The Effect of Correlating Quantum Allocation and Job Size for Gang Scheduling *

Gaurav Ghare Scott T. Leutenegger
Mathematics and Computer Science Department
University of Denver Denver, CO 80208-0189
{*gghare,leut*}*@cs.du.edu*

Abstract

Gang scheduling is an effective scheduling policy for multiprocessing workloads with significant interprocess synchronization and is in common use in real installations. In this paper we show that significant improvement in the job slowdown metric can be achieved simply by allocating a different number of quanta to different rows (control groups) depending on the number of processes belonging to jobs in a given row. Specifically, we show that allocating the number of quanta inversely proportionally to the number of processes per job in that row results in 20 - 50% smaller slowdowns without significantly affecting mean job response time. Incorporating these suggestions into real schedulers would require the addition of only a few lines of simple code, hence this work should have an immediate practical impact.

1 Introduction

Ousterhout proposed the notion of co-scheduling to efficiently support interprocess synchronization when multiprogramming a set of parallel jobs on a multiprocessor machine [22]. Coscheduling strives to ensure that all processes belonging to a job are scheduled at the same time. Subsequent work has generalized and refined the coscheduling (now often called gang scheduling) concept [5, 6, 8, 9, 11, 14, 15, 19, 23, 24, 25, 26]. Gang scheduling schemes are a practical result of the multiprocessor scheduling community and have been adapted for inclusion in several production systems including the Intel Paragon [4], CM-5 [3], Meiko CS-2, multiprocessor SGI workstations [1] and the IBM SP2 [17, 18].

Work has been done to examine the impact of quantum size allocations on mean job response times and slowdows for different quantum sizes [25, 26]. Little work has been done on investigating how quantum allocation can be modified to improve mean job slowdowns and response times. In this paper we show that negatively correlating the number of allocated quanta with the number of processes per job can significantly reduce mean job slowdown. Specifically, we show that allocating additional quanta to rows (or control groups) containing jobs with a small number of processes results in the best overall performance observed in our experiments.

*This work is supported by the NSF grant ACI-9733658.

The past work most similar to our work is that of Squillante et al, and Wang et al. [25, 26, 27]. In [25] the modeling methodology allows for gangs of different classes to be allocated different quantum sizes, but results presented do not investigate this issue, instead the the work assumes all classes have the same quantum size.

In [27] the authors set quantum sizes per class based on different offered work per class. We do not consider this aspect in our studies. Note, the authors state that it would be interesting to investigate the effect of allocating multiple quanta to short running jobs. We provide experimental results for such considerations and generalize them to include simpler gang scheduling algorithms.

In [26] the authors consider dividing processing power unequally among classes, via different size quantums, stating possible motivations of allocating larger quantum to classes with higher context switch costs and classes demanding shorter response time. In figure 8 the authors show how the mean number of class i jobs in the system decreases as the fraction of processing power (quantum length relative to other class quantum lengths) allocated to class i jobs is increased. The experiment does not show what happens to the overall number of jobs in the system as quantum lengths are increased for one particular class.

Our work differs in that we fully explore the effect of varying quantum allocation per control group size (for DHC). We consider how favoring one class affects all classes by reporting mean slowdowns and response times. We provide numerous simulation results to make a case for proper quantum allocation.

In addition to considering the DHC algorithm, we consider the effect of different quantum allocations with simpler gang scheduling schemes such as Matrix [22] and LRS [5]. These simpler algorithms are especially relevant since many current production level schedulers use variants of these simpler algorithms. Finally, our work also provides a more exhaustive comparison of DHC with Matrix and LRS by considering more workloads than considered in [5] and comparing DHC with optimized versions of LRS and Matrix.

Aside from extending the understanding of gang scheduling, this work is directly applicable to existing systems. The work involved to modify existing schedulers to incorporate these findings should be trivial. Thus, for almost no implementation work commercial systems with correlated workloads similar to our synthetic workloads can decrease slowdowns by 10 to 50% . Furthermore, for some workloads we show that mean job response time can be decreased by 50% relative to equal allocation per row (or control group).

2 Scheduling Algorithm Description

In this section we review the three previously proposed policies considered and detail how we allocate quantum sizes.

2.1 Matrix

Basic Algorithm Description

This is the scheduling algorithm as defined in the seminal paper by Ousterhout [22]. A newly arriving job is placed in the first row with a sufficient number of idle processor slots to accommodate the job. Note, the slots need not be contiguous within the row. Upon job completion row unification is not performed as it was shown to have little effect [22] since alternate selection fills in holes when scheduling a row.

The policy runs jobs by rotating through the rows of the matrix in a round-robin fashion. If there are idle slots within a row scheduled for execution, other rows are scanned (in round robin fashion from that row) to see if any other job(s) can be completely co-scheduled in the hole. We do not allow fragments to be run. In most of the algorithm variants below, a row is allocated multiple quanta. Alternate selection is performed as soon as a job completes so that processors are not left idle during the remaining quanta allocated to the slot. Note however that alternate selection is performed only between quanta.

Quantum Allocation

We consider a family of Matrix algorithms which differ in how many quanta are allocated to each row. Let J_i be the number of jobs in row i. Let Q_i be the number of quanta allocated to row i. Let S_t be the threshold size, dividing small jobs from large jobs. Let P be the number of processors. Note for our experiments, S_t was set to 8, and P was set to 128, but S_t should be larger if P were increased. We define the following policies:

Matrix-EQL: $Q_i = 1 \ \forall i$.

Matrix-S: $Q_i = J_i$.

Matrix-Sj (where j is an integer):

$$Q_i = \begin{cases} j, & \textit{if all jobs in row have} \leq S_t \textit{ processes} \\ 1, & \textit{otherwise} \end{cases} \tag{1}$$

Matrix-Lj (where j is an integer):

$$Q_i = \begin{cases} 1, & \textit{if all jobs in row have} \leq S_t \textit{ processes} \\ j, & \textit{otherwise} \end{cases} \tag{2}$$

Matrix-S and Matrix-Sj favor jobs with a small number of processes. In Matrix-S, the number of quanta allocated equal the number of jobs in the row, hence rows containing many small jobs will receive additional processing power. In Matrix-Sj, rows in which all jobs have 8 or fewer processes are allocated j quanta, a row containing one or more jobs of 9 or more processes is allocated one quantum. The parameter j allows us to vary how much preference is given to jobs with a small number of processes. In our experiments we consider Matrix-S2, Matrix-S8, and Matrix-S16. Note, the value of 8 for differentiating between large and small jobs was arbitrarily chosen based on intuition for the workload descriptions.

The Matrix-Lj policies are defined in a similar fashion but give preference to jobs with a large number of processes. Rows with at least one large job are given additional quanta.

2.2 LRS

LRS [5] differs from Matrix only in how newly arriving jobs are allocated within the matrix. Jobs that have more than 8 processes are allocated processors from left to right while the rest are allocated processors from right to left.

Quantum Allocation

The family of polices, LRS-EQL, LRS-S, LRS-Sj, and LRS-Lj, are defined in the same way as the family of Matrix algorithms above.

2.3 DHC

This algorithm is similar to the minimum fragmentation with selective disabling version of the DHC algorithm proposed by Feitelson and Rudolph [9]. It is based on a hierarchy of controllers. Each block of 2^i PEs is assigned a controller that coordinates their activities. Controllers at higher levels of the hierarchy coordinate those at the lower levels. In addition, the controllers at each level have lateral connections among them. An arriving job is mapped to a controller that has just enough processors under it to satisfy the job demand. The job is mapped to the controller that has the least load among the controllers at that level. If the controller controls more than one processor, it partitions the threads of the job as follows: Let $2^{i-1} < t <= 2^i$ be the total number of threads of the job, l_1 the load on the left child and l_2 the load on the right child. Assuming w.l.o.g. that $l_1 < l_2$, the first 2^{i-1} threads are mapped to processors controlled by the left child and the rest are mapped to processors controlled by the right child. The scheduling is done in rounds. In each round, jobs mapped to controllers at level i are scheduled ahead of those mapped to controllers at lower levels. Alternate selection is done as discussed for the previous algorithms.

