Scheduling Algorithms to Improve Utilization in Toroidal-Interconnected Systems

by

Elie Krevat

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified by .....
José E. Moreira
Research Staff Member, IBM T.J. Watson Research Center
Thesis Supervisor

Certified by .....
Madhu Sudan
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Scheduling Algorithms to Improve Utilization in Toroidal-Interconnected Systems

by

Elie Krevat

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

BlueGene/L is a massively parallel cellular architecture system with a toroidal interconnect, currently being developed at the IBM T.J. Watson Research Center. Cellular architectures with a toroidal interconnect are effective at producing highly scalable computing systems, but typically require job partitions to be both rectangular and contiguous. These restrictions introduce fragmentation issues that affect the utilization of the system and the wait time and slowdown of queued jobs. To solve these fragmentation problems, this thesis presents the analysis and application of scheduling algorithms that augment a baseline first come first serve (FCFS) scheduler. Restricting ourselves to space-sharing techniques, which constitute a simpler solution to the requirements of cellular computing, we present simulation results for migration and backfilling techniques on BlueGene/L. These techniques are explored individually and jointly to determine their impact on the system. We develop an efficient Projection Of Partitions (POP) algorithm for determining the size of the largest free rectangular partition in a toroidal system, a basic operation that is the computational bottleneck for our scheduling algorithms. Our results demonstrate that migration may be effective for a pure FCFS scheduler but that backfilling produces even more benefits. We also show that migration may be combined with backfilling to produce more opportunities to better utilize a parallel machine.

Thesis Supervisor: José E. Moreira
Title: Research Staff Member, IBM T.J. Watson Research Center

Thesis Supervisor: Madhu Sudan
Title: Associate Professor

Acknowledgments

The work presented in this thesis is the largest research project which I undertook during my cooperative internship at IBM T.J. Watson Research Center. A research note containing the major results of my work was published in Euro-Par [3], while a more complete version was published in the 8th Workshop on Job Scheduling Strategies for Parallel Processing, held in conjunction with HPDC11 [14].

I am indebted to many people at IBM for their support during my research. First and foremost, José Moreira, my manager, mentor, and friend. José is far and away the best manager anyone could hope for. From the daily meetings at the start of the project, to making himself available even during one of his many incredibly busy days, to traveling with me to Edinburgh so I could present our research - José helped shape my research direction through his guidance. José Castaños also provided important feedback and ideas, specifically for the Projection of Partitions algorithm. He is also a treasure chest of technical information. Gheorghe Almasi, thanks for the working with me on the MPI research - perhaps one day we will also publish something together. I must also thank Professor Madhu Sudan for agreeing to be my MIT thesis supervisor and spending the time to go over my work, even though the thesis topic had little connection to coding theory and probabilistically checkable proofs!

Then there are the people at IBM who entertained me during breaks and after work, whose camaraderie and support were always welcome. Joseph Gagliano missed my first summer at IBM, but he was the one constant factor during the rest of my time, even during the changing of the Brazilian mafia - thanks for the golf and the sushi. Manu Sridharan was always great to be around, whether we were exploring the cultural offerings of New York City or deliberating the merits of sketch comedy. The entity known only as "Luis and Karin", your scripts let me batch hundreds of simulations at a time on the cluster, but how can I forget the pizza and chip runs. José Brunheroto, thanks for the cheerful stories and the coffee breaks. Then there's Chris Erway for the goofy insights, Pedro and Bapi for teaching me the essentials of Portuguese, Mike Arnold for the humorous poetry and letting me dunk over you on the basketball courts, Ann Fornof for the ultimate frisbee and the friendly face in the corridor, and Martín Casado for sparking my interest in systems, networks, and security, designing the perfect CryptoSphere, the juggling, and always being around that first summer for a good laugh. While I do not have space to thank them all individually, I am grateful to the many other interns and co-workers who have made IBM such a great place to work.

Finally, I cannot conclude these acknowledgements without thanking my family. Dad, Mom, Ariela and Rina - thanks for being you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

BlueGene/L (BG/L) is a massively parallel cellular architecture system. Each node consists of two PowerPC® embedded cores using system-on-a-chip technology, and 256 MB of DRAM. 65,536 of these self-contained computing nodes, or *cells*, are interconnected in a highly dense three-dimensional toroidal pattern [21]. In that pattern, each cell is directly connected to its six nearest neighbors, two each along the x, y, and z axes. Three-dimensional toroidal interconnects are simple, modular, and scalable, particularly when compared with systems that have a separate, typically multistage, interconnection network [15]. Examples of successful toroidal-interconnected parallel systems include the Cray T3D and T3E machines [12].

There is, however, a price to pay with toroidal interconnects. We cannot view the system as a simple fully-connected interconnection network of nodes that are equidistant to each other (*i.e.*, a flat network). In particular, we lose an important feature of systems like the IBM RS/6000 SP, which lets us pick any set of nodes for execution of a parallel job, irrespective of their physical location in the machine [1]. In a toroidal-interconnected system, the spatial allocation of nodes to jobs is of critical importance. In most toroidal systems, including BG/L, job partitions must be both rectangular (in a multidimensional sense) and contiguous. It has been shown by Feitelson and Jette [8] that, because of these restrictions, significant machine fragmentation occurs in a toroidal system. Fragmentation results in low system utilization and high wait time for queued jobs.

