

Adaptive Pricing for Resource Reservations in Shared Environments

Gurmeet Singh, Carl Kesselman, Ewa Deelman
Information Sciences Institute, Marina Del Rey, CA 90292
{gurmeet, carl, deelman}@isi.edu

Abstract

Application scheduling studies on large-scale shared resources have advocated the use of resource provisioning in the form of advance reservations for providing predictable and deterministic quality of service to applications. Resource scheduling studies however have shown the adverse impact of advance reservations in the form of reduced utilization and increased response time of the resources. Thus, resource providers either disallow reservations or impose restrictions such as minimum notice periods and this reduces the effectiveness of reservations as the means of allocating desired resources at a desired time. In this paper, we suggest adaptive pricing as an alternative for allowing reservation of resources. The prices charged for allowing a reservation are based directly on the impact these reservations have on the other users sharing the resources. Using trace-based simulations, we show that adaptive pricing allows users to make reservations at the desired time while making it uneconomical in comparison to the best effort service. Thus users are induced to make the correct choice between reservations and best effort service based on their real needs. Moreover, this pricing scheme is more cost effective and sensitive to the system load as compared to a flat pricing scheme and encourages load balancing across resources.

1. Introduction

Shared distributed infrastructures such as the TeraGrid [1], the Open Science Grid [2], etc, provide the software and hardware resources for facilitating the execution of resource intensive applications from various scientific domains such as astronomy [3], high energy physics [4], earthquake science [5], and others etc. In such shared best effort environments, fairness of allocation and resource utilization is often a major criterion and thus scheduling regimens such as first-come first-serve and its backfill-based variants [6] are often used for allocating resources to tasks. Thus the start time of a task depends on the current workload and is not under control of the user. Moreover, workload characteristics such as inaccurate runtime estimates and job cancellations make it impossible to predict in advance when a task will start. This class of service is

often termed as *best effort*. Users on the other hand often desire more predictability in the availability of resources for their tasks, for example, to meet deadlines or to satisfy co-allocation requirements of their applications. Emerging classes of deadline driven scientific applications such as severe weather modeling [8] require simultaneous access to multiple resources and predictable completion times. Earthquake engineering experiments require coupling and co-allocation of geographically distributed resources [7]. Thus, users of such shared resources have diverse preferences with regards to the performance of their applications and the price that they are willing to pay for that performance.

Provisioning of resources in the form of advance reservations can be used for providing predictable application completion times [9] and co-scheduling application across different resources [10]. By committing to resource availability at a specified time and duration, advance reservations provide a means by which the needs of an application are factored into the local resource management policy. While most local compute resource management systems such as Maui [11], PBS [12], and LSF [13] support advance reservations, such reservations are known to adversely impact the utilization and response time of the resources [14-16]. Moreover, users might try to exploit the system by making advance reservations in order to get their tasks completed earlier while paying the same service charge as the other users whose tasks are queued up. Thus most resource administrators disable user-level advance reservations. Instead, reservations typically are done manually by system administrators and require a few days advance notice. As a result, application schedulers are not able to take advantage of the advance reservation facility and instead depend on prediction services such as the Network Weather Service [17] or queue wait time estimators [18, 19] in order to optimize the application performance.

Any reservation policy that solves this problem should compensate the resource providers for any lost utilization and discourage reservations to be used unless truly required. Both of these objectives can be achieved by a suitable pricing policy for reservations. In this paper, we suggest an adaptive pricing scheme for advance reservations. Reservations are allowed as

long as they do not conflict with any currently running task (in case of non-preemptive scheduling) and other reservations. The queued jobs are ignored in determining the admissibility of a reservation. Thus a reservation can bypass the queued up best-effort jobs. Moreover, it can adversely impact them i.e. cause them to start later than when they would have otherwise. We suggest a reservation price model consisting of two components. The first component is a per unit charge that is greater than or equal to its best effort counterpart. The second component is designed to account for the adverse impact of the reservation on the best effort jobs and is based on the estimated increase in completion time of the queued best effort jobs that get delayed due to this reservation. Since this second component is based on the current state of the resource, we call this pricing scheme adaptive. This pricing scheme is motivated by the work in priority pricing [20-23] where different priority classes exist and the admission price of a priority class is based on the expected impact on this class on the lower classes.