Quantum Allocation

We consider a family of DHC algorithms which differ in how many quanta are allocated to each control group. Consider control group i, such that 2^i is the largest number of processes a job belonging to control group i may have. Jobs in each control group are allocated Q_i quanta. Furthermore, let $total$ equal the total number of processes in the system and let n_{im} denote the number of processes of job m in the i^{th} control group. We define the following policies:

DHC-EQL: $Q_i = 1 \; \forall i$.

DHC-Sj (for integer j):

$$Q_i = \begin{cases} (k-i+1)j, & \forall i, j=1 \\ \\ max(1,(k-i)j), & \forall i, j>1 \end{cases} \tag{3}$$

where k such that $2^k =$ the number of processors.

DHC-Lj (for integer j):

$$Q_i = \begin{cases} i+1, & \forall i, j=1 \\ \\ max(1,\; i \cdot j), & \forall i, j>1 \end{cases} \tag{4}$$

DHC-original: $Q_i = \Sigma_m \frac{n_{im}}{total}$.

Note that this is equivalent to the original definition: $Q_i = \frac{t_m}{total}$, where t_m is the number of processes of job m.

Thus, DHC-Sj allocates control groups with small jobs a larger number of quanta and the magnitude of the difference is determined by parameter j. Conversely, DHC-Lj allocates control groups with large jobs a larger number of quanta. DHC-original is the original quantum size allocation algorithm specified [9]. Feitelson and Rudolph state that preference should be given to jobs with a large number of processes. Note, this is contrary to the findings of this paper. Regardless, we find that the DHC-original policy approximates DHC-EQL at higher loads.

3 Simulation Methodology, Workload Models, and Metric

We have simulated a 128 processor machine using the Simpack [12, 13] simulation package. Our simulator models job arrivals with an exponential mean inter-arrival time. Upon job arrival the number of processes and total job demand are determined as described below. We do not simulate interprocess synchronization or context switch overheads. All three policies considered have been shown to support interprocess synchronization well and hence such details would distract from the emphasis of this paper. Tuning gang scheduling algorithms for context switch overheads has been considered in [26] and inclusion here would obscure the main focus of the paper.

Confidence intervals were collected using batch means. We ran 60 batches of 500 jobs per batch resulting in at most 10% response time confidence intervals at a 95% confidence level. The first 500 jobs simulated were discarded to achieve a warm start.

3.1 Definitions and Metrics

We define a *job* to be composed of one or more *processes.* We will refer to jobs with a small number of processes as "small" jobs vs "large" jobs, and jobs with short execution times as "short" jobs vs "long" jobs. Job response time is defined as the difference between job completion and arrival times. The *slowdown* of a job is defined as the ratio of the response time of a job over the the execution time of the job if run in isolation.

We collected both mean job response times and mean slowdowns. We focus primarily on mean slowdowns. The mean job response time metric is dominated by the long jobs thus obscuring the impact of the scheduling algorithm on short jobs. Conversely, the slowdown metric emphasizes the penalty paid for slowing down short jobs. Since users often are actively waiting for short jobs to complete it makes sense to focus on the slowdown metric as the primary comparison metric. We note that there is not agreement yet on the most relevant comparison metric [7], hence we consider both slowdown and response time.

3.2 Workload Models

We considered 4 different workloads. The first three are minor variants of the workloads proposed by Leutenegger and Vernon [20] and subsequently used in other works [2, 21]. The workloads differ in the degree of correlation between the number of processes and total job demand. The fourth workload is that of Feitelson [5].

Geometric-Bounded N, $N^{1.5}$ and N^2 Workloads

In these workloads we first determine the number of processes, n, for a job and then the job demand. As hypothesized in [20], and seen in many measured workloads on recent systems [10, 16], a significant portion of jobs have parallelism equal to the number of processors. In addition, a significant portion of jobs have parallelism equal to half the number of processors (assuming the number of processors is a power of two) Accordingly, we assume the number of processors is a power of 2 and let X be the percentage of jobs whose parallelism is equal to 2^l and 2^{l-1}, where 2^l is equal to the number of processors. For example, for $X = 10$, 10% of the jobs have the number of processes set to 128 and 10% of the jobs have the number of processes set to 64. The remaining $100-(2*X)$ percent of the jobs have the number of processes drawn from a geometric distribution with mean $\overline{n}$. In all of our studies we set $\overline{n} = 4$. If the sample drawn from the distribution exceeds the number of processors we truncate the sample to equal the number of processors. We consider values of X = 5%, 10%, and 25%. Note, recent measurement studies have also indicated more probability mass at 16 and 32 processes. Rather than try to mimic a specific distribution we choose to capture the spirit of the distribution. To address this concern we also study the workload of Feitelson, described below, which includes more of these smaller jobs.

In Figure 1 we plot the probability mass function of the number of processes per job for each of the workloads considered in this paper. Note that when X is increased the spikes at 64 and 128 processes

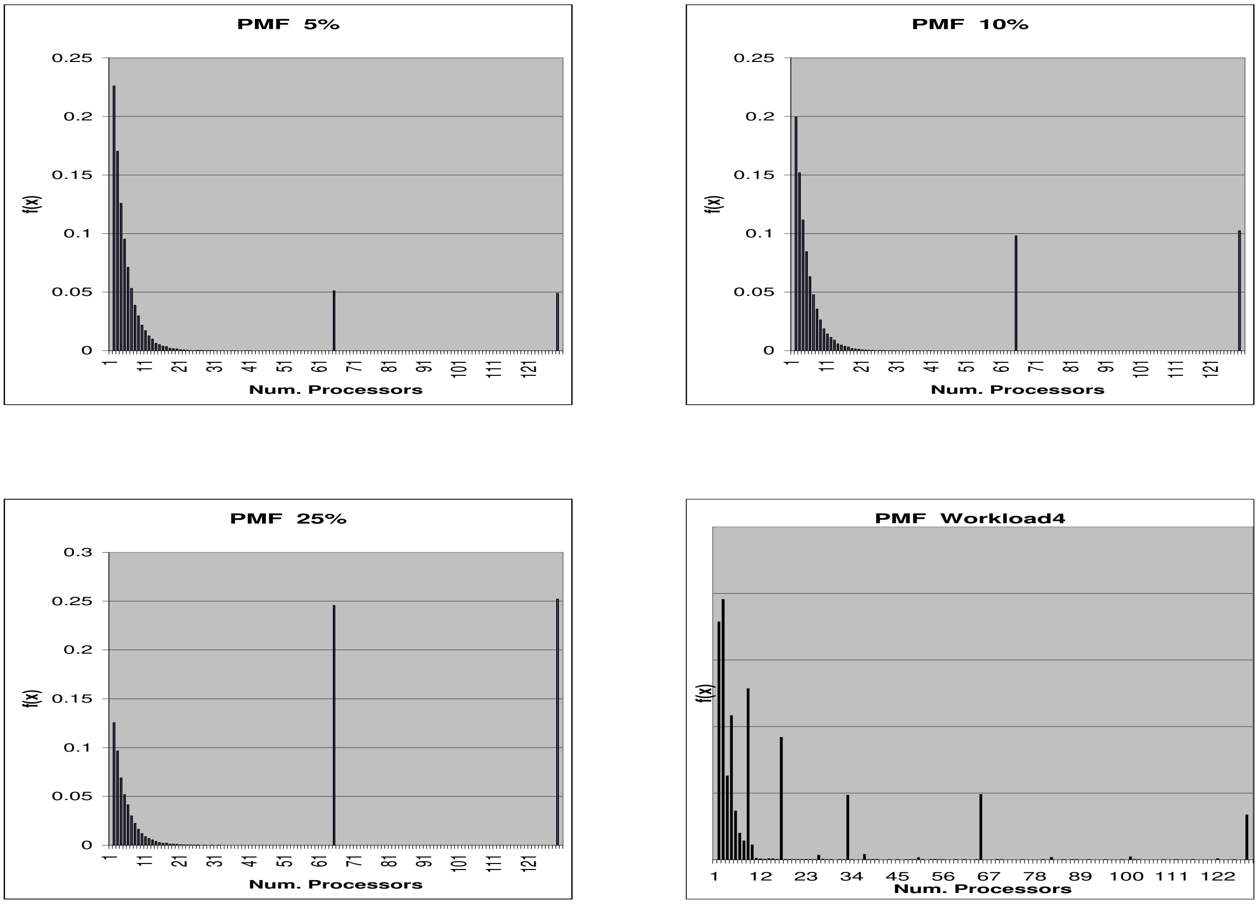

Figure 1: PMF Of Workloads: Top Left: $X = 0.05$, Top Right: $X = 0.1$, Bottom Left: $X = 0.25$, Bottom Right: Feitelson Workload

per job increase accordingly.

