Original offered load (89%)



(b) High offered load (125%)

Figure 6: Revenue improvement with inexact estimates for actual and high offered load for SDSC trace.

6 Conclusions

In this paper, we proposed two extensions to our previous QoS-aware job scheduling mechanism, *QoPS*, 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 more expensive jobs, i.e., it can viewed as paying an implicit opportunity cost. In this paper, we used predictive techniques to identify such opportunity costs and minimize their impact on the overall system revenue. We analyzed our designs and evaluated them using simulation on real workloads from the San Diego Supercomputer Center (SDSC). Our results demonstrated that we not only achieve several factors improvement in system revenue, but also good service differentiation as a much desired side-effect.

References

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

[2] B. Chun and D. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers . In *Proc. CCGrid*, 2002.

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

[4] D. Irwin, L. Grit, and J. Chase. Balancing risk and reward in a market-based task service. In *Proc. HPDC*, 2004.

[5] M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for Parallel Job Scheduling. In *Proc. JSSPP*, 2003.

[6] A. K. Mok and M. L. Dertouzos. Multi-Processor Scheduling in a Hard Real-Time Environment. In *Proc. Texas Conf. Comput. Syst.*, 1978.

[7] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. In *IEEE Trans. Par. Dist. Syst.*, 1990.

[8] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An Economy driven Job Scheduling System for Clusters. In *Proc. Intl. Conf. on High Performance Computing and Grid in Asia*, 2002.

[9] I. Stoica, H. Abdel-Wahab, and A. Pothen. A Microeconomic Scheduler for Parallel Computers. In *Proc. IPPS*, pages 122–135, 1995.

[10] D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP2 scheduler using Slack Based Backfilling. In *Proc. IPPS*, pages 513–517, 1997.

[11] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Trans. on Software Engg.*, 18(2):103–117, 1992.

[12] C. S. Yeo and R. Buyya. Pricing for Utility-driven Resource Management and Allocation in Clusters. In *Proc. Intl. Conf. on Advanced Computing and Communications*, 2004.

[13] C. S. Yeo and R. Buyya. Managing Risk of Inaccurate Runtime Estimates for Deadline Constrained Job Admission Control in Clusters. In *Proc. ICPP*, 2006.