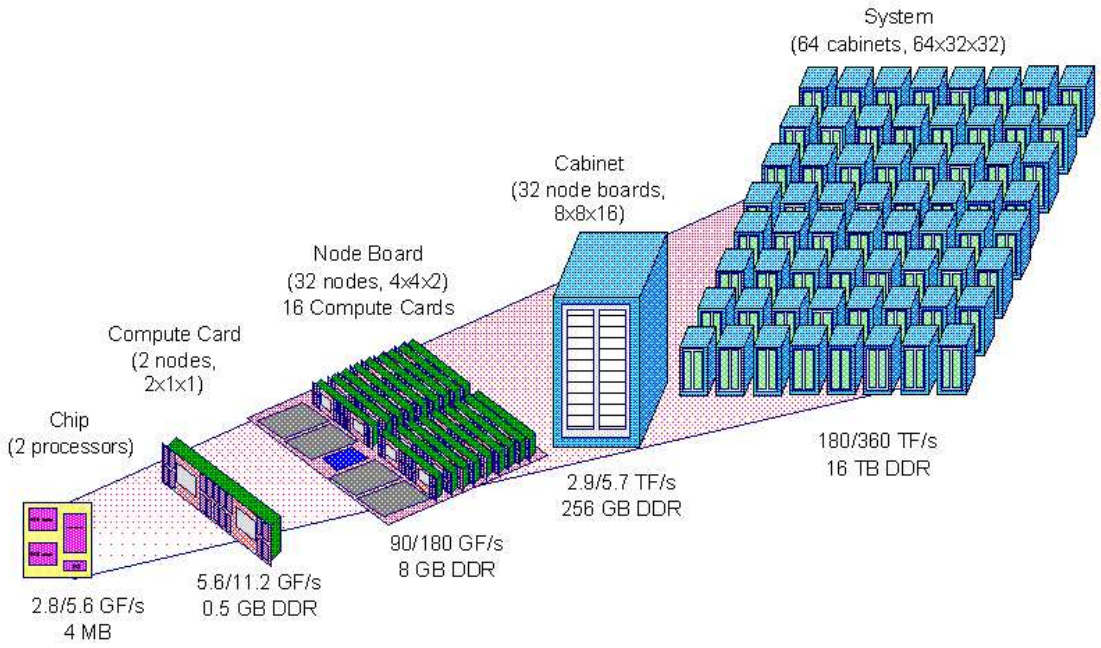


Figure 1-1: BlueGene/L architecture, with expected speed when utilizing 1 or 2 processors on each chip for computation

We may limit the system fragmentation in a few ways. By restricting the sizes of jobs that can be initialized (e.g., only powers of 2), jobs can pack more efficiently in the torus. However, this method does not eliminate fragmentation and such restrictions need not be imposed on the end user. By employing more space-efficient scheduling algorithms, we can have an even greater impact on fragmentation and system utilization. These scheduling algorithms generally fall into two categories: time-sharing and space-sharing techniques. Time-sharing techniques, such as gang-scheduling, have been shown to be very effective at reducing fragmentation on the Cray T3D [8]. However, these types of schedulers require more memory and operating system involvement than is practically available in a cellular computing environment, where processors are dedicated to only one active job at a time. Space-sharing techniques, which are better suited to the goals of cellular computing, have been shown to be effective on other architectures that do not have a toroidal interconnect [9, 16, 19].

In this thesis, we analyze a set of strictly space-sharing scheduling techniques to improve system utilization and reduce the wait time of jobs for BG/L. We analyze

the two techniques of backfilling [9, 16, 19] and migration [4, 22] in the context of a toroidal-interconnected system. Backfilling is a technique that moves lower priority jobs ahead of other higher priority jobs, as long as execution of the higher priority jobs is not delayed. Migration moves jobs around the toroidal machine, performing on-the-fly defragmentation to create larger contiguous free space for waiting jobs.

We conduct a simulation-based study of the impact of our scheduling algorithms on the system performance of BG/L. Using actual job logs from supercomputing centers, we measure the impact of migration and backfilling as enhancements to a first-come first-serve (FCFS) job scheduling policy. Migration is shown to be effective in improving maximum system utilization while enforcing a strict FCFS policy. We also find that backfilling, which bypasses the FCFS order, can lead to even higher utilization and lower wait times. Finally, we show that there is a small benefit from combining backfilling and migration.

While experimenting with these scheduling algorithms, we also developed an efficient new algorithm to determine the size of the largest available rectangular partition in a given three-dimensional torus. This algorithm computes projections of adjacent partitions and uses dynamic programming techniques. Since this basic operation is the computational bottleneck for all of our schedulers, implementing this algorithm reduced the average computation time required to schedule a job by a factor of 2 to 4 times.

The rest of this thesis is organized as follows. Chapter 2 discusses the scheduling algorithms used to improve job scheduling on a toroidal-interconnected parallel system. Chapter 3 presents the Projection Of Partitions (POP) algorithm for determining the size of the largest free partition in a torus. Chapter 4 describes the simulation procedure to evaluate these algorithms. Chapter 5 presents our simulation results. Chapter 6 describes related work and suggests future work opportunities. Finally, Chapter 7 presents the conclusions.

Chapter 2

Scheduling Algorithms

System utilization and average job wait time in a parallel system may be improved through better job scheduling algorithms [5, 6, 8, 10, 11, 13, 16, 17, 18, 19, 23, 24, 28]. The opportunity for improvement over a simple first-come first-serve (FCFS) scheduler is much greater for toroidal interconnected systems because of the fragmentation issues discussed in Chapter 1. The following chapter describes four job scheduling algorithms that we evaluate in the context of BG/L. In all algorithms, arriving jobs are first placed in a queue of waiting jobs, prioritized according to the order of arrival. The scheduler is invoked for every job arrival and job termination event in order to schedule new jobs for execution.

The first algorithm is a straightforward FCFS scheduler that serves as a base reference. The second scheduler uses backfilling to bypass higher-priority jobs in the queue. The third scheduler maintains strict FCFS order, but migrates jobs around the torus to free up larger contiguous blocks of space so that more jobs may begin execution. Finally, the fourth and last scheduler uses a combination of migration and backfilling to further improve system performance.

Scheduler 1: First Come First Serve (FCFS). For FCFS, we adopt the heuristic of traversing the waiting queue in order and scheduling each job in a way that maximizes the largest free rectangular partition remaining in the torus. This heuristic will make local greedy decisions based only on the size of each job in an attempt to

maximize the chances of scheduling the next job, prioritized by arrival time, in the waiting queue. For each job of size p, we try all the possible rectangular shapes of size p that fit in the torus. For each shape, we try all the legal allocations in the torus that do not conflict with running jobs. Finally, we select the shape and allocation that results in the maximal free rectangular partition remaining after allocation of this job. We stop when we find the first job in the queue that cannot be scheduled.