The goal of our pricing scheme is for the resource providers to allow the users to reserve resources without any abuse of the system. To this end, using trace-based simulations, we show that our pricing scheme makes reservations more expensive than best effort service. The price of a reservation seems to vary inversely with the start time of the reservation with early start times costing more than late start times. Any start time advantage over best effort service translates into a significant price differential. Thus reservations would be made by the users only when absolutely required. This need for a reservation is measured by the willingness of the user to pay a premium price for the reservation. We also show that our adaptive pricing is more cost-effective than a static pricing scheme and a significant percentage of the users pay less than the static price. Moreover, the prices are sensitive to the current load on the resource and increase as the resource becomes overloaded.

The organization of the paper is as follows. The pricing algorithm is described in Section 2. Section 3 discusses the start time selection for reservations. Evaluation of the pricing algorithm using trace-based simulations is presented in Section 4. We discuss the priority pricing work that motivated our pricing algorithm and other related work in Section 5 followed by conclusions in Section 6.

2. Pricing Algorithm

This section describes the algorithm for determining the pricing for advance reservations. We model a reservation request to consist of a desired start time s , number of processors required n and the duration for which the processors are required d .

$$ResvQuery(s, n, d) = \langle p, f \rangle$$

The resource response consists of a multiplicative cost component p and an additive component f . The implied total cost of the reservation is $(n \times p \times d + f)$. The multiplicative component p represents the basic unit charge e.g. one service unit (SU) per processor hour for the resource and might be the same as the charge for best effort service. Since we allow a reservation to be granted delaying the queued best effort jobs, the additional charge, f , is added to account for the delay caused to the best effort jobs if any. The pricing algorithm described in this section determines this additional component (f) while the multiplicative part is assumed to be a constant for the resource provider. In order to determine the delay caused to any job, the algorithm computes the schedule of the queued best effort jobs with and without the reservation in place using the resource scheduling policy. By comparing the start times of the jobs in the two schedules, we can find the jobs, which were delayed and by how much. The additional charge (f) is then set equal to the sum of the delays of the delayed jobs multiplied by the number of processors requested by these jobs. Thus if the reservations does not delay any jobs according to the computed schedules, then the additional charge is zero.

In order to illustrate the working of the algorithm, a resource with 5 processors is shown in Figure 1. Tasks A and B are currently running. C and D are queued tasks and R is a reservation. A reservation X for 2 processors and 3 time units is desired starting at time 2. Figure 1 shows the resource schedule with (bottom) and without (top) the reservation in place. As a result of putting the reservation in, Task C gets delayed by 1 time unit and task D gets delayed by 2 time units. Thus the charge to the reservation X would be $(1 \times 2) + (2 \times 2) = 6$ since both Task C and D require two processors. The algorithm is more formally listed in Figure 2. The asymptotic complexity of the algorithm is $O(S + n)$ where S is the runtime complexity of the resource scheduling algorithm and n is the number of the queued up jobs.

The user is free to ask for a reservation starting at any particular time. It is possible that a reservation cannot be done at a particular start time due to conflicts with currently running jobs and existing reservations. The price in that case would be ∞ indicating non feasibility of the desired start time. For example, in Figure 1, the reservation cannot be done before time 2 if preemption of currently running tasks is not allowed. This is also the assumption in the algorithm in Figure 2. However, we can drop this assumption and add the currently running tasks to the list of queued tasks in order to provide more flexibility in adding reservations. However, most production systems do not allow

preemption of running tasks and hence we do not model preemption in our algorithm.

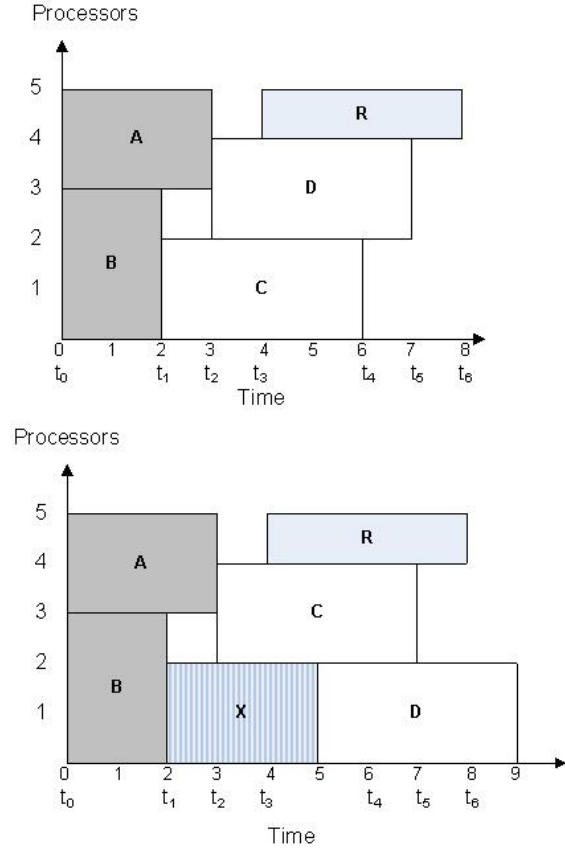


Figure 1. Resource schedule with (bottom) and without (top) the reservation in place.

