# The Impact of Job Memory Requirements on Gang-Scheduling Performance

Sanjeev Setia
Computer Science Department
George Mason University Fairfax, VA 22030
setia@cs.gmu.edu

Mark S. Squillante, Vijay K. Naik
IBM Thomas J. Watson Research Center
P.O. Box 704 Yorktown Heights, NY 10598
{mss,vkn}@watson.ibm.com

**Abstract**

*Almost all previous research on gang-scheduling has ignored the impact of real job memory requirements on the performance of the policy. This is despite the fact that on parallel supercomputers, because of the problems associated with demand paging, executing jobs are typically allocated enough memory so that their* entire address space *is memory-resident. In this paper, we examine the impact of job memory requirements on the performance of gang-scheduling policies. We first present an analysis of the memory-usage characteristics of jobs in the production workload on the Cray T3E at the San Diego Supercomputer Center. We also characterize the memory usage of some of the applications that form part of the workload on the LLNL ASCI supercomputer. Next, we examine the issue of long-term scheduling on MPPs, i.e., we study policies for deciding which jobs among a set of competing jobs should be allocated memory and thus should be allowed to execute on the processors of the system. Using trace-driven simulation, we evaluate the impact of using different long-term scheduling policies on the overall performance of Distributed Hierarchical Control (DHC), a gang-scheduling policy that has been studied extensively in the research literature.*

## 1 Introduction

The topic of job scheduling strategies for parallel computers has received considerable attention in recent years. Many different policies have been proposed (e.g., dynamic space-sharing [17, 19] and gang scheduling [12, 4]) and several issues (e.g., space-sharing vs time-sharing [10], static vs dynamic partitioning [19]) have been analyzed in the research literature.

Most of the early work on this topic ignored the constraints imposed by the memory requirements of jobs on the processor scheduling policies. While several recent studies have considered the impact of memory on job scheduling performance [14, 11, 1, 15, 2, 13, 5], the interaction between memory allocation and processor scheduling is still not well understood in practice, primarily due to the limited data on real job memory requirements.

One of the job scheduling policies that has been studied extensively is gang scheduling[12, 4]. Many commercial parallel supercomputers, including the Cray T3E, the Intel Paragon, the IBM SP2 and the Meiko CS-2, provide some support for gang scheduling. In this paper, we consider the impact of the memory requirements of real parallel jobs on gang-scheduling performance.

Under gang-scheduling, parallel jobs time-share the processors of the system in a coordinated manner; typically, several jobs are memory resident at the same time. Hence, there is competition among jobs not only for processors but also for memory. When a job is submitted to the system, the scheduler has to make two decisions: first, whether to allocate memory to the job, and second, which processors to allocate to the job. On a distributed memory parallel computer, these two decisions cannot be made independently because the amount of memory available to a job depends upon the number of processors allocated to it. As in classical operating systems, the process of making memory allocations decisions is referred to as *long-term scheduling.*

When the memory available on a computer is large relative to the memory demands of incoming jobs, the memory requirements of jobs have little or no impact on processor allocation decisions. Further, there is no competition among jobs for memory, so jobs are hardly ever swapped out of memory. In other words, long-term scheduling has a minimal impact on the system and can be ignored in the performance evaluation of the job scheduling policy in these environments. On the other hand, when the memory requirements of jobs are large relative to the amount of available memory, long term scheduling can have a significant impact on the overall performance of the job scheduling policy.

Almost all previous work on gang-scheduling has ignored the impact of long-term scheduling on the performance of the policy. This is despite the fact that on parallel supercomputers, because of the problems associated with demand paging, executing jobs are typically allocated enough memory so that their *entire address space* is memory-resident [1, 14, 15, 5]. Given the large data sets associated with several of the scientific applications that constitute the bulk of the workload on parallel supercomputers, the impact of the memory requirements of real jobs on the performance of job scheduling is likely to be significant. In this paper, we present results that support this conclusion.

We make two contributions. First, we report on the memory-usage characteristics of jobs in the workload on the Cray T3E at SDSC. Our analysis is based on job logs from May 1998, and as such, represents information about workloads on a current generation supercomputer. We also present an analysis of memory requirements of jobs that form part of the workload for the ASCI supercomputer at the Lawrence Livermore National Laboratory. Our analysis of memory-usage characteristics indicates that many jobs in a typical MPP workload are likely to have memory requirements that are large relative to the amount of available memory.

Second, using trace-driven simulation, we evaluate the impact of using different long-term scheduling polices on the performance of Distributed Hierarchical Control (DHC), a gang-scheduling policy that has been studied extensively in the literature [4, 3]. Our results show that the constraints imposed by the memory requirements of jobs have a significant impact on DHC performance. Further, we show that the interaction between long-term scheduling and processor scheduling is complex, and definitely requires further research.

The structure of the rest of this paper is as follows. In Section 2, we discuss in more detail the various levels at which scheduling decisions are made in parallel supercomputers. In Section 3, we present an analysis of the memory-usage characteristics of jobs in the production workloads. Section 4 contains our simulation study of the performance of DHC taking long-term scheduling into account. Finally, Section 5 presents our conclusions.

## 2 Scheduling Framework

A large number of scheduling strategies and variations have been proposed and examined for different parallel computing environments. In this section we present and consider a unified scheduling framework and model to better understand the fundamental aspects of different parallel scheduling policies and to systematically compare and examine the various scheduling decisions made at different levels. This unified scheduling framework is illustrated in Figure 1.

On parallel computers, scheduling can be thought of as taking place at three levels:

- Long-term Scheduling – at this level, the scheduler decides which jobs are allocated memory and thus allowed into the "system". This decision is based on several criteria – one is obviously the memory required by the job and the amount of memory available. Other possible criteria include the class of the job (interactive, batch, etc.) and the number of processors requested. Jobs may be allowed to reside in memory until they complete, or a preemptive policy may be used under which jobs can be swapped out of memory by higher priority jobs, possibly after checkpointing. Memory can be time-shared in the sense that after a time quantum $T_L$ (or after its priority has decayed to a certain level), jobs executing in the system will be swapped out of memory and return to the long-term queue. The time quantum $T_L$ must be selected keeping in mind the relatively large overheads for swapping jobs in and out of memory, as well as for checkpointing if used.

