QoPS: A QoS based scheme for Parallel Job Scheduling*

Mohammad Islam, Pavan Balaji, P. Sadayappan, and D. K. Panda

Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210, USA
{islammo, balaji, saday, panda}@cis.ohio-state.edu

Abstract. Although job scheduling has been much studied, the issue of providing deadline guarantees in this context has not been addressed. In this paper, we propose a new scheme, termed as *QoPS* to provide Quality of Service (QoS) in the response time given to the end user in the form of guarantees in the completion time to submitted independent parallel jobs. To the best of our knowledge, this scheme is the first one to implement admission control and guarantee deadlines for admitted parallel jobs.

Keywords: *QoS, Job Scheduling, Deadlines, Parallel Job Scheduling*

1 Introduction

A lot of research has focused on the problem of scheduling dynamically-arriving independent parallel jobs on a given set of resources. The metrics evaluated include system metrics such as the system utilization, throughput [4, 2], etc. and users metrics such as turnaround time, wait time [9, 14, 3, 6, 13, 8], etc. There has also been some recent work in the direction of providing differentiated service to different classes of jobs using statically or dynamically calculated priorities [16, 1] assigned to the jobs.

However, to our knowledge, there has been no work addressing the provision of Quality of Service (QoS) in Parallel Job Scheduling. In current job schedulers, the charge for a run is based on the resources used, but is unrelated to the responsiveness of the system. Thus, a 16-processor job that ran for one hour would be charged for 16 CPU-hours irrespective of whether the turn-around time were one hour or one day. Further, on most systems, even if a user were willing to pay more to get a quicker turn-around on an urgent job, there is no mechanism to facilitate that. Some systems, e.g. NERSC [1] offer different queues which have different costs and priorities: in addition to the normal priority queue, a high priority queue with double the usual charge, and a low priority queue with half

* This research was supported in part by NSF grants #CCR-0204429 and #EIA-9986052

the usual charge. Jobs in the high priority queue get priority over those in the normal queue, until some threshold on the number of serviced jobs is exceeded. Such a system offers the users some choice, but does not provide any guarantee on the response time. It would be desirable to implement a charging model for a job with two components: one based on the actual resource usage, and another based on the responsiveness sought. Thus if two users with very similar jobs submit them at the same time, where one is urgent and the other is not, the urgent job could be provided quicker response time than the non-urgent job, but would also be charged more.

We view the overall issue of providing QoS for job scheduling in terms of two related aspects, which however can be decoupled:

- **Cost Model for Jobs**: The quicker the sought response time, the larger should be the charge. The charge will generally be a function of many factors, including the resources used and the load on the system.
- **Job Scheduling with Response-time Guarantees**: If jobs are charged differently depending on the response time demanded by the user, the system must provide guarantees of completion time. Although deadline-based scheduling has been a topic of much research in the real-time research community, it has not been much addressed in the context of job scheduling.

In this paper, we address the latter issue (Job Scheduling with Response-time Guarantees) by providing Quality of Service (QoS) in the response time given to the end-user in the form of guarantees in the completion time to the submitted independent parallel applications. We do not explicitly consider the cost model for jobs; the way deadlines are associated with jobs in our simulation studies is explained in the subsequent sections.

At this time, the following open questions arise:

- *How practical is a solution to this problem?*
- *What are the trade-offs involved in such a scheme compared to a non-deadline based scheme?*
- *How does the imposition of deadlines by a few jobs affect the average response time of jobs that do not impose any deadlines?*
- *Meeting deadlines for some jobs might result in starvation of other non-deadline jobs. Does making the scheme starvation free by providing artificial deadlines to the non-deadline jobs affect the true deadline jobs?*

We study the feasibility of such an idea by providing a framework, termed as *QoPS* (Standing for **Qo**S for **P**arallel Job **S**cheduling), for providing QoS with job schedulers; we compare the trade-offs associated with it with respect to the existing non-deadline based schemes. We compare it to adaptations of two existing algorithms - the *Slack-Based (SB) algorithm* [16] and the *Real-time (RT) algorithm* [15], previously proposed in different contexts. The SB algorithm [16] was proposed as an approach to improve the utilization achieved by a back-filling job scheduler. On the other hand, the RT algorithm [15] was proposed in order to schedule non-periodic real-time jobs with hard deadlines, and was evaluated

in a static scenario for scheduling uni-processor jobs on a multiprocessor system. As explained later, we adapted these two schemes to schedule parallel jobs in a dynamic job scheduling context with deadlines.

The remaining part of the paper is organized as follows. In Section 2, we provide some background on deadline based job scheduling and how some schemes proposed in other contexts may be modified to accommodate jobs with deadlines. In Section 3, we discuss the design and implementation of a new scheduling scheme that allows deadline specification for jobs. The simulation approach to evaluate the schemes is discussed in Section 4. In Section 5, we present results of our simulation studies comparing the various schemes. In Section 6, we conclude the paper.

2 Background and Related Work

Most earlier schemes proposed for scheduling independent parallel jobs dealt with either maximizing system metrics such as the system utilization, throughput, etc., or minimizing user metrics such as the turnaround time, wait time, slowdown, etc., or both. Some other schemes have also looked at prioritizing the jobs based on a number of statically or dynamically determined weights. In this section, we review some of the related previous work and propose modifications to these to suit the problem we address.

2.1 Review of Related Work

In this subsection we describe some previous related work. In the next subsection, we show how these schemes can be modified to accommodate jobs with deadlines.