A valid rectangular partition does not always exist for a job. There are job sizes which are always impossible for the torus, such as prime numbers greater than the largest dimension size. Because job sizes are known at job arrival time, before execution, jobs with impossible sizes are modified to request the next largest possible size. Additionally, there are legal job sizes that cannot be scheduled because of the current state of the torus. Therefore, if a particular job of size p cannot be scheduled, but some free partition of size $q > p$ exists, the job will be increased in size by the minimum amount required to schedule it. For example, consider a 4×4 (two-dimensional) torus with a single free partition of size 2×2. If a user submits a job requesting 3 nodes, that job cannot be run. The scheduler increases the job size by one, to 4, and successfully schedules the job.

An FCFS scheduler that searches the torus in a predictable incremental fashion, implements the maximal partition heuristic, and modifies job sizes when necessary is the simplest algorithm considered, against which more sophisticated algorithms are compared. Determining the size of the largest rectangular partition in a given three-dimensional torus is the most time-intensive operation required to implement the maximal partition heuristic. When considering a torus of shape $M \times M \times M$, a straightforward exhaustive search of all possible partitions takes $O(M^9)$ time. We have developed a more efficient algorithm that computes incremental projections of planes and uses dynamic programming techniques. This projection algorithm has complexity $O(M^5)$ and is described in Chapter 3.

Scheduler 2: FCFS With Backfilling. Backfilling is a space-sharing optimization technique that bypasses the priority order imposed by the job queuing policy. A

backfilling policy estimates the start time of the highest priority job j in the waiting queue and uses this estimate as a *reservation time* for that job. The reservation time guarantees that the job will be scheduled no later than that time, and this schedule time may improve if some jobs complete earlier than expected. Then, a *weak backfilling* algorithm will allow a lower priority job i to be scheduled before j as long as the reservation time of j is not delayed. A *strong backfilling* algorithm will make a reservation time for all jobs in the queue and will only schedule a lower priority job i out of order if *none* of the reservation times of jobs with higher priority than i are delayed. We use a weak backfilling strategy that sacrifices a small amount of fairness for a much larger expected gain in utilization. A weak backfilling strategy is the minimum required to ensure that no job starvation will occur, since a reservation time is always made for the highest priority job. However, it is possible that the execution of lower priority jobs will be delayed. Backfilling is used in conjunction with the FCFS scheduler and is only invoked when there are jobs in the waiting queue and FCFS halts because it could not schedule the highest priority job.

The effect of backfilling a particular workload for a one-dimensional machine is depicted in Figure 2-1. Suppose we are asked to schedule five jobs, numbered from 1 to 5 in order of arrival. Figure 2-1(a) shows the schedule that would be produced by a FCFS policy without backfilling. Note the empty space between times T_1 and T_2, while job 3 waits for job 2 to finish. Figure 2-1(b) shows the schedule that would be produced by a FCFS policy with backfilling. The empty space was filled with job 5, which may be executed before job 3 without delaying it.

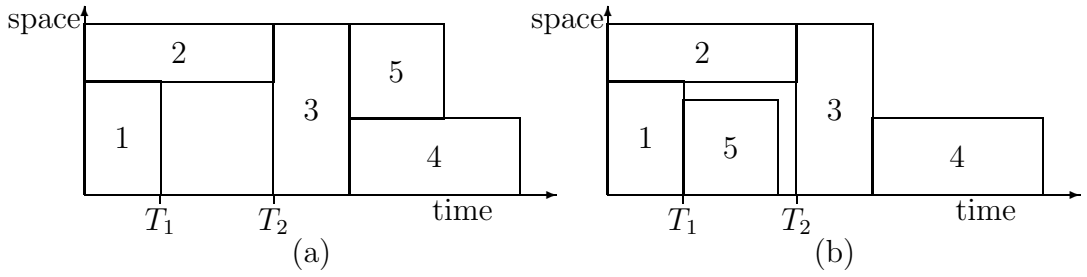


Figure 2-1: FCFS policy without (a) and with (b) backfilling. Job numbers correspond to their position in the priority queue.

The backfilling algorithm requires an estimation of job execution times, which is usually not very accurate. However, previous work [9, 20, 25] has shown that overestimating execution times does not significantly affect backfilling results. This algorithm is most effective when the waiting job queue contains many small jobs of short execution time interspersed among the larger and more time-intensive jobs. Backfilling has been shown to increase system utilization in a fair manner on the IBM RS/6000 SP [9, 25].

Just as the FCFS scheduler dynamically increases the size of jobs that cannot be scheduled with their current size, similar situations may arise during backfilling. Unlike FCFS, however, the size increase is performed more conservatively during backfilling because there are other jobs in the queue which may better utilize the free nodes of the torus. Therefore, a parameter I specifies the maximum size by which the scheduler will increase a job. For example, by setting $I = 1$ (our default value), backfilling increases a job size by at most one node. This parameter is used only during the backfilling phase of scheduling; the FCFS phase will always increase the size of the first job in the queue to successfully schedule it.

Scheduler 3: FCFS With Migration. The migration algorithm rearranges the running jobs in the torus in order to increase the size of the maximal contiguous rectangular free partition. Migration in a toroidal-interconnected system compacts the running jobs and counteracts the effects of fragmentation. Additionally, migration does not sacrifice any amount of fairness when achieving higher utilization levels.

While migration does not require any more information than FCFS to execute, it may require additional hardware and software functionality. This paper does not attempt to quantify the overhead of that functionality. However, accepting that this overhead exists, migration is only undertaken when the expected benefits are deemed substantial. The decision to migrate is therefore based on two parameters: FN_{tor}, the ratio of free nodes in the system compared to the size of the torus, and FN_{max}, the fraction of free nodes contained in the maximal free partition. In order for migration to establish a significantly larger maximal free partition, FN_{tor} must be sufficiently

high and FN_{max} must be sufficiently low. Chapter 5 contains further analysis of these parameters.