- Medium-term scheduling – This level is responsible for constructing the scheduling "matrix", i.e., mapping jobs to processors. Under most gang-scheduling policies discussed in the literature, once a job is mapped on to a set of processors, it executes on those processors until it finishes or it is swapped out of memory. However, under some policies, jobs can be migrated and/or reconfigured from one set of processors to another during their execution. In other words, the scheduling matrix can be "repacked", where repacking involves migrating jobs from one set of processors to another and/or reconfiguring executing jobs. The interval between repackings can be thought of as the time quantum, $T_M$, before a jobs returns to the medium-term queue. This time quantum $T_M$ is decided based on the overheads incurred by a job while it is being reconfigured or migrated for repacking. The interval between changes to the scheduling matrix can also be dynamic with $T_M$ as a minimum.

- Short-term scheduling – at this level, node level schedulers decide which process is going to run next. Mechanisms for coordinated context switching are used to ensure that jobs are gang-scheduled. The length of the short-term time-sharing quantum $T_S$ should be long enough to amortize the overheads of context switching on a parallel system, e.g., switch reconfiguration time.

Within this scheduling framework, let us look at a few of the policies that have been studied in the literature.

Gang-scheduling policies such as DHC[3] have a long-term queue consisting of jobs that cannot be scheduled because they require more memory than is available. However, once jobs are scheduled they never return to the long-term queue. Thus, there is no long-term time-sharing. Medium-term scheduling takes place exactly once for each job when it is mapped to a set of processors. There is no medium-term time-sharing because the scheduling matrix is never repacked. Thus DHC can be thought of as a policy in which time-sharing only occurs at the level of the short-term scheduler.

If the scheduling matrix is periodically repacked, as under the policy studied by Chandra *et al* [2], or dynamically reconfigured and adjusted, as under the policy studied and implemented in the Octopus project at IBM Research [6], then the jobs can be thought of as returning to the medium-term queue when the matrix is repacked and/or reconfigured. The Lawrence Livermore gang-scheduling implementation on the Cray T3D [8] and the FB-IQA policy studied by Setia [16] do not have short-term time-sharing since there is only one job allocated to a processor at any given time. However,

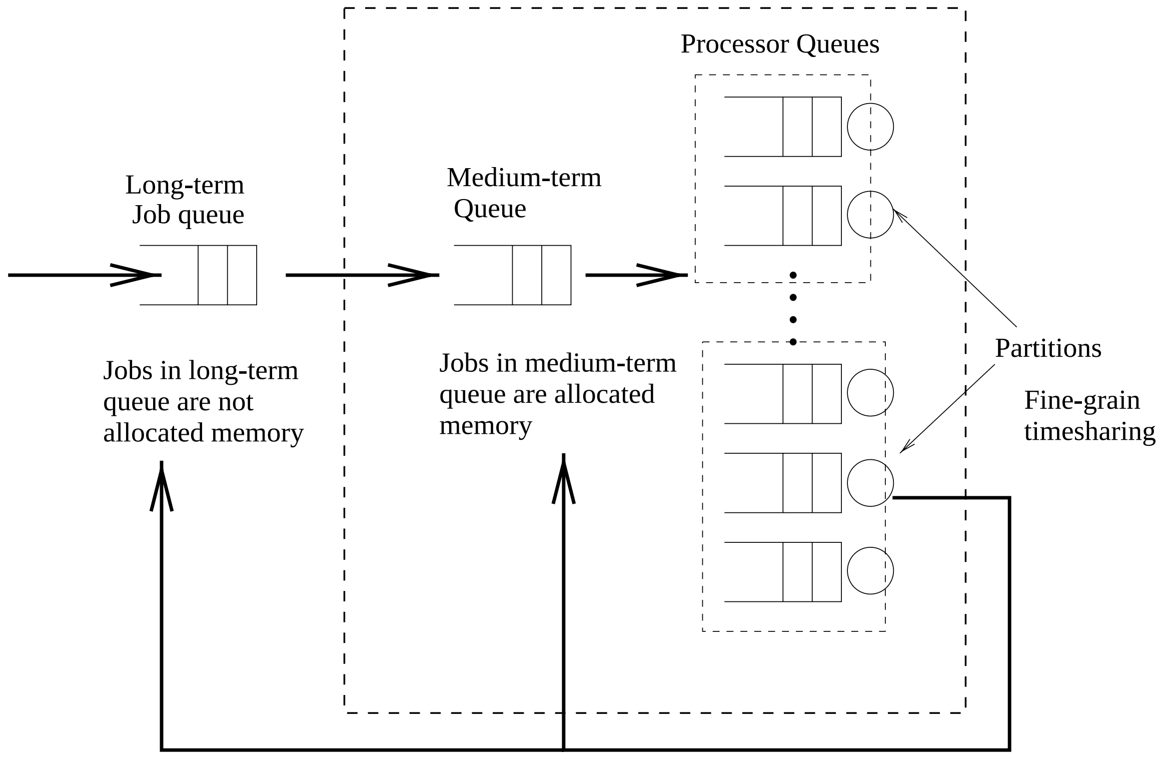


Figure 1: A Unified Scheduling Framework.

jobs are periodically swapped in and out of memory – thus, there is long-term time-sharing under both policies. Under the FB-IQA policy, jobs can be migrated from one set of processors to another. Thus, FB-IQA has both medium-term and long-term time-sharing. Finally, the gang-scheduling strategy in Octopus provides short-term, medium-term and long-term scheduling, although long-term time-sharing has not been fully exploited [6].