Once the number of processes, n, is obtained, the job demand is determined. In all cases we study we assume job demand is positively correlated with the number of processes. The job demand is drawn from a two-stage hyper-exponential distribution with mean D, where D is determined as described below. The coefficient of variation of the job demand is set equal to 2 in all of our reported experiments. The overall coefficient of variation is much greater than 2, due to the linear/superlinear dependence of job demand on processors [20]. For example, the coefficient of variation of total job demand for the N^2 workload is (8.0, 5.7, 3.6) for the workloads with X equal to (0.05, 0.1, 0.25).

We consider three cases:

$$\begin{array}{ll} N & D = n * d \\ N^{1.5} & D = n^{1.5} * d \\ N^2 & D = n^2 * d \end{array}$$

where d is a constant and n is the number of processes. Parameter d is set to 10 in all of our experiments.

Feitelson Workload

This is the workload proposed in [5]. The workload is based on observations from 6 production parallel machines. It contains a significant portion of jobs that have a parallelism that is a power of two. The correlation between number of processes and total job demand is between N and N^2. The workload also includes repeated executions of certain jobs. Arrivals are distributed according to a Poisson process. A more detailed description and the code for generating the workload is obtainable from `www.cs.huji.ac.il/labs/parallel/workload/wlmodels.html`.

4 Results

In this section we present our simulation results. In all experiments presented, bars for specific algorithms presented left to right correspond to entries in the legend from top to bottom.

4.1 Geometric-Bounded N^2 Workload

In Figure 2 we plot the slowdown of the variants of the LRS, DHC and Matrix algorithms for the N^2 workload with X, the percentage of jobs whose number of processes is set to 64 and 128, equal to 10. In each figure the left set of bars is for a utilization of 90% and the right set of bars is for a utilization of 70%. In each case we can observe that the variants that give preference to small jobs perform better than those that give preference to large jobs. But, giving too many quanta to small jobs is counter productive as exhibited by the alg-S8 and alg-S16 (where alg = LRS, Matrix, or DHC) policies. In general, variations that give extreme preference to large/small jobs perform poorly. In addition, equal allocation per row (or control group for DHC) results in larger slowdowns than alg-S. As shown in Figures 3, 4 and 5 the slowdowns of LRS-EQL, DHC-EQL, and Matrix-EQL are 36%, 14% and 39% higher than

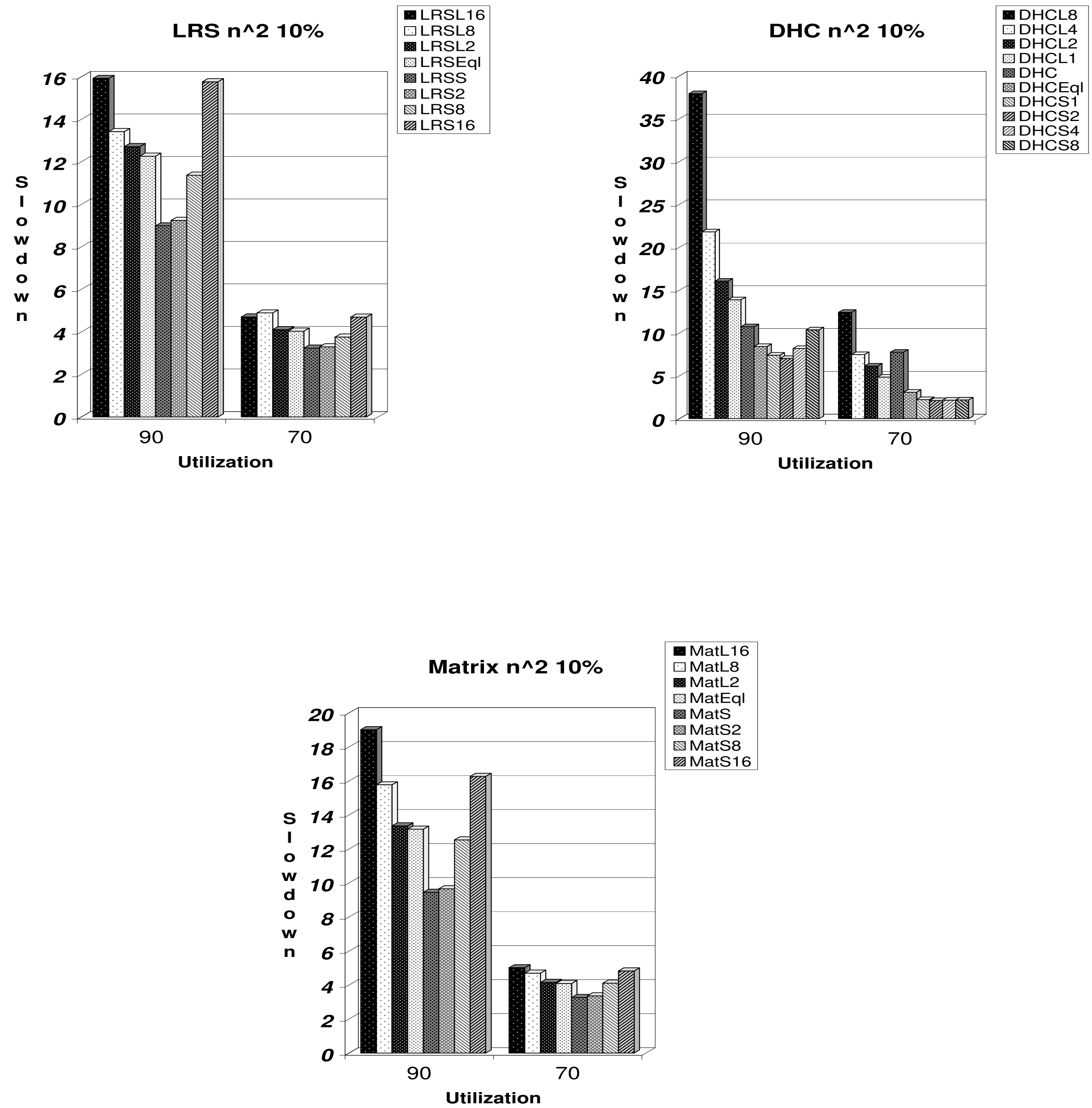

Figure 2: Slowdowns, N^2 $X = 10$ Workload: Top Left: LRS, Top Right: DHC, Bottom Left: Matrix