Slack-Based (SB) Algorithm The Slack-Based (SB) Algorithm [16], proposed by Feitelson et. al., is a backfilling algorithm used to improve the system throughput and the user response times. The main idea of the algorithm is to allow a slack or laxity for each job. The scheduler gives each waiting job a precalculated slack, which determines how long it may have to wait before running: 'important' and 'heavy' jobs will have little slack in comparison with others. When other jobs arrive, each job is allowed to be pushed behind in schedule time as long as its execution is within the initially calculated slack.

The calculation of the initial slack involves cost functions taking into consideration certain priorities associated with the job. This scheme supports both user selected and administrative priorities, and guarantees a bounded wait time for all jobs.

Though this algorithm has been proposed for improving the system utilization and the user response times, it can be easily modified to support deadlines by fixing the slack appropriately. We propose this modified algorithm in Section 2.2.

Real Time (RT) Algorithm It has been shown that for dynamic systems with more than one processor, a polynomial-time optimal scheduling algorithm does not exist [11, 10, 12]. The Real Time (RT) Algorithm [15], proposed by Ramamritham et. al., is an approach to schedule uni-processor tasks with hard real time deadlines on multi-processor systems. The algorithm tries to meet the specified deadlines for the jobs by using heuristic functions. The tasks are characterized by worst case computation times, deadlines and resource requirements. Starting with an empty partial schedule, each step in the search extends the current partial schedule with one of the tasks yet to be scheduled. The heuristic functions used in the algorithm actively direct the search for a feasible schedule i.e., they help choose the task that extends the current partial schedule. Earliest Deadline First and Least Laxity First are examples of such heuristic functions.

In order to accommodate this algorithm into the domain of scheduling dynamically arriving parallel jobs, we have made two modifications to the algorithm. The first one is to allow parallel jobs to be submitted to the algorithm and the other is to allow dynamically arriving jobs. The details of the modified algorithm are provided in Section 2.2.

2.2 Modifications of Existing Schemes

In this section we propose modifications to the Slack-Based and Real-Time algorithms to support deadlines for parallel jobs.

Modified Slack Based (MSB) Algorithm The pseudo code for the modified slack based algorithm is shown below.

Checking the admissibility of job J with Latest Start Time into an existing profile of size N:

A. set cheapPrice to MAXNUMBER
B. set cheapSchedule to existing schedule
C. for each time slot t_s in the profile starting from current time
    a. Remove all the jobs from slot t_s to the end
    b. insert job J at slot t_s
    c. schedule each removed job one by one in Ascending Scheduled Time (AST) order
    d. Calculate the price of this new schedule using the cost function
    e. if (price <cheapPrice) then
        i. set cheapPrice = price
        ii. set cheapSchedule = new Schedule
        iii. Update the slack of all jobs
    end if
end for
D. if (price != MAXNUMBER) then
    Job is accepted

else
    Job is rejected
end if

Compared to the original SB algorithm, MSB differs in the way the slack is determined for a given job. The original SB algorithm uses weighted user and political priorities to determine the slack. However, in the current scenario, we change this by setting the slack to be as: Slack = Deadline - (Arrival Time + Run Time).

The rest of the algorithm follows the approach taken by the SB algorithm. When a new job arrives, the jobs currently present in the schedule are arranged in an order decided by a heuristic function (such as Earliest Deadline First, or Least Laxity First). Once this order is fixed, the new job is inserted in each possible position in this arrangement. Thus, if there are N jobs existing in the schedule, when the new job arrives, $N+1$ schedules are possible. A pre-decided cost function is used to evaluate the cost of each of these $N+1$ schedules and the one with the least cost is accepted. We can easily see that MSB is an $O(N)$ algorithm considering the evaluation of the cost function to be a constant cost. In practice, the cost of evaluating the cost function of the schedule depends on the number of jobs in the schedule and thus is a function of N. However, for the sake of comparison between the various algorithms and for ease of understanding, we consider the time to evaluate a cost function to be constant.

Modified Real Time (MRT) Algorithm The pseudo code for the modified real time algorithm is shown below.

Checking the admissibility of job J with deadline into an existing profile of size N:

A. Remove all jobs from existing profile and add them into a Temporary List (TL)
B. Add the new job J into Temporary List(TL)
C. Sort temporary list according to the Heuristic function
D. Create an empty schedule without any job
E. for each job J_i from Temporary List
    a. Find whether job J_i is strongly feasible in the current partial schedule
    b. if J_i is strongly feasible then
        i. Add job J_i into partial schedule
        ii. Remove job J_i from the temporary list and continue from step E
    else
        i. Backtrack to the previous partial schedule
        ii. if (no of backtracks > MAXBACKTRACK) then
            1. Job J is rejected

2. Keep the old schedule and break
else
1. continue step E with new partial schedule
end if
end if
end for
F. if (all jobs are placed in the schedule) then
a. Job J is accepted
b. Update the current schedule
end if

The RT algorithm assumes that the calculation of the heuristic function for scheduling a job into a given partial schedule takes constant time. However, this assumption only holds true for sequential (single processor) jobs (which was the focus of the algorithm). However, the scenario we are looking at in this paper relates to parallel jobs, where holes are possible in the partial schedule. In this scenario, such an assumption would not hold true.