The migration process is undertaken immediately after the FCFS phase fails to schedule a job in the waiting queue. Jobs already running in the torus are organized in a queue of migrating jobs sorted by size, from largest to smallest. Each job is then reassigned a new partition, using the same algorithm as FCFS and starting with an empty torus. After migration, FCFS is performed again in an attempt to start more jobs in the rearranged torus.

In order to ensure that all jobs fit in the torus after migration, job sizes are not increased if a reassignment requires a larger size to fit in the torus. Instead, the job is removed from the queue of migrating jobs, remaining in its original partition, and reassignment begins again for all remaining jobs in the queue. If the maximal free partition size after migration is worse than the original assignment, which is possible but generally infrequent under the current scheduling heuristics, migration is not performed.

Scheduler 4: FCFS with Backfilling and Migration. Backfilling and migration are independent scheduling concepts and a scheduler may implement both of these functions simultaneously. First, we schedule as many jobs as possible via FCFS. Next, we rearrange the torus through migration to minimize fragmentation and then repeat FCFS. Finally, the backfilling algorithm from Scheduler 2 is performed to make a reservation time for the highest-priority job and attempt to schedule jobs with lower priority so long as they do not conflict with the reservation. The combination of these policies should lead to an even more efficient utilization of the torus. For simplicity, we call this scheduling technique, that combines backfilling and migration, B+M.

Chapter 3

Projection of Partitions (POP) Algorithm

In a given three-dimensional torus of shape $M \times M \times M$ where some nodes have been allocated for jobs, the POP algorithm provides a $O(M^5)$ time algorithm for determining the size of the largest free rectangular partition. This algorithm is a substantial improvement over an exhaustive search algorithm that takes $O(M^9)$ time.

Let FREEPART $= \{\langle B, S\rangle \mid B$ is a base location (i, j, k) and S is a partition size (a, b, c) such that $\forall\ x, y, z,\ i \leq x < (i + a),\ j \leq y < (j + b),\ k \leq z < (k + c)$, node ($x$ mod M, y mod M, z mod M) is free$\}$. POP narrows the scope of the problem by determining the largest rectangular partition $P \in$ FREEPART rooted at each of the M^3 possible base locations and then deriving a global maximum. Given a base location, POP works by finding the largest partition first in one dimension, then by projecting adjacent one-dimensional columns onto each other to find the largest partition in two dimensions, and iteratively projecting adjacent two-dimensional planes onto each other to find the largest partition in three dimensions.

First, a partition table of the largest one-dimensional partitions $P \in$ FREEPART is pre-computed for all three dimensions and at every possible base location in $O(M^4)$ time. This is done by iterating through each partition and whenever an allocated node is reached, all entries for the current "row" may be filled in from a counter value, where the counter is incremented for each adjacent free node and reset to zero whenever an

additional allocated node is reached.

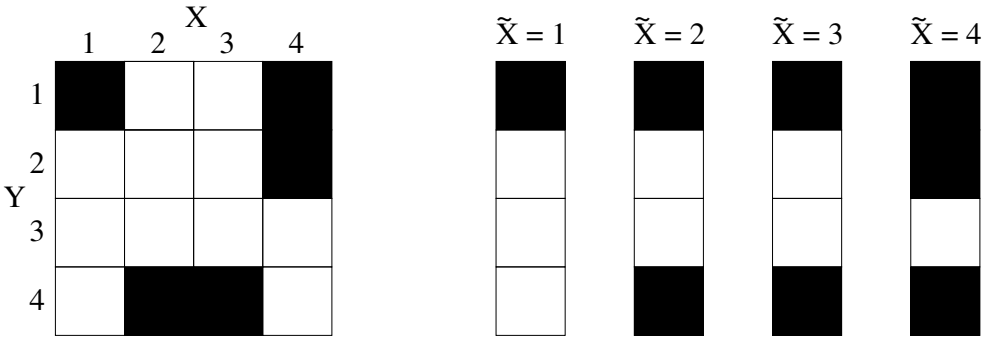


Figure 3-1: 2-dimensional POP Algorithm applied to Base Location (1,2): Adjacent 1-dimensional columns are projected onto each other as $\tilde{\text{X}}$ is incremented.

For a given base location (i, j, k), we fix one dimension (e.g., k), start a counter $\tilde{\text{X}} = 1$ in the next dimension, and multiply $\tilde{\text{X}}$ by the minimum partition table entry of the third dimension for (x mod M, j, k), where x varies as $i \leq x \leq (i+ \tilde{\text{X}} -1)$ and $\tilde{\text{X}}$ varies as $1 \leq \tilde{\text{X}} \leq M$. As the example in Figure 3-1 shows, when $\tilde{\text{X}} = 1$ for some fixed k at base location $(1, 2, k)$, the partition table entry in the Y dimension will equal 3 since there are 3 consecutive free nodes, and our largest possible partition size is initially set to $(\tilde{\text{X}}\times 3) = 3$. When $\tilde{\text{X}}$ increases to 2, the minimum table entry becomes 2 because of the allocated node at location $(2, 4, k)$ and the largest possible partition size is increased to $(\tilde{\text{X}}\times 2) = 4$. When $\tilde{\text{X}} = 3$, we calculate a new largest possible partition size of $(\tilde{\text{X}}\times 2) = 6$. Finally, when we come across a partition table entry in the Y dimension of 0, because of the allocated node at location $(4, 2, k)$, we stop incrementing $\tilde{\text{X}}$.

This same idea is extended to work for 3 dimensions. Given a similar base location (i, j, k), we start a counter $\tilde{\text{Z}}$ in the Z dimension and calculate the maximum two-dimensional partition. Then we project adjacent two-dimensional planes by incrementing $\tilde{\text{Z}}$ and forming a new minimum partition table of the X and Y dimensions for (i, j, z mod M), where z varies as $k \leq z \leq (k+ \tilde{\text{Z}} -1)$ and $\tilde{\text{Z}}$ varies as $1 \leq \tilde{\text{Z}} \leq M$. Then, we calculate the largest two-dimensional partition using the projected minimum partition table.