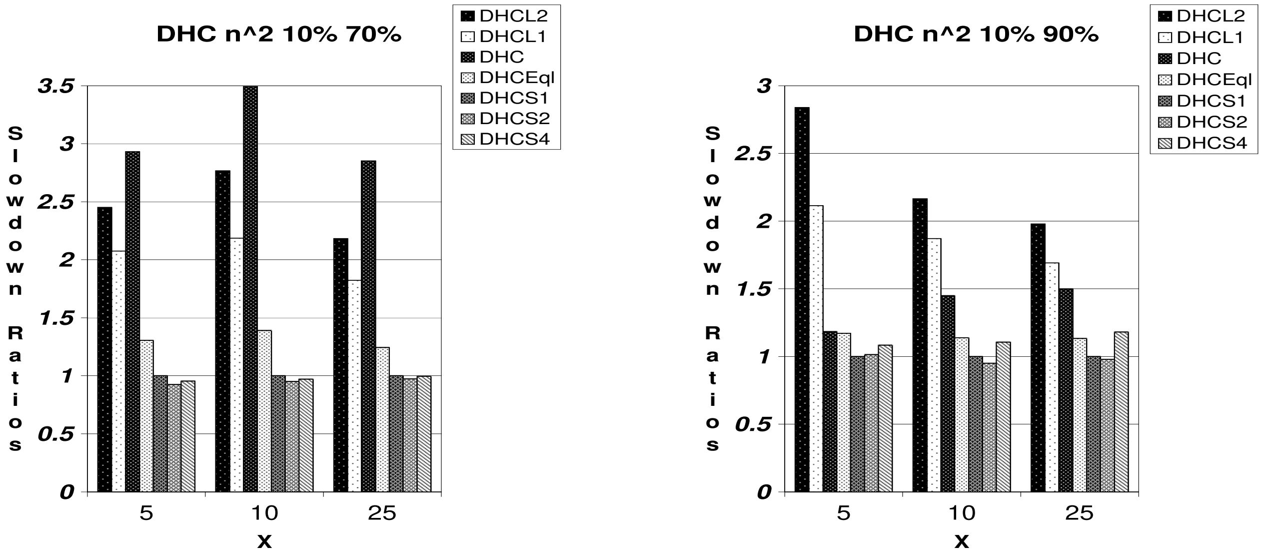

Figure 3: DHC Slowdown Ratios, N^2 X = 10 Workload: Left: 70% Utilization, Right: 90% Utilization

the slowdowns of LRS-S, DHC-S1, and Matrix-S at a 90% utilization and 25%, 39% and 25% higher at a 70% utilization.

To understand why alg-S performs better than alg-EQL, consider the case when multiple small jobs are packed per row (or control groups of the same level) whereas only one or two large jobs are packed per row. If each row is given equal allocation of processing power then the jobs with a large number of processes receive a disproportionately larger share of processing power relative to small jobs. As shown in [20], this is counter productive for workloads where the job demand is correlated with the number of processes. In [20] it was suggested that allocating equal processing power per job is beneficial in the absence of more detailed job knowledge. The LRS-S, Matrix-S, and DHC-S1 algorithms attempt to allocate equal processing power per job but are not able to given the rigid packing constraints of the algorithms. For example, if in Matrix there are two rows, one with a job of 128 processes and one row with 4 jobs of 32 processes each, then the first row is allocated 1 quantum while the second is allocated 4 quanta. In this case, each job is allocated equal processing power. On the other hand, if the second row has three jobs with 64, 32, and 32 processes each, the second row is allocated 3 quanta. In this case the job with 64 processes is allocated more processing power than the other jobs. Hence, truly equal allocation per job is not possible given the packing constraints assumed.

Consider the performance of the DHC-original quantum allocation policy in Figure 2. DHC-Original performs worse than DHC-L1, DHC-L2 and DHC-L4 at 70% utilization, but better at 90% utilization. This is because DHC-Original has less disparity in quantum allocations (to jobs with varying degrees of parallelism) at higher utilizations. For example, consider the following two scenarios:

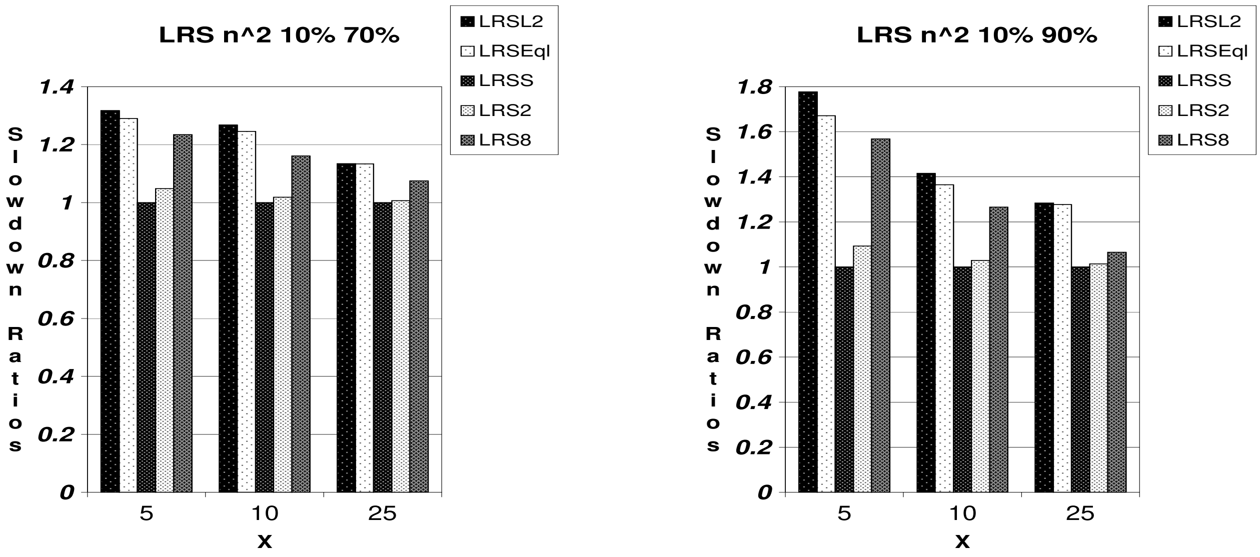

Figure 4: LRS Slowdown Ratios, N^2 X = 10 Workload: Left: 70% Utilization, Right: 90% Utilization

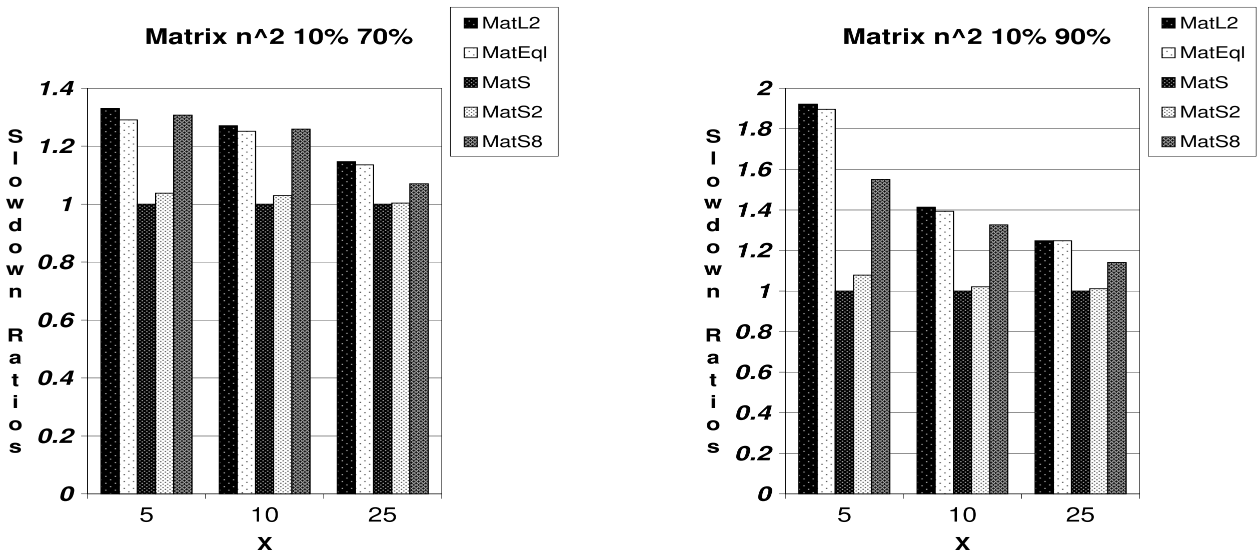

Figure 5: Matrix Slowdown Ratios, N^2 X = 10 Workload: Left: 70% Utilization, Right: 90% Utilization

- Assume there are only two jobs in the system, j_1 with 128 threads and j_2 with 1 thread. In this case, j_1 gets a quantum size of $128/129 * c$ while j_2 gets a quantum size of $1/129 * c$.
- Assume the following jobs in the system: j_1 with 128 threads, j_2 with 64 threads, j_3 with 32 threads, and j_4 and j_5 with 16 threads each. In this case j_1 gets a quantum size of $128/256 * c$ and jobs j_2 through j_5 also get an effective quantum size of $128/256 * c$ due to alternate scheduling.