The Modified RT algorithm (MRT algorithm) uses the same technique as the RT algorithm but increases the time complexity to accommodate the parallel job scenario. When a new job arrives, all the jobs that have not yet started (including the newly arrived job) are sorted using some heuristic function (this function could be Earliest Deadline First, Least Laxity First, etc). Each of these jobs is inserted into the schedule in the sorted order. A partial schedule at any point during this algorithm is said to be feasible if every job in it meets its deadline. A partial schedule is said to be strongly feasible if the following two conditions are met:

- The partial schedule is feasible
- The partial schedule would remain feasible when extended by any one of the unscheduled jobs

When the algorithm reaches a point where the partial schedule obtained is not feasible, it backtracks to a previous strongly feasible partial schedule and tries to take a different path. A certain number of backtracks are allowed, after which the scheduler rejects the job.

3 The QoPS Scheduler

In this section we present the QoPS algorithm for parallel job scheduling in deadline-based systems. As mentioned earlier, it has been shown that for dynamic systems with more that one processor, a polynomial-time optimal scheduling algorithm does not exist. The QoPS scheduling algorithm uses a heuristic approach to search for feasible schedules for the jobs.

The scheduler considers a system where each job arrives with a corresponding completion time deadline requirement. When each job arrives, it attempts to find

a feasible schedule for the newly arrived job. A schedule is said to be feasible if it does not violate the deadline constraint for any job in the schedule, including the newly arrived job. The pseudo code for the QoPS scheduling algorithm is presented below:

Checking the admissibility of job J into an existing profile of size N:

A. For each time slot t_s in position (0, N/2, 3N/4, 7N/8 ...) starting from Current Time

1. Remove all waiting jobs from position t_s to the end of profile and place them into a Temporary List (TL)
2. Sort the temporary list using the Heuristic function
3. Set Violation Count = 0
4. For each job J_i in the temporary List (TL)
   i. Add Job J_i into the schedule
   ii. if (there is a deadline violation for job J_i at slot T) then
      a. Violation Count = Violation Count + 1
      b. if (Violation Count > K-Factor) break
      c. Remove all jobs from the schedule from position mid(t_s + T) to position T and add them into TL
      d. Sort the temporary list using the Heuristic function
      e. Add the failed job J_i into the top of temporary list to make sure it will be scheduled at mid(t_s + T)
   end if
end for
5. if (Violation Count > K-Factor) then
   i. Job is rejected
   ii. break
end if

end for

B. if (violation count < K-Factor) then
   Job is accepted
end if

The main difference between the MSB and the QoPS algorithm is the flexibility the QoPS algorithm offers in reordering the jobs that have already been scheduled (but not yet started).

For example, suppose jobs J_1, J_2, ..., J_N are the jobs which are currently in the schedule but not yet started. The MSB algorithm specifies a fixed order for the jobs as calculated by some heuristic function (the heuristic function could be least laxity first, earliest deadline first, etc). This ordering of jobs specifies the order in which the jobs have to be considered for scheduling. For the rest of the algorithm, this ordering is fixed. When a new job J_{N+1} arrives, the MSB algorithm tries to fit this new job in the given schedule without any change to the initial ordering of the jobs.

On the other hand, the QoPS scheduler allows flexibility in the order in which jobs are considered for scheduling. The amount of flexibility offered is determined by the K-factor denoted in the pseudo code above.

When a new job arrives, it is given $log_2(N)$ points in time where its insertion into the schedule is attempted, corresponding to the reserved start-times of jobs {0, N/2, 3N/4, ... } respectively, where N is the number of jobs currently in the queue. The interpretation of these options is as follows: For option 1 (corresponding to 0 in the list), we start by removing all the jobs from the schedule and placing them in a temporary ordering (TL). We then append the new job to TL and sort the *N+1* jobs in TL according to some heuristic function (again, the heuristic function could be least laxity first, earliest deadline first, etc). Finally, we try to place the jobs in the order specified by the temporary ordering TL. For option 2, we do not start with an empty schedule. Instead, we only remove the latter N/2 jobs in the original schedule, chosen in scheduled start time order, place them in the temporary list TL. We again append the new job to TL and sort the *N/2 + 1* jobs in the temporary list TL (based on the heuristic function). We finally generate reservations for these *N/2 + 1* jobs in the order specified by TL. Thus, $log_2(N)$ options for placement are evaluated.

For each option given to the newly arrived job, the algorithm tries to schedule the jobs based on this temporary ordering. If a job misses its deadline, this job is considered as a critical job and is pushed to the head of the list (thus altering the temporary schedule). This altering of the temporary schedule is allowed at most 'K' times; after this the scheduler decides that the new job cannot be scheduled while maintaining the deadline for all of the already accepted jobs and rejects it. This results in a time complexity of *O(K* $log_2(N)$*)* for the QoPS scheduling algorithm.

4 Evaluation Approach

In this section we present the approach we took to evaluate the QoPS scheme with the other schemes and the non-deadline based EASY backfilling scheme.

4.1 Trace Generation

Job scheduling strategies are usually evaluated using real workload traces, such as those available at the Parallel Workload Archive [5]. However real job traces from supercomputer centers have no deadline information.

A possible approach to evaluating the QoPS scheduling strategy might be based on the methodology that was used in [15] to evaluate their real-time scheduling scheme. Their randomized synthetic job sets were created in such a way that a job set could be packed into a fully filled schedule, say from time = 0 to time = T, with no holes at all in the entire schedule. Each job was then given an arrival time of zero, and a completion deadline of (1+r)*T. The value of 'r' represented a degree of difficulty in meeting the deadlines. A larger value of 'r' made the deadlines more lax. The initial synthetic packed schedule is clearly

a valid schedule for all non-negative values of 'r'. The real-time scheduling algorithm was evaluated for different values of 'r', over a large number of such synthesized task sets. The primary metric was the fraction of cases that a valid schedule for all tasks was found by the scheduling algorithm. It was found that as 'r' was increased, a valid schedule was found for a larger fraction of experiments, asymptotically tending to 100% as 'r' increased.