In this paper, we focus on the performance of DHC while taking long-term scheduling into account. We study both preemptive and non-preemptive long-term scheduling policies. Under the non-preemptive policies, there is no long-term time-sharing, i.e, $T_L = \infty$. In other words, once a job has been allocated memory, it is never swapped out. Under the preemptive policy, jobs return to the long-term queue after they have executed for more than $T_L$ time units. Under all of these policies, the long-term and medium-term scheduler act in concert; a job is only allocated memory if it can be allocated processors and vice versa.[1]

## 3 Memory Usage Characteristics

One of the reasons why long-term scheduling has received relatively little attention in the context of gang-scheduling is that there has been very little information available about the memory-usage characteristics of real parallel jobs. The only previous study that has reported memory-usage characteristics is Feitelson's study [5] of the workload on the CM5 at Los Alamos National Laboratory.

We now examine the memory-usage characteristics of jobs submitted to the Cray T3E at SDSC in May, 1998. Table 1 summarizes the statistics for this workload. Jobs submitted to the system can be classified into two categories: batch jobs, which are submitted via NQS, and interactive "jobs", which actually correspond to a session consisting of a series of commands. There are 1467 batch jobs and 1053 interactive sessions in the workload. The statistics for the interactive session are totals for all commands during the session; hence, we do not report on statistics such as service time that are only meaningful for individual jobs. The memory usage and parallelism reported are the maximum memory usage and parallelism for the commands in the interactive session. The Total CPU Time statistic reported in the table corresponds to the actual cpu time consumed by the batch job or interactive session.

Our analysis shows that the characteristics of this workload are similar to those of other MPP workloads that have been reported in the literature. Specifically, there is a large variance in interarrival times, service times, and total cpu time. The distribution of parallelism of the jobs resembles the distributions observed in other workloads with the exception that there are no sequential jobs in the workload. Another difference from

[1] Note that if the degree of multiprogramming (the number of slots in the scheduling matrix) has an upper bound for performance reasons, then a job cannot be allocated processors if all the slots are "full".

|  |  | Mean | Coeff. of variation | Minimum | Maximum |
|---|---|---|---|---|---|
| All jobs | Interarrival time (seconds) | 1141 | 1.78 | 0 | 32905 |
|  | Total CPU Time | 227191 | 3.2 | 0.19 | 9208799 |
|  | Parallelism | 28 | 1.3 | 2 | 260 |
|  | Memory Usage Per Node (MB) | 28.5 | 1.22 | 0.7 | 116 |
| Interactive sessions | Total CPU Time | 44272 | 7.7 | 0.19 | 5526746 |
|  | Parallelism | 22 | 1.4 | 2 | 260 |
|  | Memory Usage Per Node (MB) | 20 | 1.71 | 0.7 | 116 |
| Batch jobs | Service Time (seconds) | 51059 | 1.5 | 2 | 782305 |
|  | Total CPU Time | 358489 | 2.47 | 0.34 | 9208799 |
|  | Parallelism | 32 | 1.23 | 2 | 128 |
|  | Memory Usage Per Node (MB) | 34.6 | 0.99 | 2.9 | 34.5 |

Table 1: Statistics for the workload on the SDSC Cray T3E

other workloads is that the batch jobs in the workload have much larger service times; the average service time is over 14 hours, and the largest service time is more than 9 days.

The workload characteristic of greatest interest to us is the per-processor memory usage of each job. Here we find that the memory-usage patterns of jobs in the workload resemble the memory-usage patterns exhibited by jobs in the LANL CM5 workload. The average memory used per processor by jobs in the SDSC Cray T3E workload is much higher than in the LANL CM5 workload. This is to be expected since each node on the Cray T3E has 128 MB of memory, while the CM5 has 32 MB of memory per node. However, there are strong similarities between the two workloads:

- The distribution of memory requests is wide. In Figure 2, we plot the frequency of the per-processor memory usage for the workload. We observe that more than half of the jobs in the workload have per-processor memory usage less than 10 MB. However, there are a significant number of jobs that have large memory requirements.

- Long running jobs tend to use more processors and more memory than other jobs. To illustrate this, we divided batch job into five categories based on their execution time. Figure 3 (a) and (b) plot the average memory usage per processor and the average parallelism as a function of the job category. These figures show that there is a positive correlation between execution time and job parallelism, and also between execution time and per-processor memory usage.

These characteristics have two important implications for gang-scheduling policies. First, since most jobs have small memory requirements, relatively fine-grain (or short-term) time-slicing among several memory-resident jobs is distinctly possible. Second, since many long-running jobs have large memory requirements, relatively coarse-grain (or long-term) time-sharing is probably necessary for providing good service to these jobs while not penalizing smaller jobs.

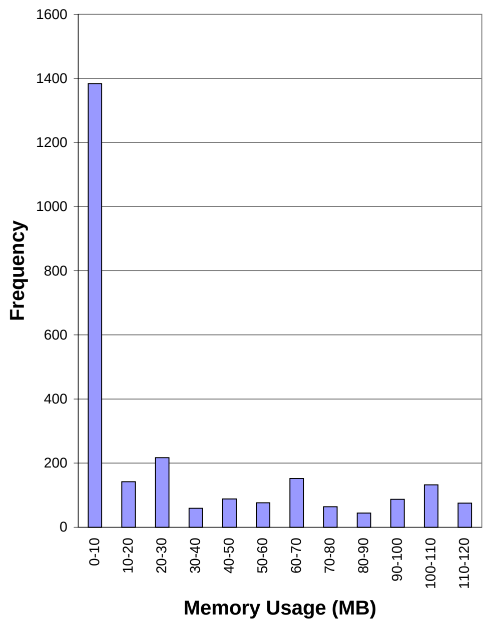
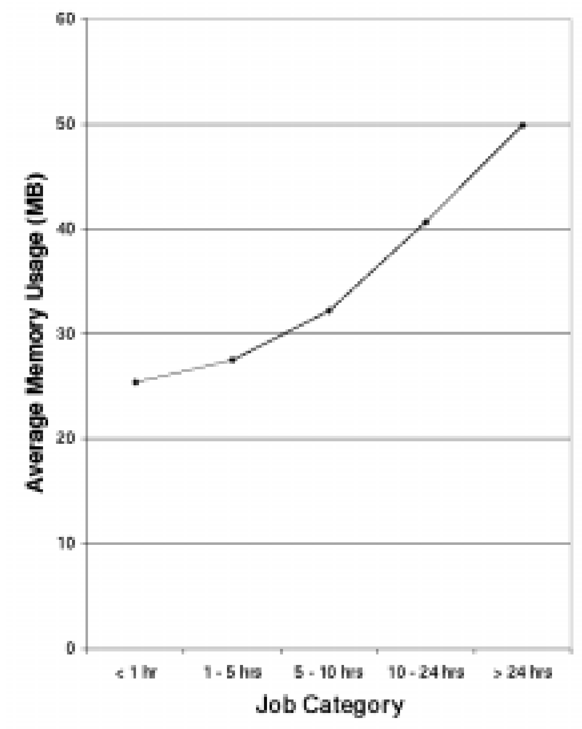