Thus, the original proposed DHC quantum size allocation algorithm approaches DHC-EQL at high loads, but is more similar to DHC-Lx (for some x) at lower loads.

To make the relative slowdowns more clear, we plot the slowdown ratios for the DHC algorithm in Figure 3. We plot the slowdown ratio of each DHC variant relative to DHC-S1. The three groups of bars from left to right are for X = 5, 10 and 25 %. The extreme variants viz. DHC-L8, DHC-L4 and DHC-S8 are not shown since they compress the scale of the figure. DHC-EQL results in slowdown that are (31, 39, 24) % larger than DHC-S1 for X = (5, 10, 25%) at a 70% utilization, and (17, 14, 13) % larger at a 90% utilization. Overall, the performance of DHC-S1 is comparable or better than all other variants. In some cases DHC-S2 performs slightly better.

For the LRS and Matrix algorithms the difference between alg-EQL and alg-S is even greater (at a 90% utilization) than for DHC as can be seen in Figures 4 and 5. LRS-EQL results in slowdown that are (29, 25, 13) % larger than LRS-S for X = (5, 10, 25%) at a 70% utilization, and (67, 36, 28) % larger at a 90% utilization. Matrix-EQL results in slowdown that are (29, 25, 14) % larger than Matrix-S for X = (5, 10, 25%) at a 70% utilization, and (90, 39, 25) % larger at a 90% utilization. Thus, even bigger improvements can be expected when modifying an existing scheduling algorithm based on Matrix or LRS to allocate the number of quanta proportional to the number of jobs in the row rather than equally per row.

4.2 Geometric-Bounded $N^{1.5}$ Workload

The qualitative results obtained for this workload were similar (within 10%) to those obtained for the Geometric-Bounded N^2 workload. Experimental results are not included for purposes of brevity.

4.3 Geometric-Bounded N Workload

In Figure 6 we plot the slowdowns for the N correlated workload with X = 10 at 70% utilization. The left set of bars is for the LRS variants, the middle set for the DHC variants and the set on the right is for the Matrix variants. Unlike in the previous cases, variants that give equal preference to all jobs have better performance compared to variants that give preference to large/small jobs. This behavior occurs for this workload since job execution time (if run alone) is not correlated to job parallelism. Thus, giving preference to small jobs is counter-productive.

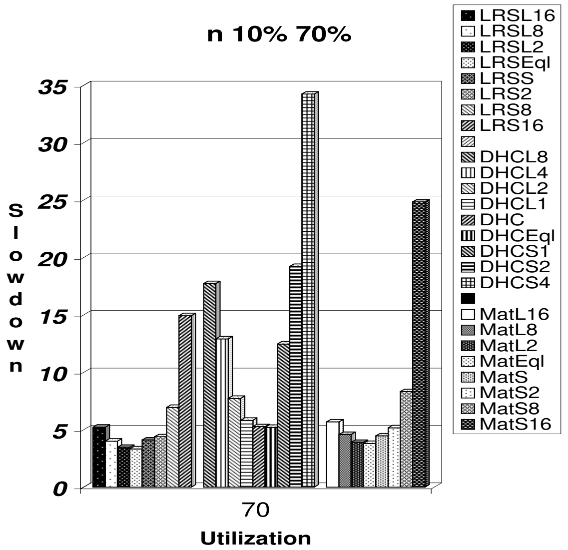

Figure 6: Slowdowns, N X = 10 Workload, Utilization = 70%

4.4 Feitelson96 workload

In Figure 7 we plot the slowdowns of variants of the LRS, DHC and Matrix algorithms respectively for the Feitelson96 workload at 70% and 80% utilization. For this workload too, giving preference to small jobs results in better slowdowns. This is especially true at an 80% utilization where the slowdowns of LRS-EQL and Matrix-EQL are 192% and 188% higher than those of LRS-S and Matrix-S. Note, for this workload there is little difference in DHC-EQL and DHC-S1.

4.5 Response Times

In Figure 8 we plot the mean response times for the N^2 and N workloads assuming X = 10. For the Matrix and LRS policies there is only a minor difference (up to about 20%) between the mean response times of the variants that give preference to large jobs and those that give preference to small jobs. Conversely, DHC exhibits a serious degradation in mean job response time going from DHC-EQL to DHC-S8. We conjecture that this is because giving multiple quanta to the smallest control groups results in idling of processors since larger alternates can not fill the holes left when scheduling the small control groups. The problem is magnified at higher utilizations and also for the N workload.

In Figure 9 we plot the mean response times of the various algorithms (and variants) for the Feitelson96 workload at 70% and 80% utilization. For the Feitelson96 workload, giving preference to small jobs has a significant improvement in the MRT for the LRS and Matrix algorithms at higher

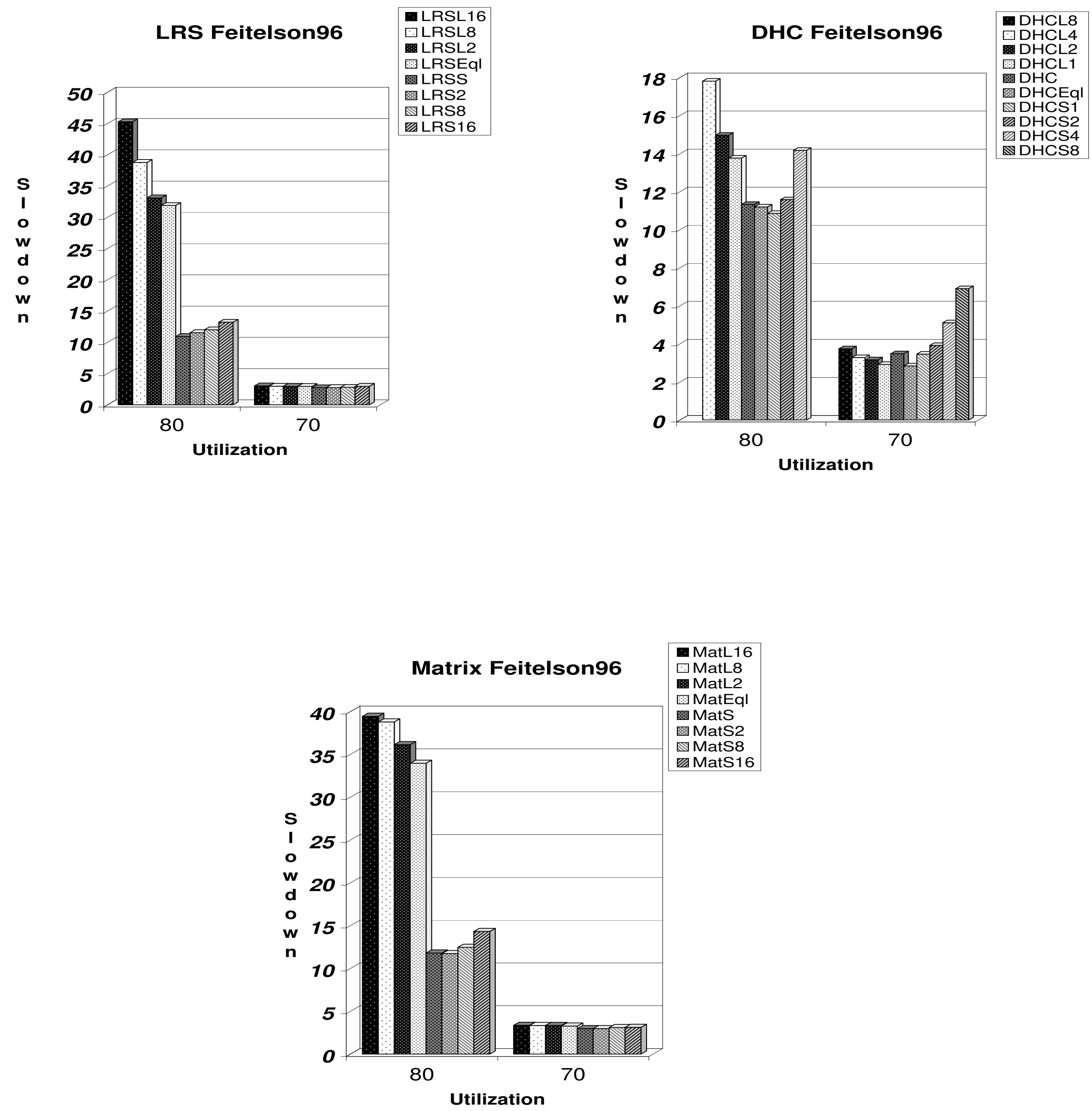

Figure 7: Slowdowns, Feitelson96 Workload: Top Left: LRS, Top Right: DHC, Bottom Left: Matrix