Using the initial partition table, it takes $O(M)$ time to calculate a projection for

two adjacent planes and to determine the largest two-dimensional partition. Since there are $O(M)$ projections required for $O(M^3)$ base locations, our final algorithm runs in $O(M^5)$ time.

When we implemented this algorithm in our scheduling simulator, we achieved a significant speed improvement. For the original NASA log, scheduling time for our B+M scheduler improved from an average of 0.51 seconds for every successfully scheduled job to 0.16 seconds, while the SDSC log improved from an average of 0.125 seconds to 0.063 seconds. The longest time to successfully schedule a job also improved from 38 seconds to 8.3 seconds in the NASA log, and from 50 seconds to 8.5 seconds in the SDSC log.

Chapter 4

Experiments

We use a simulation-based approach to perform quantitative measurements of the efficiency of the proposed scheduling algorithms. An event-driven simulator was developed to process actual job logs from supercomputing centers. The results of simulations for all four schedulers were then studied to determine the impact of their respective algorithms.

We begin this chapter with a short overview of our scheduling abstractions for the BG/L system. We then describe our simulation environment and conclude with a discussion of the workload characteristics for the two simulated job logs.

4.1 BlueGene/L Scheduling Abstractions

The BG/L system is organized as a $32 \times 32 \times 64$ three-dimensional torus of nodes. Each node contains processors, memory, and links for interconnecting to its six neighbors, with full hardware routing capability. The unit of allocation for job execution in BG/L is a 512-node ensemble organized in an $8 \times 8 \times 8$ configuration. This allocation unit is the smallest granularity whereby any contiguous rectangular partition of these units can be electrically partitioned into a toroidal topology. Electrical partitioning isolates a job and prevents other jobs from interfering with its communication patterns. Therefore, BG/L behaves as a $4 \times 4 \times 8$ torus of these *supernodes*. We use this supernode abstraction when scheduling jobs.

4.2 The Simulation Environment

The simulation environment models a torus of 128 (super)nodes in a three-dimensional $4 \times 4 \times 8$ configuration. The event-driven simulator receives as input a job log and the type of scheduler to simulate (FCFS, Backfill, Migration, or B+M). There are four primary events in the simulator:

- An *arrival event* occurs when a job is first submitted for execution and placed in the scheduler's waiting queue
- A *schedule event* occurs when a job is allocated on the torus
- A *start event* occurs after a standard delay of one second following a schedule event, at which time a job begins to run
- A *complete event* occurs upon completion of a job, at which point the job is deallocated from the torus. The scheduler is invoked at the conclusion of every event that affects the states of the torus or the waiting queue (*i.e.*, the arrival and complete events).

A job log contains information on the arrival time, execution time, and size of all jobs. Given a torus of size N, and for each job j the arrival time t_j^a, execution time t_j^e and size s_j, the simulation produces values for the start time t_j^s and finish time t_j^f of each job. These results are analyzed to determine the following parameters for each job:

- wait time $t_j^w = t_j^s - t_j^a$
- response time $t_j^r = t_j^f - t_j^a$
- bounded slowdown $t_j^{bs} = \frac{\max(t_j^r, \Gamma)}{max(t_j^e, \Gamma)}$ for $\Gamma = 10$ seconds. The Γ term appears according to recommendations in [9], because some jobs have very short execution time, which may distort the slowdown.

Global system statistics are also determined. Let the simulation time span be $T = \max_{\forall j}(t_j^f) - \min_{\forall k}(t_k^a)$. We then define system utilization (also called *capacity utilized*) as

$$w_{\text{util}} = \sum_{\forall j} \frac{s_j t_j^e}{TN}. \tag{4.1}$$

Similarly, let $f(t)$ denote the number of free nodes in the torus at time t and $q(t)$ denote the total number of nodes requested by jobs in the waiting queue at time t. Then, the total amount of unused capacity in the system, w_{unused}, is defined as:

$$w_{\text{unused}} = \int_{\min(t_j^a)}^{\max(t_j^f)} \frac{\max(0, f(t) - q(t))}{TN} dt. \tag{4.2}$$

This parameter is a measure of the work unused by the system because there is a lack of jobs requesting free nodes. The max term is included because the amount of unused work cannot be less than zero. The balance of the system capacity is lost despite the presence of jobs that could have used it. The measure of lost capacity in the system, which includes capacity lost because of the inability to schedule jobs and the delay before a scheduled job begins, is then derived as:

$$w_{\text{lost}} = 1 - w_{\text{util}} - w_{\text{unused}} \tag{4.3}$$

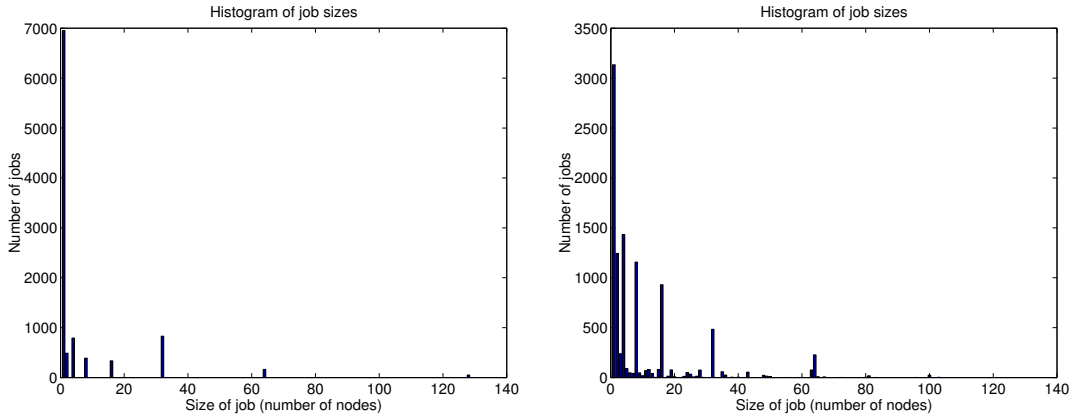
4.3 Workload characteristics

We performed experiments on a 10,000-job span of two job logs obtained from the *Parallel Workloads Archive* [7]. The first log is from NASA Ames's 128-node iPSC/860 machine (from the year 1993). The second log is from the San Diego Supercomputer Center's (SDSC) 128-node IBM RS/6000 SP (from the years 1998-2000). For our purposes, we will treat each node in those two systems as representing one supernode (512-node unit) of BG/L. This is equivalent to scaling all job sizes in the log by 512, which is the ratio of the number of nodes in BG/L to the number of nodes in these 128-node machines. Table 4.1 presents the workload statistics and Figure 4-1 summarizes the distribution of job sizes and the contribution of each job size to the