We first attempted to extend this approach to the dynamic context. We used a synthetic packed schedule of jobs, but unlike the static context evaluated in [15], we set each job's arrival time to be its scheduled start time in the synthetic packed schedule, and set its deadline beyond its start-time by (1+r) times its runtime. When we evaluated different scheduling algorithms, we found that when 'r' was zero, all schemes had a 100% success rate, while the success rate dropped as 'r' was increased! This was initially puzzling, but the reason was quickly apparent; With r = 0, as each job arrives, the only possible valid placement of the new job corresponds to that in the synthetic packed schedule, and any deadline-based scheduling algorithm exactly tracks the optimal schedule. When 'r' is increased, other choices are feasible, and the schedules begin diverging from the optimal schedule, and the failure rate increases. Thus, this approach to generating the test workload is attractive in that it has a known valid schedule that meets the deadlines of all jobs; but it leads to the unnatural trend of decreasing scheduling success rate as the deadlines of jobs are made more relaxed.

Due to the above problem with the evaluation methodology used in [15], we pursued a different trace-driven approach to evaluation. We used traces from Feitelson's archive (5000-job subsets of the CTC and the SDSC traces) and first used EASY back-fill to generate a valid schedule for the jobs. Deadlines were then assigned to all jobs, based on their completion time on the schedule generated by EASY back-fill. A deadline stringency factor determined how tight the deadline was to be set, compared to the EASY back-fill schedule. With a stringency factor of 0, the deadlines were set to be the completion times of the jobs with the EASY back-fill schedule. With a stringency factor of 'S', the deadline of each job was set ahead of its arrival time by max(runtime, (1-S)*EASY-Schedule-Response-time). The metric used was the number of jobs successfully scheduled. As 'S' was increased, the deadlines became more stringent. So we would expect the number of successfully scheduled jobs to decrease.

4.2 Evaluation Scenarios

With the first set of simulation experiments, the three schemes (MRT, MSB and QoPS) were compared under different offered loads. A load factor "l" was varied from 1.0 to 1.6. The variation of the load was done using two schemes: Duplication and Expansion. For the duplication scheme, with l = 1.0, only the original jobs in the trace subset were used. With l = 1.2, 20% of the original jobs were picked and duplicates were introduced into the trace at the same arrival time to form a modified trace. For the expansion scheme, the arrival times of the jobs were unchanged, but the run-times were magnified by a factor of 1.2 for l =

1.2. The modified trace was first scheduled using EASY back-fill, and then the deadlines for jobs were set as described above, based on the stringency factor.

After evaluating the schemes under the scenario described above, we carried out another set of experiments under a model where only a fraction of the jobs have deadlines associated with them. This might be a more realistic scenario at supercomputer centers; while some of the jobs may be urgent and impose deadlines, there would likely also be other jobs that are non-urgent, with the users not requiring any deadlines. In order to evaluate the MSB, MRT, and QoPS schemes under this scenario of mixed jobs, some with user-imposed deadlines and others without, we artificially created very lax deadlines for the non-deadline jobs. While the three schemes could be run with an "infinite" deadline for the non-deadline jobs, we did not do that in order to avoid starvation of any jobs. The artificial deadline of each non-deadline job was set to max(24 hours, R*runtime), 'R' being a "relaxation" factor. Thus, short non-deadline jobs were given an artificial deadline of one day, while long jobs were given a deadline of R*runtime.

5 Experimental Results

As discussed in the previous section, the deadline-based scheduling schemes were evaluated through simulation using traces derived from the CTC and the SDSC traces archived at the Parallel Workloads Archive [5]. Deadlines were associated with each job in the trace as described earlier. Different offered loads were simulated by addition of a controlled number of duplicate jobs. The results for the expansion scheme are omitted here due to space reasons and can be found in [**?**]. The introduction of duplicate jobs is done incrementally, i.e. the workload for a load of 1.6 would include all the jobs in the trace for load 1.4, with the same arrival times for the common jobs. For a given load, different experiments were carried out for different values of the stringency factor 'S', with jobs having more stringent deadlines for a higher stringency factor. In this paper, we show only the results for the CTC trace and refer the reader to [7] for the results with the SDSC trace.

The values of the K-factor and the Relaxation Factor 'R' used for the QoPS scheme are 5 and 2 respectively, and the number of backtracks allowed for the MRT scheme was 300 for all the experiments.

5.1 All Jobs with Deadlines

We first present results for the scenario where all the submitted jobs had deadlines associated with them, determined as described in the previous section. The metrics measured were the percentage of unadmitted jobs and the percentage of lost processor-seconds from the unadmitted jobs.

Figure 1 shows the percentage of unadmitted jobs and lost processor-seconds for the MRT, MSB and QoPS schedules, for a stringency factor of 0.2, as the load factor is varied from 1.0 to 1.6. It can be seen that the QoPS scheme performs better, especially at high load factors.