Input: reservation start time s , processors p , duration d
 Input: set of queued tasks Q , scheduling policy π
 Input: existing reservations and running tasks.
 Output: a price for the reservation, f

1. Create a schedule of the tasks in Q using π . Let $v_q =$ start time of task q in Q as per this schedule.
2. Add the reservation at time s .
3. If the reservation cannot be added at s due to conflicts with running tasks and other reservations, return ∞ .
4. Recreate a schedule of the tasks in Q using π . Let $v'_q =$ start time of task q in Q as per this schedule
5. Return $f = \sum_{q \in Q} \{(v'_q - v_q)^+ \times n_q\}$ as the price

for the reservation where n_q is the number of processors required by task q and $(v'_q - v_q)^+$ is defined as

$$(v'_q - v_q)^+ = \begin{cases} (v'_q - v_q), & \text{if } v'_q \geq v_q \\ 0, & \text{otherwise} \end{cases}$$

Figure 2. Reservation pricing algorithm.

3. Reservation start times

A user might wish to examine different possible start times for a desired reservation in order to optimize the price to be paid and/or the utility derived from the reservation. For example, the user might want to know the earliest possible time when a reservation can start or the time when the additive cost of a reservation will be zero i.e. when the resources can be reserved without delaying any other jobs. In order to provide more possibilities for the user, we model a feature where the resource sends back a set of possible start times for a desired reservation and the prices at these start times. The response to a reservation query for n processors for d duration can be represented as

$$ResvQuery(n, d) = \{\dots, \langle s_i, p_i, f_i \rangle, \dots\}$$

where s_i denotes a possible start time with associated price components of p_i and f_i as explained earlier. The price component p_i is assumed to be a constant for a resource and we do not discuss it further. For the rest of this paper, the term price always refers to the additive price (f_i) unless otherwise mentioned. The response should include the earliest start time and one or more start times when the price would be zero. One possible approach for generating the list of possible start times that we use for the experiments in the next section is to create a schedule of all the running and queued jobs and existing reservations. We examine this schedule from the current time to the end of the schedule and whenever there is a discontinuity in the resource availability e.g. as a result of tasks or reservations starting or finishing, we include that time instant into the list of possible start times. The intuition for picking these start times is that the availability of the resource changes at these time instants and hence the price would be likely affected too. The current time is also included in the list since it is possible that the reservation might be able to start right away if enough resources are available. Then the price for each start time is computed using the algorithm in Figure 2 and the list of start times along with the price for each of them is sent back to the user. Since the earliest possible time for a reservation is always either the current time or the end time of a job or reservation, this list also contains this earliest start time for the reservation. The list also contains at least one time when the additive price is zero, for example at the maximum finish time of any job or reservation. At this time the resource is empty and the reservation can be done without delaying any jobs.

As an illustration, based on the resource schedule shown in Figure 1(top), the list of start time would include times $\{t_0, t_1, t_2, t_3, t_4, t_5, t_6\}$. t_0 is an unfeasible start time and the price for the other start times is shown in Table 1 based on the delays to jobs C and D.

The table shows that an earlier start time does not always mean a larger cost e.g. t_3 is costlier than t_2 which is more costlier than t_1 . The reservation can be granted at any time from t_4 to t_6 without delaying any queued task and thus have a price of 0. Thus the resource response can be very useful for the user to determine a desirable start time for the reservation. However the user is not constrained by these start times and can always self select a start time for the reservation provided that he/she is willing to pay the resulting additional cost.

Table 1. Reservation cost from t_1 to t_6 .