Figure 2: Distribution of per-processor memory usage for jobs in the workload.

### 3.1 Characterization of Application Memory Requirements

We now report on the memory-usage characteristics of some of the applications that are expected to



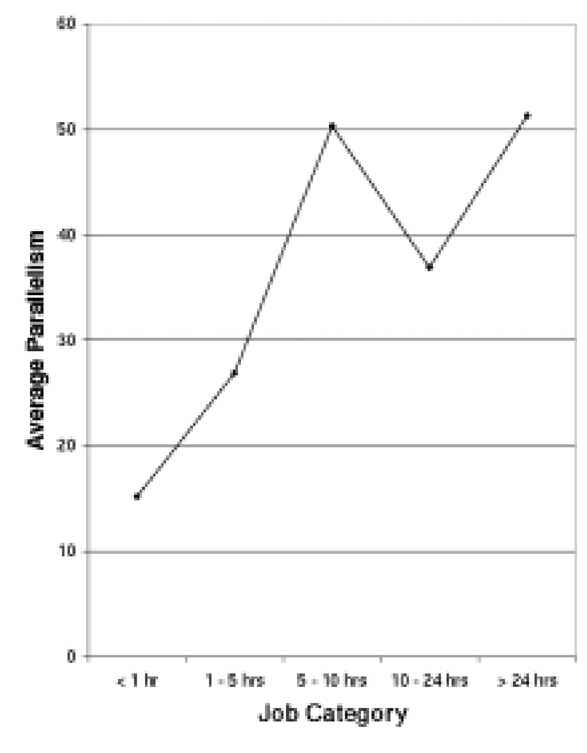


Figure 3: Correlation between job execution time and (a) memory usage per node , and (b) Job Parallelism.

| Application | Data Set Size |
|---|---|
| Camille | 15.6 MB |
| Ale3D | 18 GB |
| Paradyn | 70 MB |
| F3D | 19 GB |
| ICF3D | 8 GB |
| Pagosa | 24 GB |
| Tecolote | 28 GB |
| Flag | 80 GB |
| Parflow | 800 MB |

Table 2: Total Problem Memory Size for applications that form part of the workload for the ASCI supercomputer at LLNL.

form the workload on the ASCI supercomputer at the Lawrence Livermore National Laboratory. Table 2 reports the data set size for nine important scientific applications. We observe that the total data set sizes for these applications are large and in most cases are of the order of several Gigabytes. The main point illustrated by this table is that although the per node memory sizes on the ASCI supercomputers are of the order of 2 GB, the memory demands of the applications that are expected to run on these supercomputers have correspondingly large data set sizes. Further, given the low swapping speeds observed on these systems [7], the importance of long-term scheduling is likely to increase rather than diminish in such systems, and long-term scheduling can also effectively exploit block-paging methods [18] to reduce the costs of swapping (as well as paging).

## 4 Performance Evaluation of Long-term Scheduling Policies

In this section, we evaluate the impact of job memory requirements on the performance of the DHC gang-scheduling policy. Using trace-driven simulation, we evaluate the performance of DHC when it is combined with various long-term scheduling scheduling policies.

### 4.1 Simulation Model and Methodology

**Workload Model** Our simulation was driven by a trace that was derived from the workload described in Section 3. This trace consists of all the jobs in the workload with the exception of some batch jobs that had very small CPU times in relation to their execution times, e.g. execution time of several hours but total CPU time of a few minutes. Since we were not sure of the reason for this behavior,[2] we decided to eliminate them from our workload. We also decided to treat the interactive sessions in the workload as interactive jobs. This preserves many of the characteristics of the workload such as the job arrival and memory-usage patterns, and is preferable to creating a separate synthetic trace for the interactive jobs.

[2]One possible reason is that these jobs were excessively I/O bound.

**System Model** To evaluate policy performance at different utilizations without changing any of the characteristics of the workload, we varied the number of processors in the system (denoted by $P$) from 768 to 1152 in increments of 128 processors. The offered loads corresponding to these values of $P$ are shown in Table 3. Note that this is the offered load corresponding to the jobs in the trace, not the actual load on the system. As we will see later in this section, some jobs may not get scheduled during a simulation run because their memory requirements cannot be satisfied. This has the effect of reducing the actual utilization of the system.

| $P$ | Offered Load |
|---|---|
| 1152 | 0.65 |
| 1024 | 0.73 |
| 896 | 0.84 |
| 768 | 0.98 |

Table 3: Offered load corresponding to different system sizes for the workload trace.

We considered two memory configurations for the system with per processor memory ($M$) of 128 and 256 MB. Note that the SDSC Cray T3E has 128 MB on each node. We assumed that jobs could be swapped in and out of memory at a rate of 20 MB/sec, and that each processor transfers its portion of the job to disk in parallel. We believe that this rate should be achievable given current network and disk technology.

**Metrics** The primary metrics used to evaluate the performance of a policy are the average response time (denoted by $R$) and the average normalized response time (denoted by $U$), where the normalized response time is defined as the ratio of a job's response time to its service time. For our workload, the average normalized response time is dominated by the normalized response time of interactive jobs, whereas the average response time is dominated by the response times of long running batch jobs.