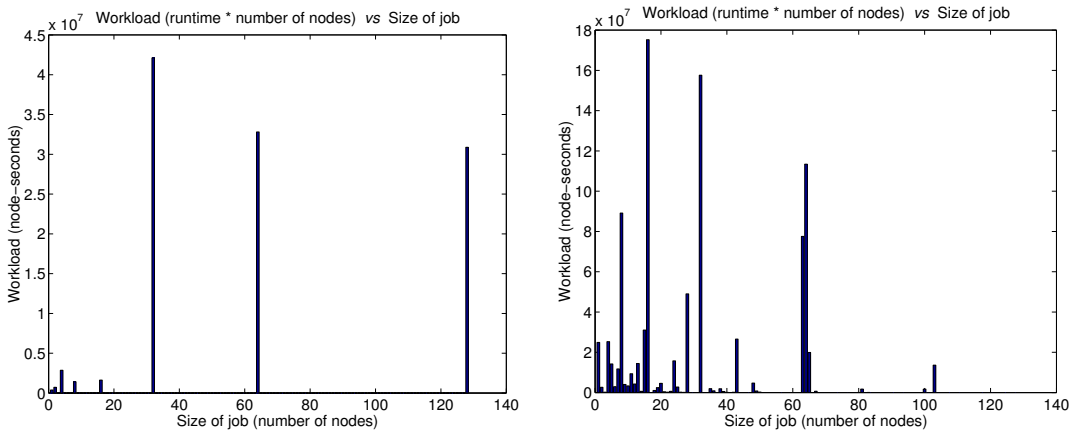
total workload of the system. Using these two logs as a basis, we generate logs of varying workloads by multiplying the execution time of each job by a coefficient c, mostly varying c from 0.7 to 1.4 in increments of 0.05. Simulations are performed for all scheduler types on each of the logs. With these modified logs, we plot wait time and bounded slowdown as a function of system utilization.

Table 4.1: Statistics for 10,000-job NASA and SDSC logs.

	NASA Ames iPSC/860 log	*SDSC IBM RS/6000 SP log*
Number of nodes:	128	128
Job size restrictions:	powers of 2	none
Job size (nodes)		
Mean:	6.3	9.7
Standard deviation:	14.4	14.8
Workload(node-seconds)		
Mean:	0.881×10^6	7.1×10^6
Standard deviation:	5.41×10^6	25.5×10^6



(a) NASA Ames iPSC/860 (b) SDSC RS/6000 SP



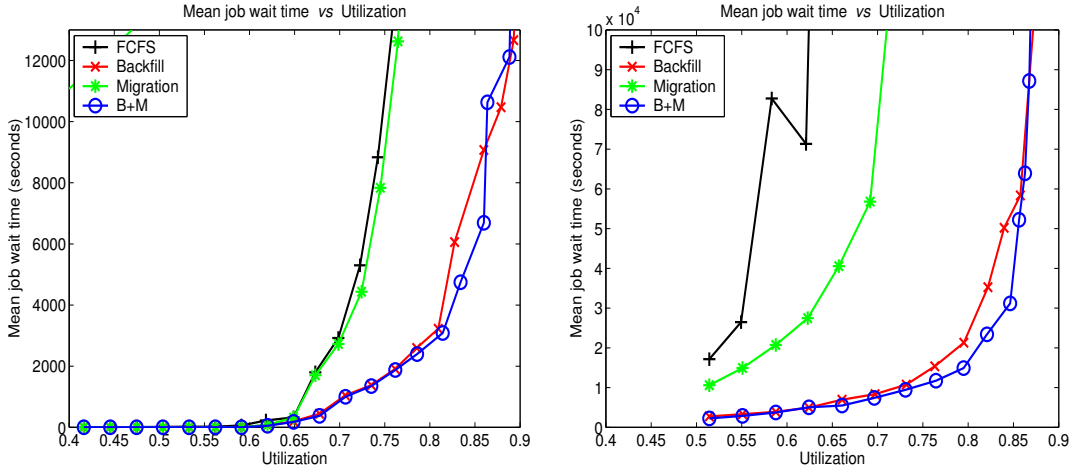
(c) NASA Ames iPSC/860 (d) SDSC RS/6000 SP

Figure 4-1: Job sizes and total workload for NASA Ames iPSC/860((a) and (c)) and San Diego Supercomputer Center (SDSC) IBM RS/6000 SP((b) and (d)).

Chapter 5

Simulation Results

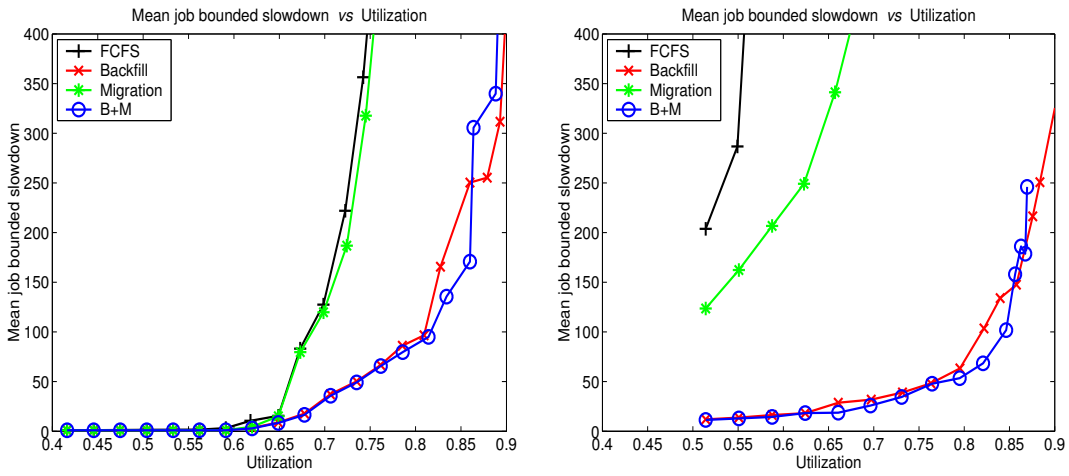
Figures 5-1 and 5-2 present plots of average job wait time (t_j^w) and average job bounded slowdown (t_j^{bs}), respectively, *vs* system utilization (w_{util}) for each of the four schedulers considered and each of the two job logs. We observe that the overall shapes of the curves for wait time and bounded slowdown are similar.