Time instants	Delay for task C	Delay for task D	Additive cost(f_i)
t_1	1	2	$1 \times 2 + 2 \times 2 = 6$
t_2	1	3	$1 \times 2 + 2 \times 3 = 8$
t_3	1	4	$1 \times 2 + 2 \times 4 = 10$
t_4	0	0	0
t_5	0	0	0
t_6	0	0	0

4. Experiments

In order to evaluate the performance of the adaptive pricing scheme for advanced reservations, we performed experiments using trace-based simulations. The simulations were done using workload traces of an IBM SP2 128 processor system and a CTC 430 processor system from the parallel workload archives [24]. These workloads have been used extensively for scheduling studies [6] and to characterize the effect of reservations on best effort jobs [14, 16]. In this paper, we evaluate the impact of the best effort jobs on the advance reservations for these workload traces in the terms of the pricing and start time of these reservations. We use aggressive backfilling [6] as the resource scheduling policy for the best effort jobs in the simulations.

Each resource was simulated independently. To obtain a reservation, a query was first sent to the resource being simulated and a response including a set of possible start times and prices is obtained using the approach described in Section 3. A possible start time is selected from this set using a preference factor α that varies between 0 and 1 such that it minimizes the following cost function.

$$\min_{\langle s_i, f_i \rangle} \left\{ \alpha \times \left(\frac{f_i - f_{\min}}{f_{\max} - f_{\min}} \right) + (1 - \alpha) \times \left(\frac{s_i - s_{\min}}{s_{\max} - s_{\min}} \right) \right\}$$

$f_{\min(\max)}$ = minimum(maximum) price in the set

$s_{\min(\max)}$ = earliest(latest) start time in the set

$\alpha = 0$ signifies that the user wants the earliest possible start time irrespective of cost. $\alpha = 1$ signifies the other extreme where the user wants to minimize the cost irrespective of start time. Note that when $\alpha = 1$, there might be several start times that correspond to the same minimum price. In that case, we take the minimum start time among those. Because of the fact that the resource response always contains at least one start time with a zero cost, $\alpha = 1$ always selects a start time with zero cost. Different values of α model different sensitivities of the user towards price and start time. Each experiment was run using a single value of α only unless otherwise stated.

The complexity of the algorithm in Figure 2 was mentioned earlier. Experimentally it never took more than 5 ms for the SP2 and 30 ms for the CTC cluster to execute this algorithm on a dual processor Pentium 4 2 GHz machine. Creating a set of possible start times and price take longer because a number of start times need to be evaluated instead of just one. However, it never took more than 20 seconds to create the set of start times and prices and this time was much less than the average inter-arrival time of reservations that was around 30 minutes.

Due to the computational complexity of the resource scheduling and the pricing algorithms we limit the simulations to a 30 day workload trace from the beginning of the trace. This 30 day trace had 2888 jobs in the SP2 trace and 7588 jobs in the CTC trace. In the experiments, 10% of the tasks were selected using random sampling and executed using reservations. For the tasks simulated as reservations, we use the actual task runtime as the requested duration for the reservation. The reason is that since a user would need to pay for the reservation irrespective of use unlike best effort service, we assume the user would use tight estimates of the task runtimes. The prices shown in this section are computed using the algorithm in Section 2 and are in service units (SU) where one SU is equal to one processor hour.

For the first experiment, the value of α was varied from 0 to 1 in increments of 0.1. Figure 3 shows the average price paid by the reservations, the average wait time of the reservations and the average queue wait time of the best effort jobs. The wait time of a reservation is the difference between the time when the reservation query was made and the start time of the reservation. For the CTC system, the reservation wait time is less than the best effort wait time for small values of α . However, the price is significantly higher. For the SP2 system, the reservation wait time is more than or about the same as the best effort wait time even for small values of α . There may be two reasons for this behavior. First, the CTC runtime estimates are better than the SP2 runtime estimates. The average CTC job runtime is 40.4% of

the estimated run time whereas it is 33.9% for the SP2 system. Second, the SP2 had a higher resource load 63.4% as compared to the CTC load of 58.7%. When there is a start time advantage with reservation i.e. wait time of reservation is less than the average best effort wait time, a significant price premium is charged for the reservation.