In the case of QoPS, as the load is increased from 1.4 to 1.6, the total number of unaccepted jobs actually decreases, even though the total number of jobs in the trace increases from 7000 to 8000. The reason for this counter-intuitive result is as follows. As the load increases, the average wait time for jobs under EASY backfill increases nonlinearly as we approach system saturation. Since the deadline associated with a job is based on its schedule with EASY backfill, the same job will have a higher response time and hence looser deadline in a higher-load trace than in a trace with lower load. So it is possible for more jobs to be admitted with a higher-load trace than with a lower-load trace, if there is sufficient increase in the deadline. A similar and more pronounced downward trend with increasing load is observed for the unadmitted processor-seconds. This is due to the greater relative increase in response time of "heavier" jobs (i.e. those with higher processor-seconds) than lighter jobs. As the load increases, more heavy jobs are admitted and more light jobs are unable to be admitted.

The same overall trend also holds for a higher stringency factor (0.5), as seen in Figure 2. However, the performance of QoPS is closer to the other two schemes. In general, we find that as the stringency factor increases, the performance of the different strategies tends to converge. This suggests that the additional flexibility that QoPS tries to exploit in rearranging schedules is most beneficial when the jobs have sufficient laxity with respect to their deadlines.

We next look at the achieved utilization of the system, as the load is varied. As a reference, we compare the utilization for the deadline-based scheduling schemes with non-deadline based job scheduling schemes (EASY and Conservative backfilling) using the same trace. Since a fraction of submitted jobs are unable to be admitted in the deadline-based schedule, clearly we can expect the achieved system utilization to be worse than the non-deadline case. Figure 3 shows the system utilization achieved for stringency factors of 0.2 and 0.5. There is a loss of utilization of about 10% for QoPS when compared to EASY and Conservative, when the stringency factor is 0.2. With a stringency factor of 0.5, fewer jobs are admitted, and the utilization achieved with the deadline-based scheduling schemes drops by 5-10%. Among the deadline-based schemes, QoPS and MSB perform comparably, with MRT achieving 3-5% lower utilization at high load (load factor of 1.6).

5.2 Mixed Job Scenario

We next consider the case when only a subset of submitted jobs have user-specified deadlines. As discussed in Section 4, non-deadline jobs were associated with an artificial deadline that provided considerable slack, but prevented starvation. We evaluated the following combinations through simulation: a) 80% non-deadline jobs and 20% deadline jobs, and b) 20% non-deadline jobs and 80% deadline jobs. For each combination, we considered stringency factors of 0.20 and 0.50.

Figure 4 shows the variation of the admittance of deadline jobs with offered load for the schemes, when 80% of the jobs are deadline jobs, and stringency

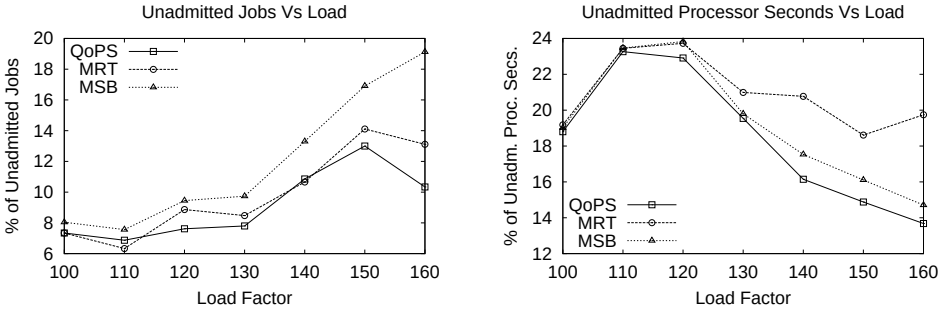

Fig. 1. Admittance capacity for less stringent (Stringency Factor = 0.2) deadlines: (a) Unadmitted jobs, (b) Unadmitted Processor Seconds

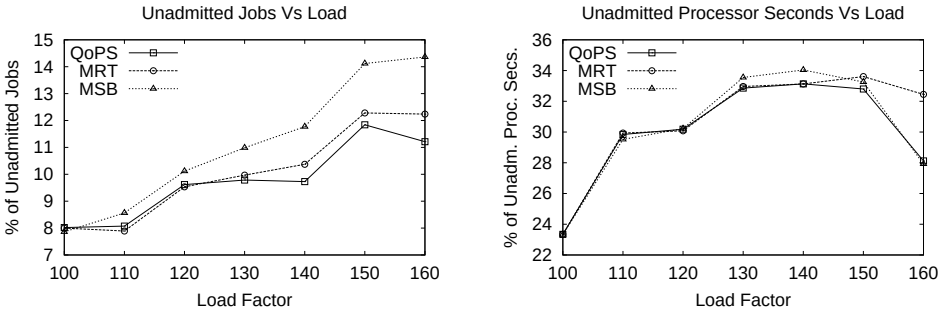

Fig. 2. Admittance capacity for more stringent (Stringency Factor = 0.5) deadlines: (a) Unadmitted jobs, (b) Unadmitted Processor Seconds

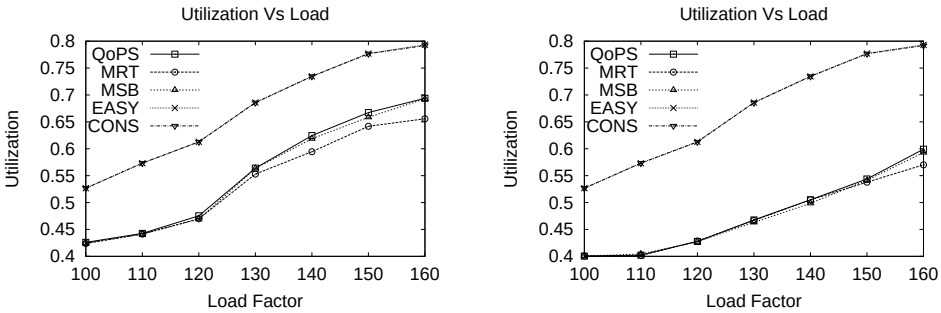

Fig. 3. Utilization comparison: (a) Stringency Factor = 0.2, (b) Stringency Factor = 0.5