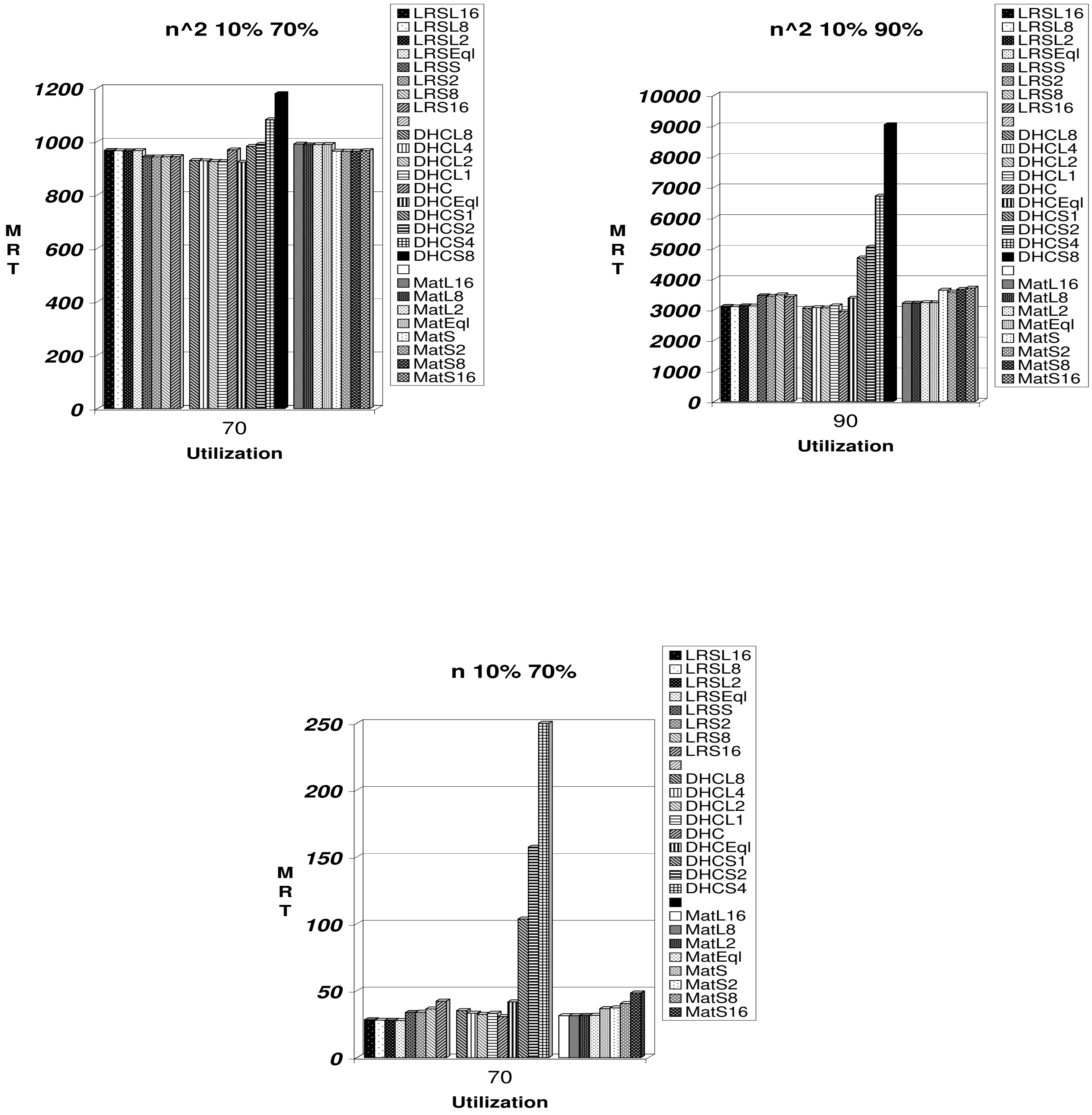

Figure 8: Mean Response Times, Top Left: $N^2X = 10$ Workload at 70% utilization, Top Right: $N^2X = 10$ Workload at 90% utilization, Bottom Left: $NX = 10$ Workload at 70% utilization

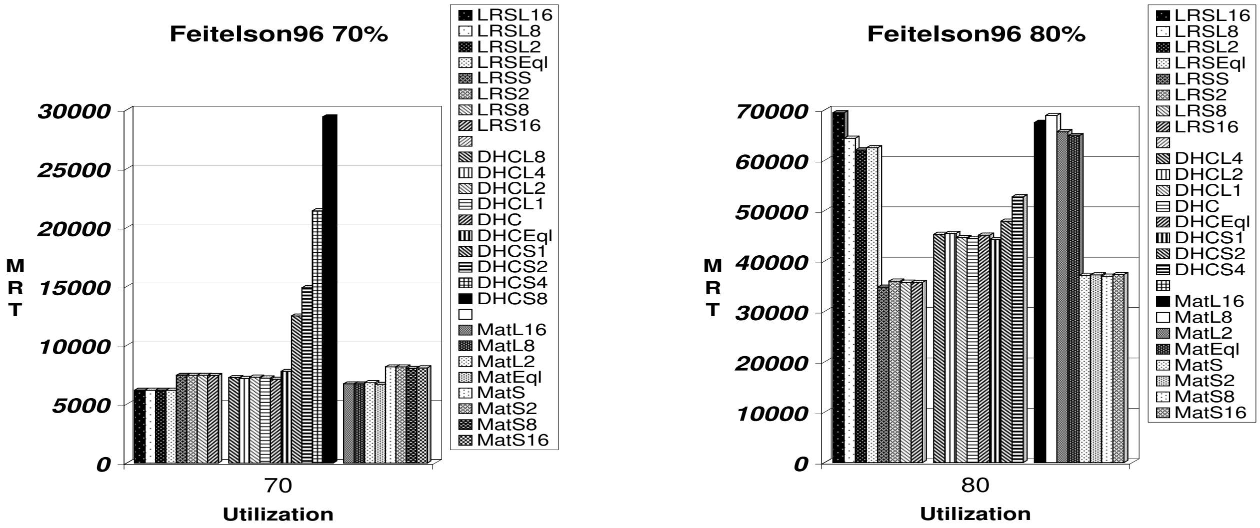

Figure 9: Mean Response Times, Feitelson96 Workload, Left: 70% utilization, Right: 90% utilization

utilization. The MRT of Matrix-EQL (LRS-EQL) is 75% (79%) larger than the MRT of Matrix-S (LRS-S). Again, the MRT of DHC is significantly increased by giving additional quanta to small control groups.

Based on the mean response times of DHC shown in Figures 8 and 9, we suggest that DHC-EQL be used instead of DHC-S1. Another approach would be to detect when the smaller control groups do not fill the processors and allocate fewer quanta.