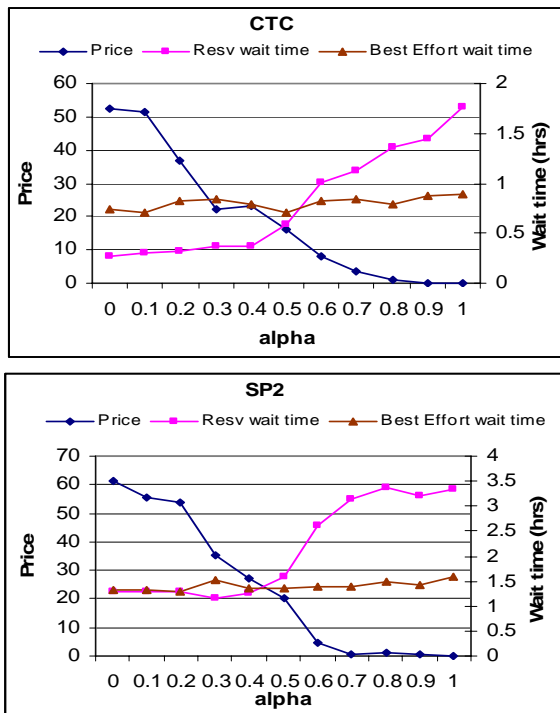


Figure 3. Average price and wait times of reservations and best effort jobs for different α .

When reservations do not cost extra i.e. $f = 0$ at $\alpha = 1$, the average wait time for the reservations is significantly higher than that of the best effort jobs. The reason is that best effort jobs take advantage of backfilling opportunities created when tasks finish before their specified wall clock time to jump ahead and finish earlier than otherwise predicted. However, the reservations are fixed in time and cannot move forward to take advantage of these opportunities. Thus, when the reservation price is zero, there are no start time advantage with the reservation. Thus under this pricing scheme there doesn't appear to be any advantage to the user to make reservations over using the best effort service unless a specific requirement for the reservation exists. When such a requirement for reservation exists, a price-sensitive user is well off making the reservation well in advance when the price for the reservation would be low. This pricing scheme is somewhat similar to advance purchase discount pricing [25] that is widespread in the airline and the hotel industry.

In order to examine the effect of the resource load on the prices and wait times of the reservations, we increase the resource utilization by duplicating 20% and 40% of the tasks, selected randomly, in the simulated workload. As a result the utilization of the CTC system rises from 58.7% to 68.4% and 78.8% respectively and that of the SP2 system rises from 63.4% to 77.8% and 87.4% respectively. We keep the percentage of reservations in the final workload to 10%. Figure 4 shows the increased prices and wait times of reservations when the utilization is increased. As resource utilization increases, the average prices for reservations and the start times when these reservations can be provided increase, reflecting the current state of the resource. This shows the adaptiveness of the pricing algorithm to the current state of the resource. As the resource comes under high utilization, the high prices would tend to deflect the reservations towards less utilized resources in the Grid.

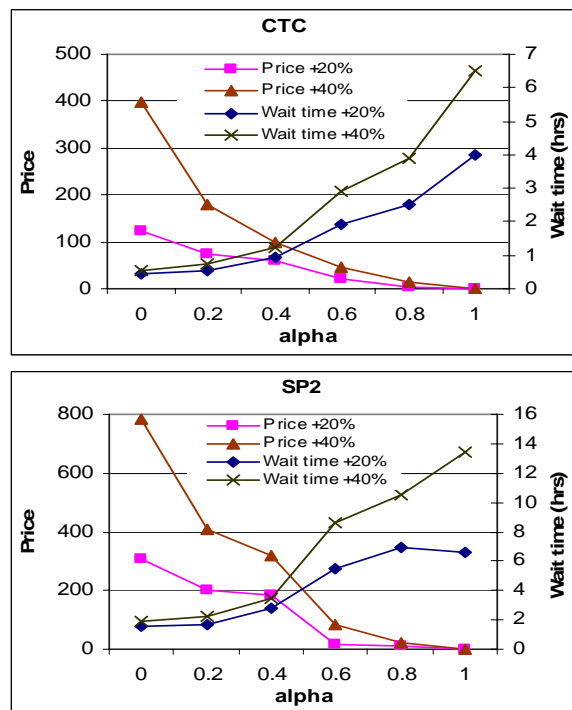


Figure 4. Average prices and start times of reservations for different utilization and α .

The resource scheduling policy for the best effort jobs would also affect the reservation price since it determines the job placements and hence the resulting delay due to a reservation. We also implemented conservative backfilling (CB) [6] and compared the resulting reservation prices to that of the aggressive backfilling (AB) that has been used for the preceding experiments. Conservative backfilling allows a job to be executed out of order only when it would not delay any job ahead of it in the queue while aggressive backfilling allows out of order execution only when it

would not delay the first job in the queue. Thus, AB provides more freedom than CB in scheduling jobs from the queue.