## 4.2 Performance Results

As discussed in Section 2, on a distributed memory MPP, memory allocation and processor allocation decisions need to be made in concert. Under DHC, the long-term queue contains all jobs that are waiting to be allocated processors and memory. We assume that the long-term queue is partitioned into two queues – one for interactive jobs and one for batch jobs. Interactive jobs have higher priority than batch jobs.

Under the base policy, if a newly arrived job cannot be allocated processors because the available memory is less than the memory required by the job, it is added to the end of the interactive or batch long-term queues depending upon its class. Otherwise, it is mapped on to a set of processors using the DHC algorithm. (The reader is referred to [3] for details of this mapping scheme.) Once a job is mapped on to a set of processors, it executes on those processors until it completes. Whenever a job departs the system, the long-term scheduler examines first the interactive job queue and then the batch job queue to see if any of the queued jobs can be allocated memory. Within their class, jobs are examined in order of arrival; thus our base long-term scheduling policy is a FCFS policy.

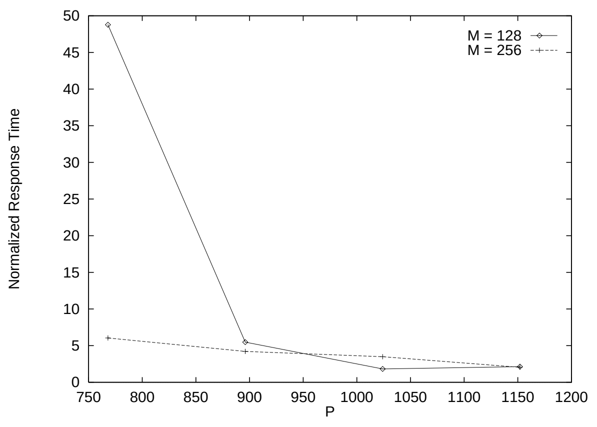
Figure 4 plots $R$ and $U$ for the base policy as a function of $P$ when $M = 128$ and 256. Surprisingly, DHC has better performance for $M = 128$ than it does for $M = 256$. At high loads ($P = 896$ and 768), $U$ is higher for the small memory configuration because interactive jobs experience larger delays than they do for $M = 256$. However, at moderate loads ($P = 1024$ and 1152), $U$ is lower for $M = 128$. Further, $R$ is lower at all loads for $M = 128$ than it is for $M = 256$.

This result shows that the amount of memory available on the system can have an unexpected impact on the performance of a gang-scheduling policy. Not only is this performance trend unexpected but it is also undesirable, since we would naturally want to see an improvement in performance upon adding more memory to the system.

In order to better understand the reasons for this unexpected behavior, we identified two aspects of the long-term scheduling policy that were affected by the amount of memory available on the system. First, decreasing the amount of memory available on the system tends to decrease the priority of the jobs with large memory requirements in the long-term queue. This is especially the case at high loads, when there is a large demand for memory. In such situations, jobs with large memory requirements cannot be scheduled since the amount of available memory is smaller than is required by the job. Second, decreasing the amount of available memory tends to reduce the degree of multiprogramming on the system.

To quantify the performance impact of these aspects of the long-term scheduling policy, we made two modifications to our base long-term scheduling policy. First, we changed the FCFS ordering of the long term queue to a Smallest Memory First (SMF) ordering. This has the effect of making the priority of a job in the long-term queue independent of the amount of memory installed. Second, we introduced a policy input parameter ($D$) that controlled the degree of multiprogramming on the system. In the context of DHC, this parameter controls the maximum number of slots in the scheduling matrix.

**Effect of Job Ordering Policy** Figure 5 plots $R$ and $U$ for the FCFS and SMF policies with $M = 128$ and 256 as a function of $P$. We observe that SMF long-term scheduling results in lower $R$ for both $M = 128$ and 256.



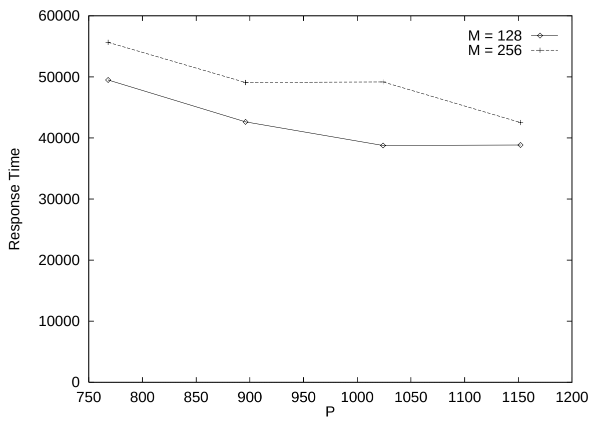


Figure 4: Average normalized response time ($U$) and average response time ($R$) for the base policy.

This result can be explained by the positive correlation between memory requirements and job execution times. Smallest Memory First tends to give a higher priority to shorter jobs, which results in improved response time performance.

However, the response time for $M = 128$ is still significantly smaller than it is for $M = 256$. At high loads, SMF also results in better performance than FCFS for interactive jobs when $M = 128$. This implies that the order in which jobs are selected from the long-term scheduling queue is not the direct reason for the difference in performance of the base policy for $M = 128$ and $M = 256$.

**Effect of Restricting the Degree of Multiprogramming** We varied $D$ from 1 to 7 in increments of 2 for both $M = 128$ and $M = 256$. Figure 6 plots $R$ and $U$ for each configuration. When $M = 128$, increasing $D$ beyond 3 has no effect on performance, hence we only plot the curves for $D = 3$. Similarly, restricting $D$ to 1 implies that $M$ has no impact on performance; hence we plot a single curve for $D = 1$.