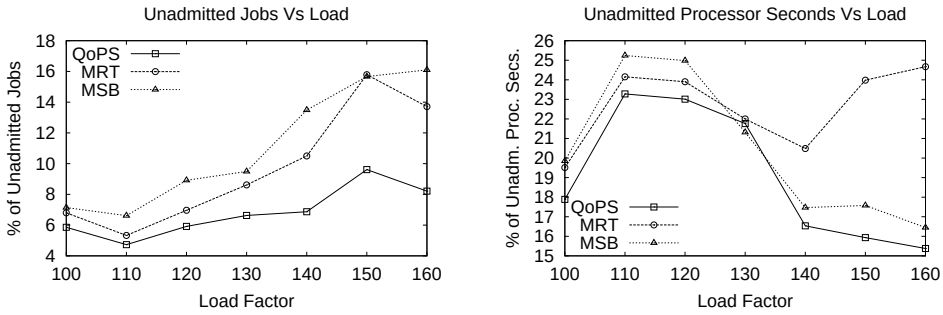

Fig. 4. Admittance capacity with a mix of deadline and non-deadline jobs (Percentage of Deadline Jobs = 80%) for less stringent (Stringency Factor = 0.2) deadlines: (a) Unadmitted jobs, (b) Unadmitted Processor Seconds

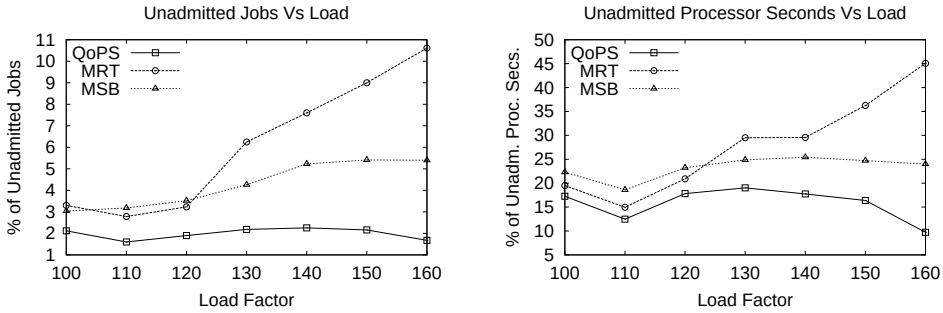

Fig. 5. Admittance capacity with a mix of deadline and non-deadline jobs (Percentage of Deadline Jobs = 20%) for less stringent (Stringency Factor = 0.2) deadlines: (a) Unadmitted jobs, (b) Unadmitted Processor Seconds

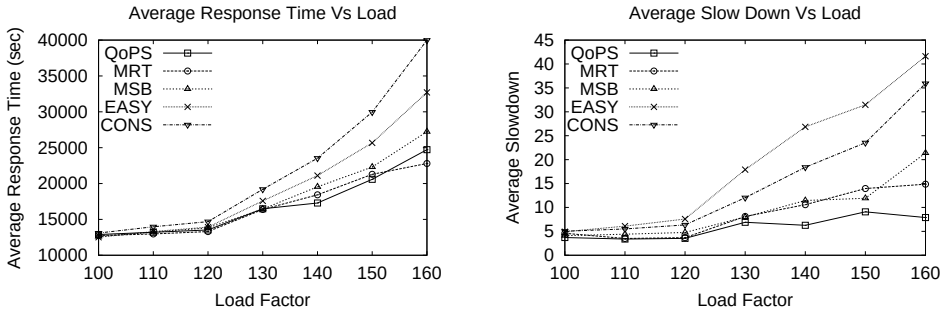

Fig. 6. Performance of Non-deadline jobs with Percentage of Deadline Jobs = 20%; Stringency Factor = 0.2 (a) Response time, (b) Average slowdown

factor is 0.2. It can be seen that the QoPS scheme provides consistently superior performance compared to the MSB and MRT schemes, especially at high load.

Figure 5 presents data for cases with 20% of jobs having user-specified deadlines, and a stringency factor of 0.2. Compared to the cases with 80% of jobs being deadline-jobs, the QoPS scheme significantly outperforms the MSB and MRT schemes. This again suggests that in scenarios where many jobs have significant flexibility (here the non-deadline jobs comprise 80% of jobs and they have significant flexibility in scheduling), the QoPS scheme makes effective use of the available flexibility. The results for a stringency factor of 0.5 are omitted and can be found in [7].

Figures 6 and 7 show the variation of the average response time and average slowdown of non-deadline jobs with load, for the cases with 20% and 80% of the jobs being deadline jobs and a stringency factor of 0.2. In addition to the data for the three deadline-based scheduling schemes, data for the EASY and Conservative back-fill mechanisms is also shown. The average response time and slowdown can be seen to be lower for QoPS, MRT and MSB, when compared to EASY or Conservative. This is because the delivered load for the non-deadline based schemes is equal to the offered load (the X-axis), whereas the delivered load for the deadline-based scheduling schemes is lower than offered load. In other words, with the non-deadline based schemes, all the jobs are admitted, whereas with the other deadline based schemes, not all deadline jobs are admitted. This also explains the reason why the performance of QoPS appears inferior to MSB and MRT - as seen from Figure 4, the rejected load from the deadline jobs is much higher for MRT than QoPS.