In these experiments, the value of α for determining the start time of a reservation was randomly chosen between 0 and 1. Figure 5 shows the average reservation price using the aggressive and conservative backfilling for the CTC and SP2 systems with different utilization levels. The average reservation price under aggressive backfilling is generally greater than that of conservative backfilling. The reason is that aggressive backfilling packs the schedule more tightly than conservative backfilling due to the additional flexibility in placement of jobs and thus a reservation under aggressive backfilling causes more perturbation than under conservative backfilling. Moreover, this difference in price becomes accentuated as the resource utilization increases. Aggressive backfilling is more widely used than conservative backfilling since it has been shown to result in faster average turnaround time [6]. Thus we have used aggressive backfilling as the default scheduling policy in our experiments.

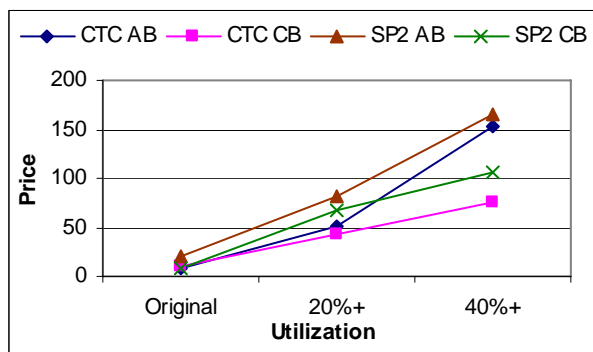


Figure 5. Price Effect of Aggressive (AB) and Conservation Backfilling (CB).

Some resources often implement a high priority queue. Jobs from this queue are executed before tasks from the low priority queue but are also charged at double the unit rate. Thus assuming that the total cost of using the high priority queue is $2nxd$ and that of the best effort queue is $1nxd$, the additional price (over the best effort) for the high priority queue is $1nxd$ where n is the number of requested processors and d is the duration. In order to compare our reservation pricing with that of a high priority queue, we convert the reservation price f to a unit rate ρ by dividing it by $n \times d$ where n is the number of processors requested and d is the duration. Thus $f = \rho nxd$. Note that ρ for a high priority queue is 1. Figure 6 shows the cumulative distribution of ρ for reservations when they are selected to start at the earliest possible time ($\alpha = 0$) under various resource utilization levels. Each point shows the percentage of users (in Y axis) who pay no more than the unit price (ρ) shown in the X axis. Since a high

priority queue has $\rho = 1$, Figure 6 shows that under the original utilization level, around 75% of the reservations pay no price and 80% pay less than the rate of the high priority queue.

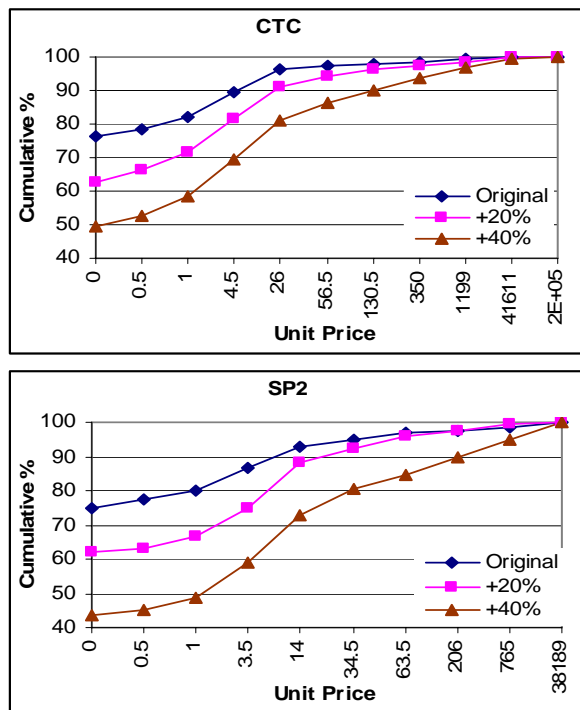


Figure 6. Cumulative price distribution of reservation on a per unit basis.

As the resource utilization goes up by 20%, the percentage of users who pay less than the high priority queue rate is around 70% for both the systems. When utilization is increased by 40%, the percentage of users who pay less than the high priority queue rate is 58% and 48% for the CTC and SP2 systems respectively. Thus the reservation pricing scheme is more cost effective for the users than a priority queue system since a large percentage of the users pay less than the high priority queue rate. The top 10-20% of the reservations seem to pay a very high unit rate in comparison to a high priority queue in Figure 6. However, in practice, when faced with a high price, users are likely to choose the priority or the best effort queue or choose another less loaded resource provider. Thus prices can help in load balancing between the resources.