(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-1: Mean job wait time *vs* utilization for (a) NASA and (b) SDSC logs.

The most significant performance improvement is attained through backfilling, for both the NASA and SDSC logs. Also, for both logs, there is a certain benefit from migration, whether combined with backfilling or not. We analyze the results from each log separately.



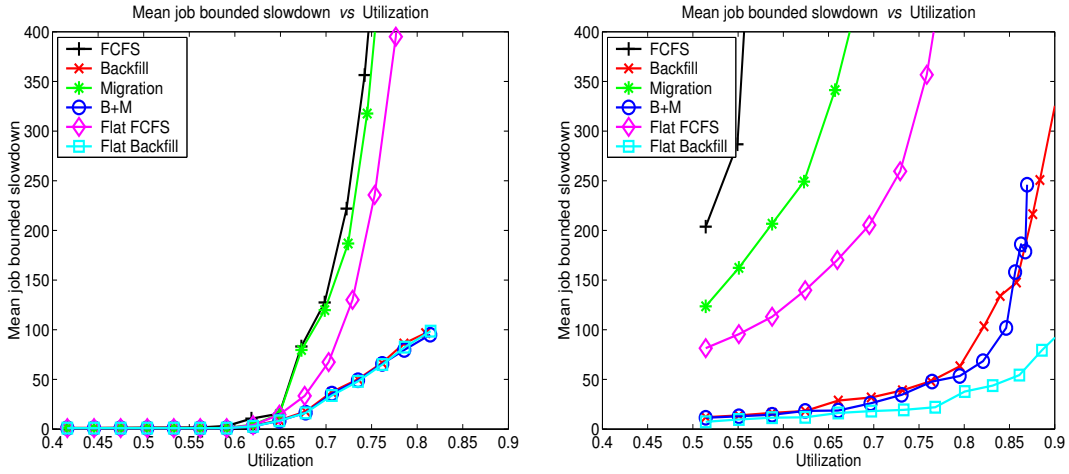
(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-2: Mean job bounded slowdown *vs* utilization for (a) NASA and (b) SDSC logs.

NASA log: All four schedulers provide similar average job wait time and average job bounded slowdown for utilizations up to 65%. The FCFS scheduler saturates at about 77% utilization, whereas the Migration scheduler saturates at about 80% utilization. Backfilling (with or without migration) allows utilizations above 80% and saturates closer to 90% (the saturation region for these schedulers is shown here by plotting values of $c > 1.4$). We note that migration provides only a small improvement in wait time and bounded slowdown for most of the utilization range, and the additional benefits of migration with backfilling becomes unpredictable for utilization values close to the saturation region. In the NASA log, all jobs are of sizes that are powers of two, which results in a good packing of the torus. Therefore, the benefits of migration are limited.

SDSC log: With the SDSC log, the FCFS scheduler saturates at 63%, while the stand-alone Migration scheduler saturates at 73%. In this log, with jobs of more varied sizes, fragmentation occurs more frequently. Therefore, migration has a much bigger impact on FCFS, significantly improving the range of utilizations at which the system can operate. However, we note that when backfilling is used there is again only a small additional benefit from migration, more noticeable for utilizations between 75 and 85%. Utilization above 85% can be achieved, but only with exponentially

growing wait time and bounded slowdown, independent of performing migrations.