When the data for the case of 20% deadline jobs is considered (Figure 6), the performance of QoPS has improved relative to MSB; the turnaround time is comparable or better except for a load of 1.6, and the average slowdown is lower at all loads. These user metrics are better for QoPS than MSB/MRT despite accepting a higher load (Figure 5).

The achieved utilization for the different schemes as a function of load is shown in Figure 8 for a stringency factor of 0.2. It can be seen that the achieved utilization with QoPS is roughly comparable with MSB and better than MRT, but worse than the non-deadline based schemes. Compared to the case when all jobs had user-specified deadlines (Figure 3), the loss of utilization compared to the non-deadline based schemes is much less: about 8% when 80% of the jobs are deadline jobs, and 2-3% when 20% of the jobs are deadline jobs.

As discussed above, a direct comparison of turnaround time and slowdown as a function of offered load is complicated by the fact that different scheduling schemes accept different numbers of jobs. A better way of comparing the schemes directly is by plotting average response time or slowdown against achieved utilization on the X-axis (instead of offered load). This is shown in Figure 9, for the case of 20% deadline jobs and a stringency factor of 0.2. When there is sufficient laxity available, it can be seen that QoPS is consistently superior to MSB and MRT. Further, QoPS has better performance than EASY and conservative backfilling too, especially for the slowdown metric. Thus, despite the constraints

of the deadline-jobs, QoPS is able to achieve better slowdown and response time for the non-deadline jobs when compared to EASY and conservative backfilling, i.e. instead of adversely affecting the non-deadline jobs for the same delivered load, QoPS provides better performance for them when compared to the standard back-fill mechanisms. However, when the number of deadline jobs increases to 80% (Figure 10), the performance of QoPS degrades.

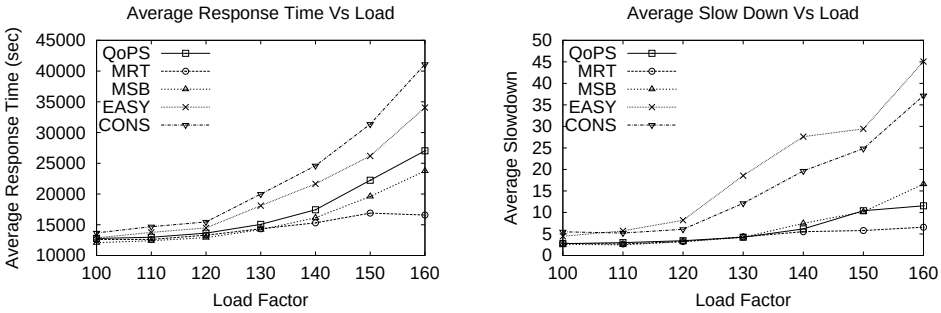

Fig. 7. Performance of Non-deadline jobs with Percentage of Deadline Jobs = 80%; Stringency Factor = 0.2 (a) Response time (b) Average Slowdown

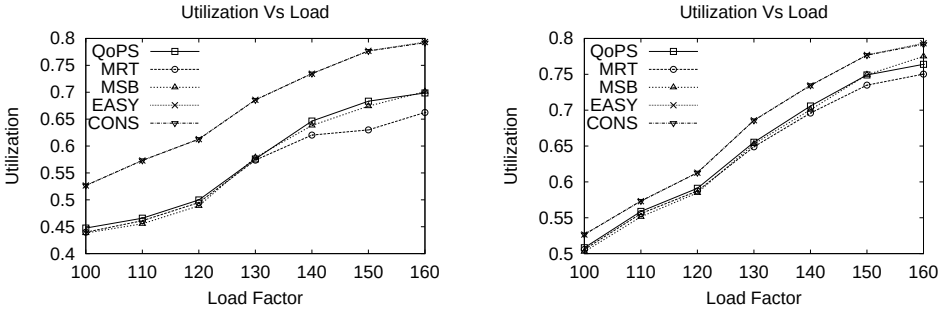

Fig. 8. Utilization comparison for a mix of deadline and non-deadline jobs (Stringency Factor = 0.2): (a) Percentage of Deadline Jobs = 80%, (b) Percentage of Deadline Jobs = 20%

6 Conclusions and Future Work

Scheduling dynamically-arriving independent parallel jobs on a given set of resources is a long studied problem; issues addressed include from evaluation of system and user metrics such as utilization, throughput, turnaround time, etc. to

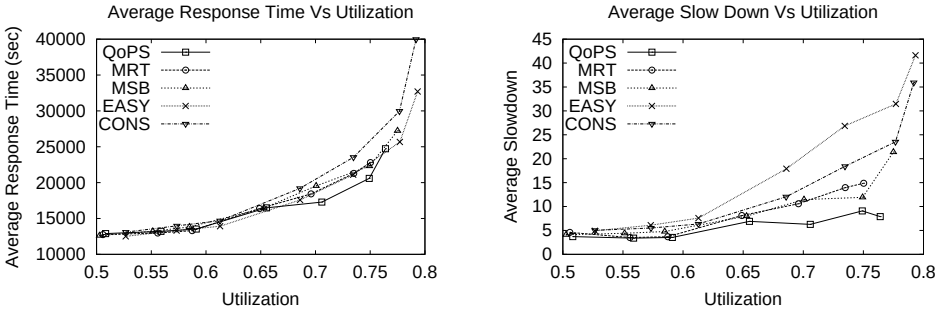

Fig. 9. Performance variation of Non-deadline jobs with utilization for Percentage of Deadline Jobs = 20%; Stringency Factor = 0.2 (a) Average response time (b) Average slowdown