A high priority queue would however, generally lead to a faster turnaround time than a reservation system even when the earliest start time is chosen. The reason is that the reservation start time is computed based on the maximum run time estimates of the best effort jobs and once granted, it cannot be shifted in time. However, jobs from the high priority queue can be backfilled when a job finishes earlier than estimated leading to a

faster turnaround time. If the exact runtime estimates were available, then the turnaround time of the high priority queue and that of reservation with earliest start time would be similar.

Multiple priority queues can also coexist with the reservation system described in this paper. The algorithm described in Section 2 can be extended to account for delays to jobs in more than one queue. Each queue will have a delay cost associated with it and the delay caused to the jobs in the queue would be weighted by this delay cost factor in determining the price for a reservation.

We also increased the percentage of reservations from 10% to 30% in the workload and found the price for the reservations to decrease slightly overall and the average wait time of best effort jobs to increase. This is due to the corresponding reduction in the best effort workload. When the entire workload is composed of reservations, the price would become zero since there is no best effort traffic that would get delayed because of reservations.

It is also important to note that in a system where the runtimes of the queued and running jobs are not accurately known, the delays computed by the pricing algorithm are only an estimate since they depend on the maximum runtime estimate of these jobs. If the runtimes were accurately known, then the delay calculation would be accurate and one can even think of making refunds to the best effort jobs based on how much the job was delayed due to a reservation. The refund can be done from the price charged to a reservation. However, due to the inaccurate runtime estimates, some jobs which were supposed to get delayed by a reservation might end up executing sooner than anticipated due to backfilling and this change of run order might delay some jobs more than anticipated. Thus in such a stochastic environment, the exact delays caused to jobs because of a reservation cannot be determined in advance. However, the prices computed with maximum runtimes still have the desired effect of providing cost differentiation between reservations and best effort service while being sensitive to the resource load.

5. Related Work

Pricing of services in priority based systems have received a significant amount of attention during the seventies and eighties [20-23]. In these studies, a delay cost is associated with each user or job where the delay cost is defined as the amount of money the user will be willing to spend to get the job finished one time unit earlier. Thus each user or job incurs an implied waiting cost which is the product of the waiting time for service and the delay cost. Price discrimination has been shown to provide efficient allocation of resources

minimizing the total expected waiting costs in the system when demand exceeds capacity and users differ in their delay costs. This is achieved by creating a number of priority classes and setting the price of each priority class based on the expected increase in the waiting costs of lower priority classes due to this class. Each user or job selects a class for service that minimizes the sum of the price paid and the expected waiting cost using that class. In an M/M/1 system, this price scheme has also been shown to be incentive compatible in the sense that it makes it optimal for the users to make decisions based on their true delay costs [26].

Delay cost pricing as summarized in the previous paragraph has been the motivation for the adaptive pricing scheme developed in this paper. However, there are important differences. First, we don't explicitly elicit the delay cost of the best effort users whose jobs get delayed. Instead, the size (number of processors) is used as an implicit delay cost in the pricing algorithm described in Section 2. Thus there is a greater penalty for delaying larger jobs than smaller ones. Getting users to specify their actual perceived delay costs in the current framework would require a pricing scheme that is provably incentive compatible and is the subject of future work. Second, the analysis of pricing schemes of dynamic systems [20, 26] is generally performed under steady state conditions using expected values of arrival and service rates with an assumed distribution. The resulting prices are constant for a priority class. We have done the pricing analysis under a static condition by taking a snapshot of the system and using currently queued jobs and their characteristics. The reason is that it is difficult to assign a distribution and its parameters that accurately model the job and reservations arrivals and their characteristics. Instead, our pricing algorithm is more responsive to the current state of the resource. For example, the user does not have to pay a premium when the system is empty encouraging use of an underloaded system and the premium is significantly high when the resource is overloaded causing demand to shift to the less loaded resources or greater start times. Moreover, our pricing scheme does not assume any particular job scheduling algorithm and allows the resource provider to use any scheduling algorithm for the best effort jobs.