We observe that, for our workload, $D = 1$ results in the lowest response time. At the same time, $D = 1$ has the largest normalized response time since there is no time-sharing. Clearly, a policy that results in poor performance for interactive jobs is not acceptable. Nevertheless, the fact that batch jobs have the best performance when $D$ is 1 sheds light on the reason why the overall response time under the base policy is lower for $M = 128$ than it is for $M = 256$.

We also observe that the response time for $M = 256$ is much worse than the performance for $M = 128$, even when $D = 3$ in both cases. Further, when $M = 256$, setting $D$ to 3 results in very poor performance for interactive jobs.

These results are helpful in understanding the impact of memory on the performance of DHC. When the available memory is large relative to the memory requirements of jobs, e.g., when $M = 256$, and the degree of multiprogramming is small ($D \leq 3$), interactive jobs have poor performance. This is because the jobs in memory occupying the slots in the scheduling matrix tend to be batch jobs, and interactive jobs experience large delays before they can be scheduled. On the other hand, when the available memory is small relative to the memory requirements of jobs, e.g., when $M = 128$, and the degree of multiprogramming is larger than 1 ($D = 3$), interactive jobs have good performance. This is because the small amount of available memory itself tends to reduce the number of batch jobs that can be allocated memory and interactive jobs do not experience any delays before they can be scheduled.

This suggests that it is not enough to restrict the degree of multiprogramming to get good performance for the workload under consideration – instead, it is necessary to restrict the number of batch jobs that are allocated memory on a processor. To confirm this reasoning, we introduced another input parameter to the long-term scheduling policy that controlled the maximum number of batch jobs that can be allocated memory on a processor. We call this input parameter the batch job limit, denoted by $B$.

In Figure 7, we plot $R$ and $U$ for $M = 128$ and $M = 256$ with $B = 1$. For comparison, we also plot the curve for $M = 128$ when there is no batch job limit. We observe that setting $B = 1$ results in DHC having practically the same response time for $M = 128$ and $M = 256$. Further, interactive jobs have much better performance when $B = 1$ than when there is no upper limit on the number of batch jobs. At the same time, the performance of batch jobs does not appear to be unduly affected by setting $B$ to 1.

Overall, we can conclude that, for the workload under consideration, the best non-preemptive long-term scheduling policy is one that restricts the degree of multiprogramming of batch jobs to be no larger than one and schedules jobs in the long-term queue in a Smallest Memory First order. Unlike our base policy, this policy has the desirable property that its performance is

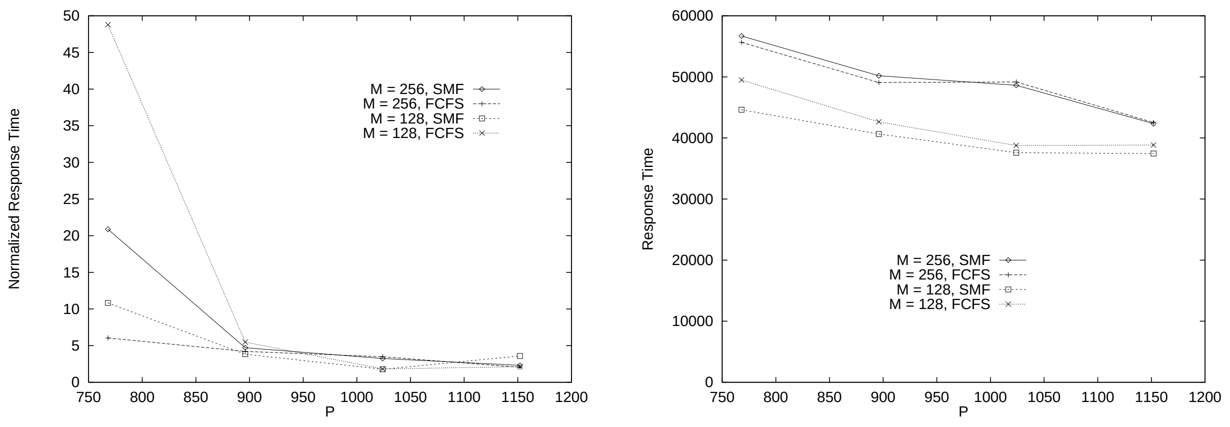


Figure 5: Average normalized response time ($U$) and average response time ($R$) for the Smallest Memory First (SMF) and the base long-term scheduling policies.

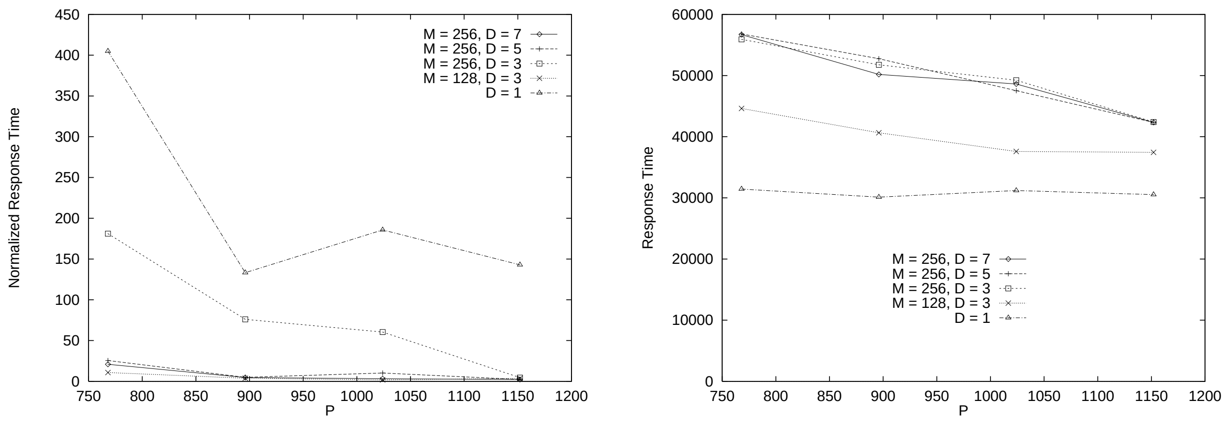


Figure 6: Average normalized response time ($U$) and average response time ($R$) for different values of the degree of multiprogramming ($D$).