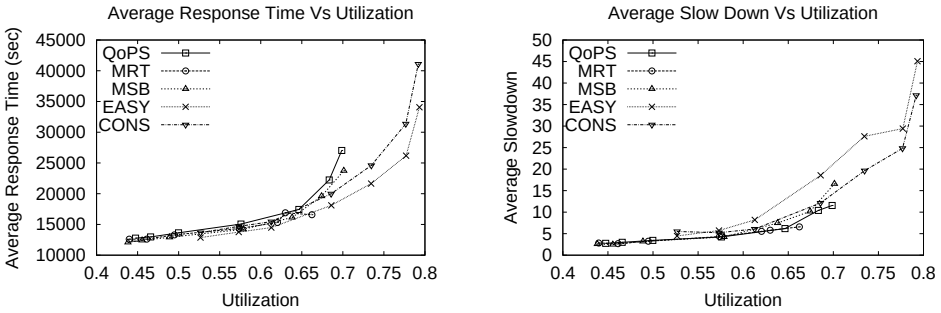

Fig. 10. Performance variation of Non-deadline jobs with utilization for Percentage of Deadline Jobs = 80%; Stringency Factor = 0.2 (a) Average response time, (b) Average slowdown

soft time-guarantees for the response time of jobs using priority based scheduling. However, a solution to the problem of providing Quality of Service (QoS) for Parallel Job Scheduling has not been previously addressed. In this paper, we proposed a new scheme termed as the *QoPS Scheduling Algorithm* to provide QoS in the response time given to the end user in the form of guarantees in the completion time to the submitted independent parallel jobs. Its effectiveness has been evaluated using trace-driven simulation.

The current scheme does not explicitly deal with cost-metrics for charging the jobs depending on the deadlines and resource usage. Also, when a job arrives, it has a number of options for placement in the schedule. The current scheme looks at each of these options in an FCFS order and does not do any kind of evaluation to see if one option is better (for the system and user metrics, such as utilization for example) than the others. We plan to extend this to define cost functions for both charging the jobs and for evaluating and choosing between several feasible schedules when admitting new jobs.

References

1. NERSC. http://hpcf.nersc.gov/accounts/priority_charging.html.
2. S.-H. Chiang and M. K. Vernon. Production Job Scheduling for Parallel Shared Memory Systems. In *the Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, April 2001.
3. W. Cirne and F. Berman. Adaptive Selection of Partition Size of Supercomputer Requests. In *the Proceedings of 6th workshop on Job Scheduling Strategies for Parallel Processing*, April 2000.
4. D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *the Proceedings of IEEE Workshop on Job Scheduling Strategies for Parallel Processing*, 1997.
5. D. G. Feitelson. Logs of Real Parallel Workloads from Production Systems.
6. P. Holenarsipur, V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan. Characterization and Enhancement of Static Mapping Heuristics for Heterogeneous Systems. In *the Proceedings of the IEEE International Symposium on High Performance Computing (HiPC)*, December 2000.
7. M. Islam, P. Balaji, P. Sadayappan, and D. K. Panda. QoPS: A QoS based scheme for Parallel Job Scheduling. Technical report, The Ohio State University, Columbus, OH, April 2003.
8. B. Jackson, B. Haymore, J. Facelli, and Q. O. Snell. Improving Cluster Utilization Through Set Based Allocation Policies. In *IEEE Workshop on Scheduling and Resource Management for Cluster Computi ng*, September 2001.
9. P. Keleher, D. Zotkin, and D. Perkovic. Attacking the Bottlenecks in Backfilling Schedulers. In *Cluster Computing: The Journal of Networks, Software Tools and Applications*, March 2000.
10. A. K. Mok. *Fundamental design problems of distributed systems for the hard real-time environment.* PhD thesis, Massachussetts Institute of Technology, Cambridge, MA, May 1983.
11. A. K. Mok. The design of real-time programming systems based on process models. In *the Proceedings of IEEE Real Time Systems Symposium*, December 1984.

12. A. K. Mok and M. L. Dertouzos. Multi-Processor Scheduling in a Hard Real-Time Environment. In *the Proceedings of the Seventh Texas Conference on Computing Systems*, November 1978.
13. A. W. Mualem and D. G. Feitelson. Utilization, Predictability, Workloads and User Estimated Runtime Esti mates in Scheduling the IBM SP2 with Backfilling. In *in IEEE Transactions on Parallel and Distributed Systems*, volume 12, June 2001.
14. D. Perkovic and P. Keleher. Randomization, Speculation and Adaptation in Batch Schedulers. In *the Proceedings of the IEEE International Conference on Supercomputing*, November 2000.
15. K. Ramamritham, J. A. Stankovic, and P.-F. Shiah. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume I, April 1990.
16. D. Talby and D. G. Feitelson. Supporting Priorities and Improving Utilization of the IBM SP2 scheduler using Slack Based Backfilling. In *the Proceedings of the 13th Intl. Parallel Processing Symposium*, pages 513–517, April 1997.