6. Conclusion

Advance reservations allow users to get deterministic quality of service from resources. Yet, the resource providers disallow reservations because these can be used to get unfair advantage over the other users and tend to degrade the quality of service for other users by increasing their response times. In this paper we presented an adaptive pricing scheme for reservations

that charges a premium for a reservation based on the adverse impact of the reservation on the best effort jobs. Thus it encourages users to use reservations carefully. The need for a reservation is indicated by the willingness of the user to pay a premium price for the reservation or to accept a reservation start time that gives little advantage over the best effort queue. The pricing scheme is also more cost effective and adaptive to the current system load than a flat price high priority queue system. In the future, we plan to explore the incentive compatibility of this pricing scheme.

7. Acknowledgements

This work is supported by the National Science Foundation under Cooperative Agreement CCR-0331645 and NGS-0305390 and CNS-0615412.

8. References

- Catlett, C. *The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility*. in *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*. 2002.
- The Open Science Grid Consortium*, <http://www.opensciencegrid.org>.
- Katz, D.S., et al. *A Comparison of Two Methods for Building Astronomical Image Mosaics on a Grid*. in *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*. 2005.
- Deelman, E., et al. *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists*. in *11th Intl Symposium on High Performance Distributed Computing*. 2002.
- Maechling, P., et al., *Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment SIGMOD Rec.*, 2005 **34** (3): p. 24-30
- Mu'alem, A.W. and D.G. Feitelson, *Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*. *Parallel and Distributed Systems, IEEE Transactions on*, 2001. **12**(6): p. 529-543.
- Pearlman, L., et al. *Distributed hybrid earthquake engineering experiments: experiences with a ground-shaking grid application*. in *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*. 2004.
- Droegemeier, K.K., et al. *Linked Environments for Atmospheric Discovery (LEAD): A CyberInfrastructure for Mesoscale Meteorology Research and Education*. in *20th Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*. 2004. Seattle, WA.
- Wieczorek, M., et al. *Applying Advance Reservation to Increase Predictability of Workflow Execution on the Grid*. in *Second IEEE International Conference on e-Science and Grid Computing*. 2006. Amsterdam.
- Kenneth Yoshimoto, P.K., Phil Andrews. *Co-Scheduling with User-Settable Reservations*. in *11th Workshop on Job Scheduling Strategies for Parallel Processing*. 2005. Cambridge, MA.
- Maui Cluster Scheduler*, <http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php>.
- PBSPro*, <http://www.pbspro.com>.
- Zhou, S., et al., *Utopia: a load sharing facility for large, heterogeneous distributed computer systems* *Softw. Pract. Exper.*, 1993 **23** (12): p. 1305-1336
- Smith, W., I. Foster, and V. Taylor. *Scheduling with advanced reservations*. in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. 2000.
- Cao, J. and F. Zimmermann. *Queue scheduling and advance reservations with COSY*. in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. 2004.
- Heine, F., et al., *On the Impact of Reservations from the Grid on Planning-Based Resource Management*, in *Computational Science - ICCS*. 2005, Springer Berlin. p. 155-162.
- Wolski, R., N. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*. *Future Generation Computer Systems*, 1999. **15**(5-6): p. 757-768.
- Brevik, J., D. Nurmi, and R. Wolski. *Predicting bounds on queuing delay for batch-scheduled parallel machines*. in *ACM Symposium on Principles and Practice of Parallel Programming*. 2006. New York.
- Downey, A.B. *Predicting queue times on space-sharing parallel computers*. in *Proceedings of the 11th International Parallel Processing Symposium*. 1997.
- Marchand, M.G., *Priority Pricing*. *Management Science*, 1974. **20**(7): p. 1131-1140.
- Harris, M. and A. Raviv, *A Theory of Monopoly Pricing Schemes with Demand Uncertainty*. *American Economic Review*, 1981. **71**(3): p. 347-365.
- Mendelson, H., *Pricing Computer Services: Queueing Effects*. *Communications of the ACM*, 1985. **28**(3): p. 312-321.
- Wilson, R., *Efficient and Competitive Rationing*. *Econometrica*, 1989. **57**(1): p. 1-40.
- Feitelson, D.G., *Logs of real parallel workloads from production systems*, in URL: <http://www.cs.huji.ac.il/labs/parallel/workload>.
- Gale, I.L. and T.J. Holmes, *Advance-Purchase Discounts and Monopoly Allocation of Capacity*. *American Economic Review*, 1993. **83**(1): p. 135-146.
- Mendelson, H. and S. Whang, *Optimal Incentive-Compatible Priority Pricing for the M/M/1 Queue*. *Operations Research*, 1990. **38**(5): p. 870-883.