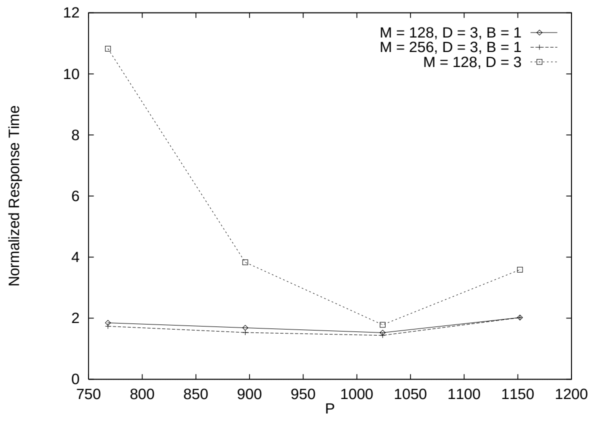
largely independent of the amount of memory installed per processor.

**Impact of Long-term Time-Sharing** The long-term scheduling policies evaluated so far are all non-preemptive in nature, i.e., once a job has been allocated memory it holds on to the memory until it completes. Our results indicate that the number of batch jobs allocated memory on a processor should be restricted to at most one. This implies that there is no fine-grain time-sharing with respect to batch jobs under the policy that was shown to have the best performance for our workload. The statistics in Table 1, however, indicate that there is a great deal of variation in the service times of batch jobs. Thus, time-sharing should lead to improved performance for these jobs [9].

The issue, therefore, is how to time-share the processors (and memory) among batch jobs without letting the degree of multiprogramming of batch jobs exceed one. One solution, clearly, is that the long-term scheduling policy should be preemptive in nature and should support long-term time-sharing among batch jobs.

To assess the performance benefits of long-term time-sharing, we extended the long-term scheduling policy described above so that batch jobs returned to the long-term queue after a time quantum ($T_L$) as in Figure 1. Each job has a priority that is inversely proportional to the CPU service it has accumulated so far. Periodically, the scheduler recalculates the priorities of all the batch jobs in the system (e.g., by dividing them by 2) so that jobs are not penalized forever for past CPU usage.

Figure 8 shows the impact of long-term time-sharing for our workload. Since most of the batch jobs in our workload run for several hours we selected a long-term quantum of 3 hrs. The $T_L = \infty$ policy corresponds to the non-preemptive policy that was shown to have the best performance, i.e, the policy with batch limit, $B$ = 1. Our results show that long-term time-sharing substantially improves the normalized response time for the workload. Most of this reduction in $U$ as compared to the policy with no long-term time-sharing is due to an improvement in the performance of the batch jobs. (Interactive jobs are more or less unaffected by long-term



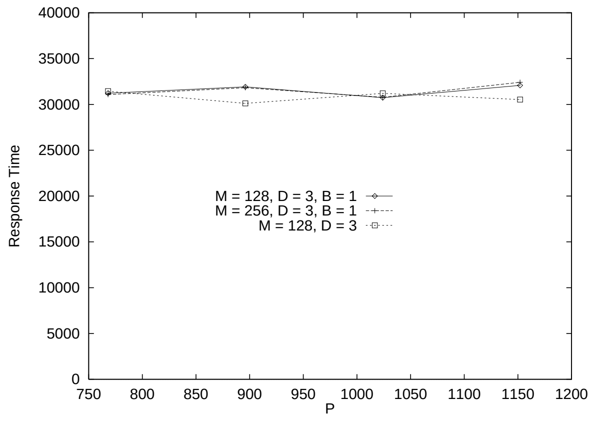
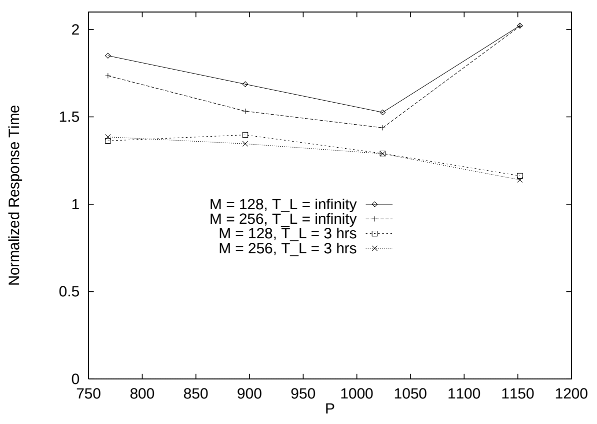


Figure 7: Average normalized response time ($U$) and average response time ($R$) for batch limit ($B$) = 1.



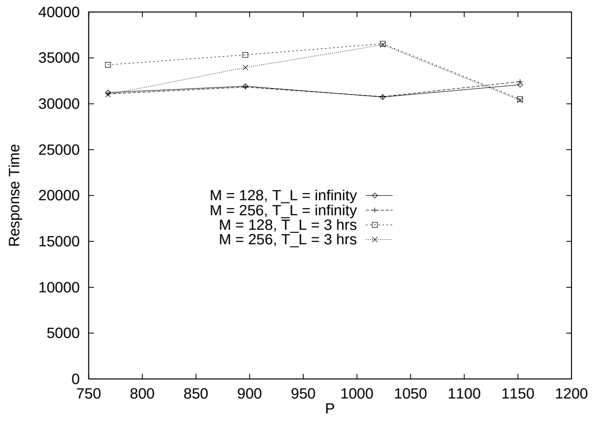


Figure 8: Average normalized response time ($U$) and average response time ($R$) for long-term scheduling policies with and without time-sharing.

time-sharing of batch jobs). The response time under long-term time-sharing is higher than the response time when there is no time-sharing. This result is somewhat misleading and is an artifact of our simulation methodology as explained below.

Overall, our results indicate that long-term time-sharing is beneficial for the workload under consideration.