4.6 Relative performance of algorithms

In this section we compare the relative performance of the 3 algorithms for the N^2 workload. A comparison of LRS-EQL, Matrix-EQL and DHC-original for the Feitelson96 workload can be found in [5], but that comparison did not consider the LRS-S and Matrix-S variants proposed in this paper. We plot the slowdowns and mean response times of the EQL and S variants (S1 in case of DHC) for the Geometric-Bounded N^2 workload in Figure 10. This workload was not considered in previous papers on DHC. We choose only the EQL and S variants since their performance is comparable or better than the other variants of each algorithm.

DHC-S1 has the best slowdowns among these variants at both 70% and 90% utilization, but larger mean response times. LRS performs better than Matrix both in terms of slowdowns and mean response times. DHC-EQL performs marginally better than LRS-S. Thus, for this workload the DHC-EQL and LRS-S policies are the best and about equal.

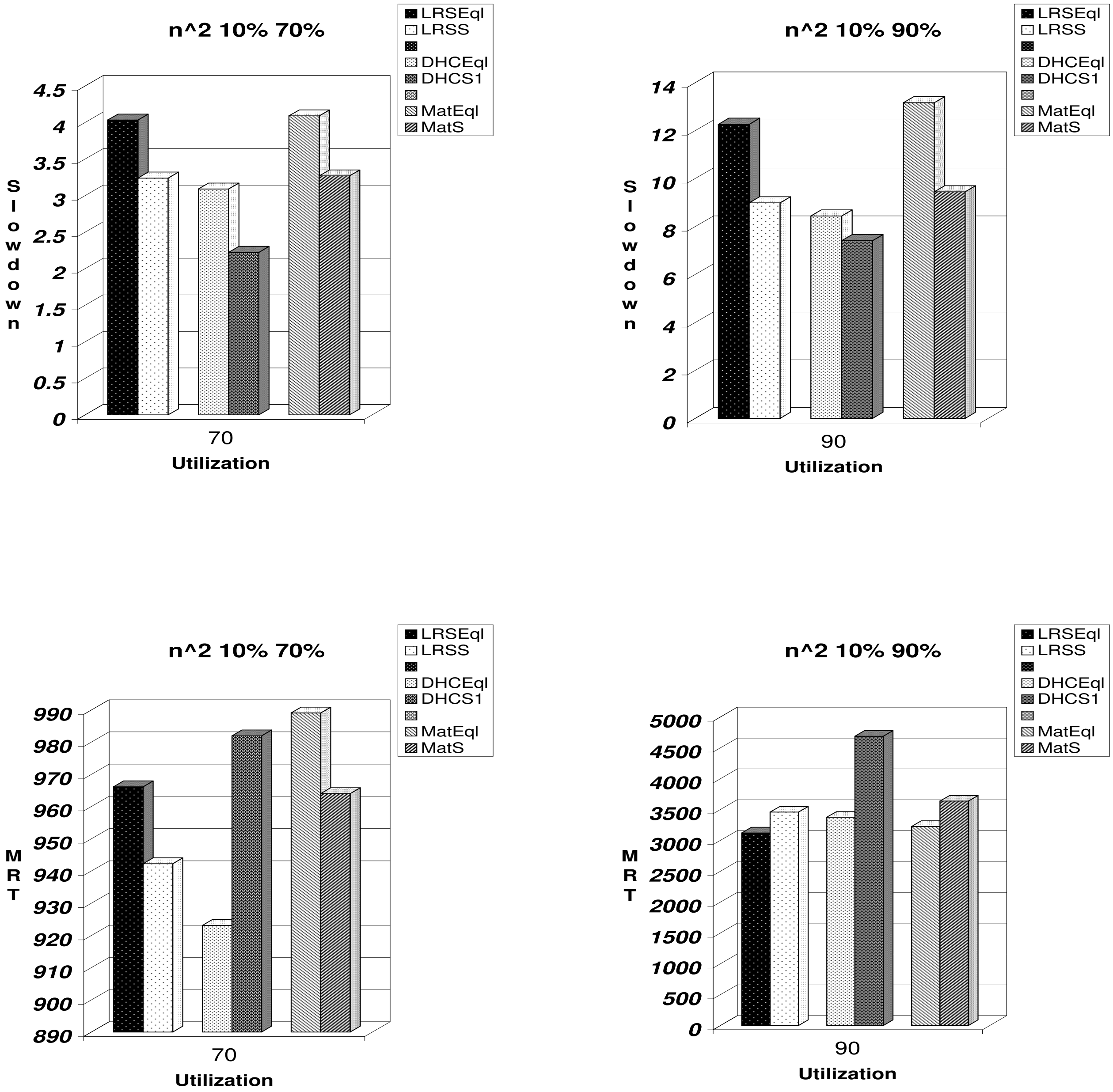

Figure 10: Policy Comparison, Geometric-Bounded N^2 Workload: Top Left: Slowdowns at 70% utilization, Top Right: Slowdowns at 80% utilization, Bottom Left: Mean Response Times at 70% utilization, Bottom Right: Mean Response Times at 80% utilization

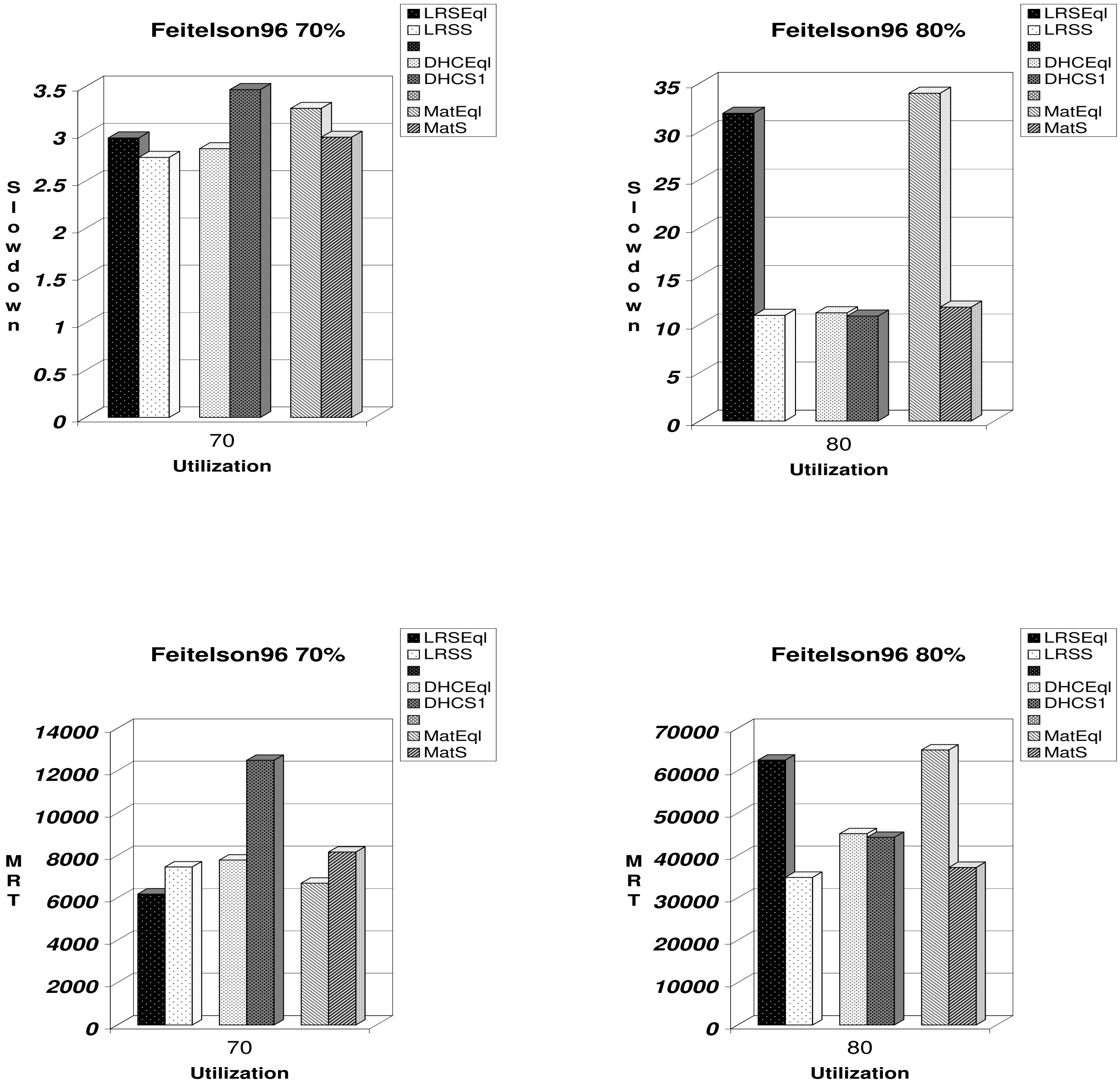

Figure 11: Feitelson96 Workload: Top Left: Slowdowns at 70% utilization, Top Right: Slowdowns at 80% utilization, Bottom Left: Mean Response Times at 70% utilization, Bottom Right: Mean Response Times at 80% utilization

In Figure 11 we plot the slowdowns and mean response times of the EQL and S variants (S1 for DHC) for the Feitelson96 workload. Again, for this workload, the performance of DHC-EQL is comparable to that of LRS-S in terms of slowdown and mean response times.

5 Conclusions

In this paper we showed via a simulation study that for LRS and Matrix allocating quanta inversely proportional to mean job parallelism can significantly decrease mean job slowdowns with little impact on mean job response time. Note, the amount of work to modify existing production gang scheduling policies to incorporate these findings would be trivial and should result in a significant performance gain.

We consider three gang scheduling policies: Matrix, LRS, and DHC. We have run simulations on four different workloads varying numerous job parallelism parameters. Total job demand is super linearly correlated with the number of processes for most of the workloads considered.

In general, we show that mean job slowdown can be decreased by 20 to 50% for the Matrix and LRS algorithms simply by letting the number of quanta allocated to a row equal the number of jobs in the row rather than equal allocation per row. Equal allocation per row gives a dis-proportionate share of processing power to jobs with a large number of processes. These large jobs also have longer execution times, hence this is poor allocation choice. By allocating additional quanta to rows containing jobs with less parallelism we do a better job of allocating equal processing power per job, resulting in reduced mean job slowdowns.

For the DHC policy, we have shown that allocation of additional quanta to control groups containing jobs with a small number of processes sometimes decrease mean job slowdown compared to equal allocation per control group, but at the expense of significantly larger mean job response times. Furthermore, the performance of the quantum size allocation policy originally proposed performs similar to equal allocation per quantum.

Finally, we compared the performance of LRS-S, Matrix-S, and DHC-EQL. In earlier work LRS-EQL, which is worse than LRS-S, was shown to result in larger slowdowns than DHC. We show that LRS-S and DHC-EQL have similar performance across all studies considered. The DHC algorithm has the advantage of being less centralized than LRS, making it more attractive in a distributed or massively parallel setting, but the LRS-S algorithm is considerably simpler to implement and achieves comparable performance.

References

[1] J.M. Barton and N. Bitar. A scalable multi-discipline, multiple-processor scheduling framework for irix. In *Proc. of the IPPS'95 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 45–69. Springer LNCS #949, 1995.

[2] S-H. Chiang and M. Vernon. Dynamic vs. static quantum-based parallel processor allocation. In *Proc. of the IPPS'96 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 200–223. Springer LNCS #1162, 1996.

[3] Thinking Machines Corporation. Connection machine cm-5 techinical summary. Technical report, nov 1992.

[4] Intel Supercomputer Systems Divsion. Paragon users guide. Technical Report Order number 312489-003, jun 1994.

[5] D. Feitelson. Packing schemes for gang scheduling. In *Proc. of the IPPS'96 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 65–88. Springer LNCS #1162, 1996.

[6] D. Feitelson and M.A. Jette. Improved utilization and responsiveness with gang scheduling. In *Proc. of the IPPS'97 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 238–261. Springer LNCS #1291, 1997.

[7] D. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *Proc. of the IPPS'98 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 1–24. Springer LNCS #1459, 1998.

[8] D. G. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306–318, December 1992.

[9] D. G. Feitelson and L. Rudolph. Evaluation of design choices for gang scheduling using distributed hierarchical control. *Journal of Parallel and Distributed Computing*, 35(1):18–34, May 1996.

[10] D.G. Feitelson and B. Bitzberg. Job characteristics of a production parallel scientifc workload on the nasa ames ipsc/860. In *Proc. of the IPPS'95 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 337–360. Springer LNCS #949, apr 1995.

[11] D.G. Feitelson and L. Rudolph. Distrubuted hierarchical control for parallel processing. *IEEE Computer*, 23(5):65–77, 1990.

[12] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds.* Prentice Hall, 1995.

[13] P.A. Fishwick. Simpack: Getting started with simulation programming in c and c++. In *Proc. 1992 Winter Simulation Conference*, pages 154–162, 1992.

[14] A. Hori, H Tezuka, and Y Ishikawa. Overhead analysis of preemptive gang scheduling. In *Proc. of the IPPS'98 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 217–230. Springer LNCS #1459, 1998.

[15] A. Hori, H Tezuka, Y Ishikawa, N. Soda, H Konaka, and M. Maeda. Implementation of gang-scheduling on workstation cluster. In *Proc. of the IPPS'96 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 126–129. Springer LNCS #1162, 1996.

[16] S. Hotovy. Workload evolution on the cornell theory center ibm sp2. In *Proc. of the IPPS'96 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 27–40. Springer LNCS #1162, 1996.

[17] N. Islam, A. Prodromidis, M.S. Squillante, , A.S. Gopal, and L.L. Fong. Extensible resource scheduling for parallel scientific applications. In *Proc. Eight SIAM Conf on Parallel Processing for Scientific Computation*, mar 1997.

[18] N. Islam, A. Prodromidis, M.S. Squillante, L.L. Fong, and A.S. Gopal. Extensible resource managmeent for cluster computing. In *Proc. 1997 Int. Conf. on Distributed Computing Systems (ICDCS-97)*, 1997.

[19] M. Jette. Expanding symmetric multiprocessor capability through gang scheduling. In *Proc. of the IPPS'98 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 199–216. Springer LNCS #1459, 1998.

[20] S.T. Leutenegger and M.K. Vernon. Performance of multiprogrammed multiprocessor scheduling policies. In *Proc. 1990 ACM SIGMETRICS*, May 1990.

[21] R. K. Mansharamani and M.K. Vernon. Properties of the eqs paralle processor allocation policy. Technical Report 1192, Computer Sciences Department, University of Wisconsin-Madision, nov 1993.

[22] J. Ousterhout. Scheduling techniques for concurrent systems. In *Proc. of Distributed Computing Systesms Conference*, pages 22–30, 1982.

[23] P. Sabalvarro, S. Pakin, W. Weihl, and A.A. Chien. Dynamic coscheduling on workstation clusters. In *Proc. of the IPPS'98 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 231–256. Springer LNCS #1459, 1998.

[24] U. Schwiegelshohn and R. Yahyopour. Improving first-come-frist-serve job scheduling by gang scheduling. In *Proc. of the IPPS'98 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 180–198. Springer LNCS #1459, 1998.

[25] M.S. Squillante, F. Want, and M. Papaefthymiou. Stochastic analysis of gang scheduling in parallel and distributed systems. *Performance Evaluation*, 27:273–296.

[26] F. Wang, H. Franke, M Papaefthymiou, P Pattnaik, L. Rudolph, and M. Squillante. A gang scheduing design for multiprogrammed parallel computing environments. In *Proc. of the IPPS'96 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 111–125. Springer LNCS #1162, 1996.

[27] F. Wang, M Papaefthymiou, and M. Squillante. Performance evaluation of gang scheduling for parallel and distributed multiprogramming. In *Proc. of the IPPS'97 Workshop on Job Scheuling Strageies for Parallel Processing*, pages 277–298. Springer LNCS #1291, 1997.