**Discussion** Each run of our simulation terminates after the last interactive job in the trace completes. At the time the simulation terminates, there can be several batch jobs that have either not received any service at all (in the case of the non-preemptive long term scheduling policy) or have received partial service. At high loads, under the non-preemptive policies, several batch jobs that require a large number of processors and a large amount of memory do not get scheduled during the simulation run. On the other hand, these jobs do receive service when long-term time-sharing is used. We found that the actual offered load on the system under the long-term time-sharing policy was higher than the load under the non-preemptive policy. This is the main reason for the higher response time under the long-term time-sharing policy.

Note that the same reasoning applies to all the results in this paper, especially the results corresponding to $P = 768$. As shown in Table 3, when $P = 768$ the offered load is 0.98. However, none of our response time curves shows the expected increase in response time as the offered load approaches 1 because the actual load on the system is smaller than the offered load. In our future work, we plan to quantify this effect or to modify our simulation methodology to allow a fairer comparison of policies.

## 5 Conclusions

Most previous research on gang-scheduling has ignored the impact of long-term scheduling on the performance of the policy. In large part, this is because there has been very little information available about the memory-usage characteristics of real parallel jobs. In this paper, we reported the memory-usage characteristics of jobs in the production workload on the Cray T3E at the San Diego Supercomputer Center. We also characterized the memory usage of some of the applications that form part of the workload on the LLNL ASCI supercomputer. Our analysis shows that since most jobs have small memory requirements, relatively fine-grain (or short-term) time-slicing among several memory-resident jobs is distinctly possible. Second, since many long-running jobs have large memory requirements, relatively coarse-grain (or long-term) time-sharing is probably necessary for providing good service to these jobs while not penalizing smaller jobs.

This conclusion is confirmed by our simulation study that evaluated the impact of using different long-term scheduling policies on the overall performance of Distributed Hierarchical Control (DHC), a gang-scheduling policy. Our simulation results indicate that for the workload considered in this paper the best long-term scheduling policy restricts the degree of multiprogramming of batch jobs to one and uses coarse-grain time-sharing to provide good service to batch jobs. While our results are based on a single workload, at the very least, they show that the interaction between long-term scheduling and processor scheduling is complex and definitely requires further research.

## Acknowledgements

We would like to thank Allen Downey (Colby College) for providing us with the workload trace from the SDSC Cray T3E. This work would not have been possible without his efforts. Thanks also to Morris Jette of LLNL for providing us with data on the characteristics of the ASCI workload and allowing us to report that information in this paper.

## References

[1] D. Burger, R. Hyder, B. Miller, and D. Wood. Paging tradeoffs in distributed shared-memory multiprocessors. In *Proc. of Supercomputing '94*. IEEE, Nov. 1994.

[2] Rohit Chandra, Scott Devine, Ben Verghese, Mendel Rosenblum, and Anoop Gupta. Scheduling and page migration for multiprocessor compute servers. In *Proc. of ASPLOS-VI*, pages 12–24. ACM, Oct. 1994.

[3] D. Feitelson. Packing Schemes for Gang Scheduling. In *IPPS Job Scheduling Workshop*, Apr. 1996.

[4] D. G. Feitelson and L. Rudolph. Distributed Hierarchical Control for Parallel Processing. *Computer*, May 1990.

[5] Dror Feitelson. Memory Usage in the LANL CM-5 Workload. In *Job Scheduling Strategies for Parallel Procesing*, volume 1291 of *Lecture Notes in Computer Science*, pages 78–94. Springer-Verlag, 1997.

[6] N. Islam, A. Prodromidis, M. Squillante, A. Gopal, and L. Fong. Extensible resource scheduling for parallel computers. In *Proc. of 8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[7] M. Jette, 1998. Personal Communication.

[8] M. Jette, D. Storch, and E. Yin. Timesharing the Cray T3D. In *Cray User Group*, pages 247–252, Mar. 1996.

[9] L. Kleinrock. *Queueing Systems*, volume 2. John Wiley, 1976.

[10] S. Leutenegger and M. Vernon. The Performance of Multiprogrammed Multiprocessor Scheduling Policies. In *Proc. of Sigmetrics '90*, May 1990.

[11] Cathy McCann and John Zahorjan. Scheduling memory constrained jobs on distributed memory parallel computers. In *Proc. of 1995 ACM Sigmetrics Conference*, pages 208–219, May 1995.

[12] J. K. Ousterhout. Scheduling Techniques for Concurrent Systems. In *Proc. of the Third International Conference on Distributed Computing Systems*, pages 22 – 30, Oct. 1982.

[13] Eric Parsons and Ken Sevcik. Coordinated Allocation of Memory and Processors in multiprocessors. In *Proc. of ACM Sigmetrics/Performance '96*, pages 57–67, May 1996.

[14] Vinod G. J. Peris, Mark S. Squillante, and Vijay K. Naik. Analysis of the impact of memory in distributed parallel processing systems. In *Proc. of 1994 ACM Sigmetrics Conference*, pages 5–18, Nashville, May 1994.

[15] Sanjeev Setia. The Interaction between Memory Allocation and Adaptive Partitioning in Message-passing Multicomputers. In *IPPS Job Scheduling Workshop*, Apr. 1995.

[16] Sanjeev Setia. Trace-driven Analysis of Migration-based Gang-scheduling Policies for Parallel Computers. In *Proc. of ICPP '97*, Aug. 1997.

[17] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors. In *Proc. of the Twelfth ACM Symposium on Operating Systems Principles*, pages 159–166, Dec. 1989.

[18] Fang Wang, Marios Papaefthymiou, and Mark S. Squillante. Performance evaluation of gang scheduling for parallel and distributed multiprogramming. In *Job Scheduling Strategies for Parallel Processing,* D. G. Feitelson and L. Rudolph (eds.), pages 277–298. Springer-Verlag, 1997.

[19] J. Zahorjan and C. McCann. Processor Scheduling in shared Memory Multiprocessors. In *Proc. of ACM SIGMETRICS Conf.*, 1990.