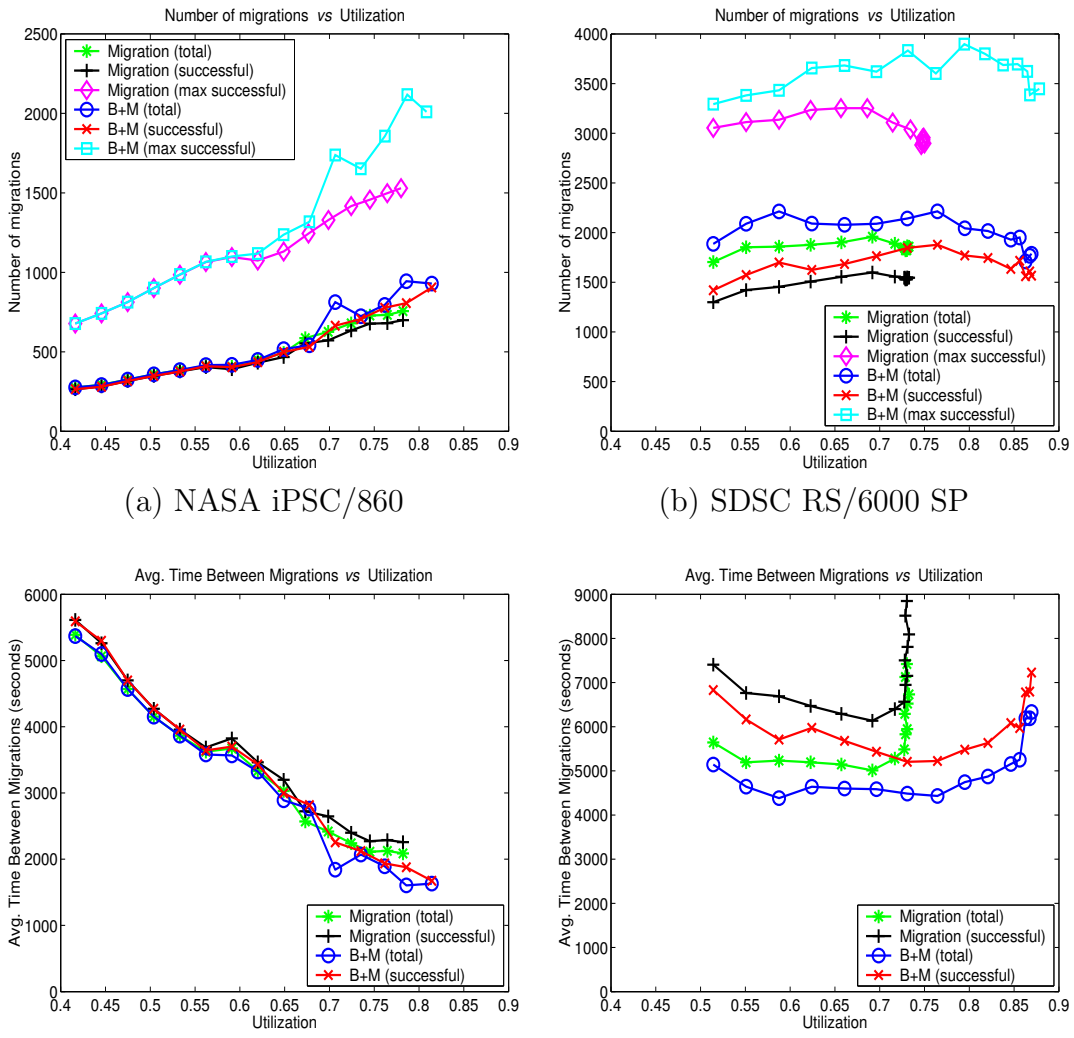
(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-3: Mean job bounded slowdown *vs* utilization for the NASA and SDSC logs, comparing toroidal and flat machines.

Figure 5-3 presents a plot of average job bounded slowdown (t_j^{bs}) *vs* system utilization (w_{util}) for each of the four schedulers considered and each of the two job logs. We also include results from the simulation of a fully-connected (*flat*) machine, with and without backfilling. Since a fully-connected machine does not suffer from fragmentation, this allows us to assess the effectiveness of our schedulers in overcoming the difficulties imposed by a toroidal interconnect. The overall shapes of the curves for wait time are similar to those for bounded slowdown.

Migration by itself cannot make the results for a toroidal machine as good as those for a fully connected machine. For the SDSC log, in particular, a fully connected machine saturates at about 80% utilization with just the FCFS scheduler. For the NASA log, results for backfilling with or without migration in the toroidal machine are just as good as the backfilling results in the fully connected machine. For utilizations above 85% in the SDSC log, not even a combination of backfilling and migration will perform as well as backfilling on a fully connected machine.

Figure 5-4 plots the number of migrations performed and the average time between migrations *vs* system utilization for both workloads. We show results for the number of *total* migrations attempted, the number of *successful* migrations, and the maximum



(a) NASA iPSC/860 (b) SDSC RS/6000 SP

(c) NASA iPSC/860 (d) SDSC RS/6000 SP

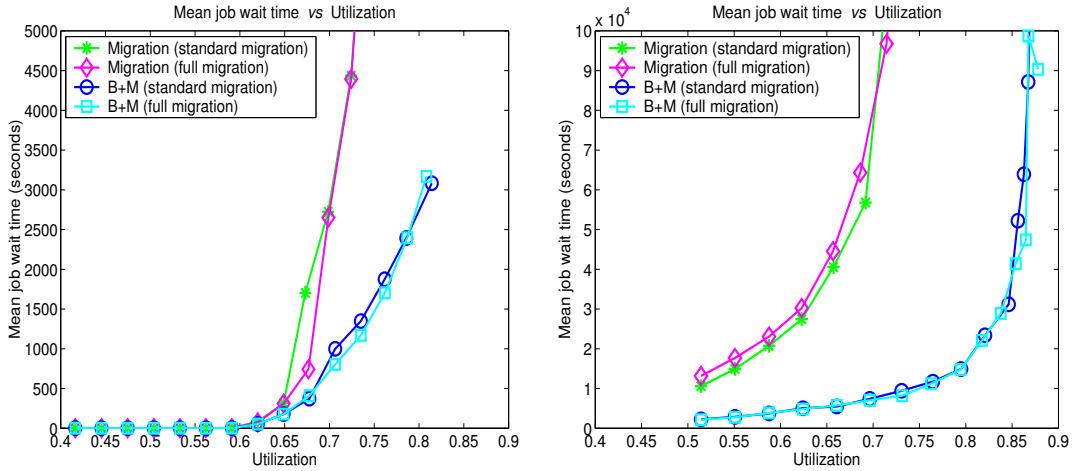
Figure 5-4: Number of total, successful, and maximum possible successful migrations *vs* utilization ((a) and (b)), and average time between migrations *vs* utilization ((c) and (d)).

possible number of successful migrations (*max successful*). As described in Chapter 2, the parameters which determine if a migration should be attempted are FN_{tor}, the ratio of free nodes in the system compared to the size of the torus, and FN_{max}, the fraction of free nodes contained in the maximal free partition. According to our standard migration policy, a migration is only attempted when $FN_{tor} \geq 0.1$ and $FN_{max} \leq 0.7$. A *successful* migration is defined as a migration attempt that improves the maximal free partition size. The *max successful* value is the number of migrations that are successful when a migration is always attempted (*i.e.*, $FN_{tor} \geq 0.0$

and $FN_{max} \leq 1.0$).

Almost all migration attempts were successful for the NASA log. This property of the NASA log is a reflection of the better packing caused by having jobs that are exclusively power of two in size. For the SDSC log, we notice that many more total attempts are made while about 80% of them are successful. If we always try to migrate every time the state of the torus is modified, no more than 20% of these migrations are successful, and usually much less.

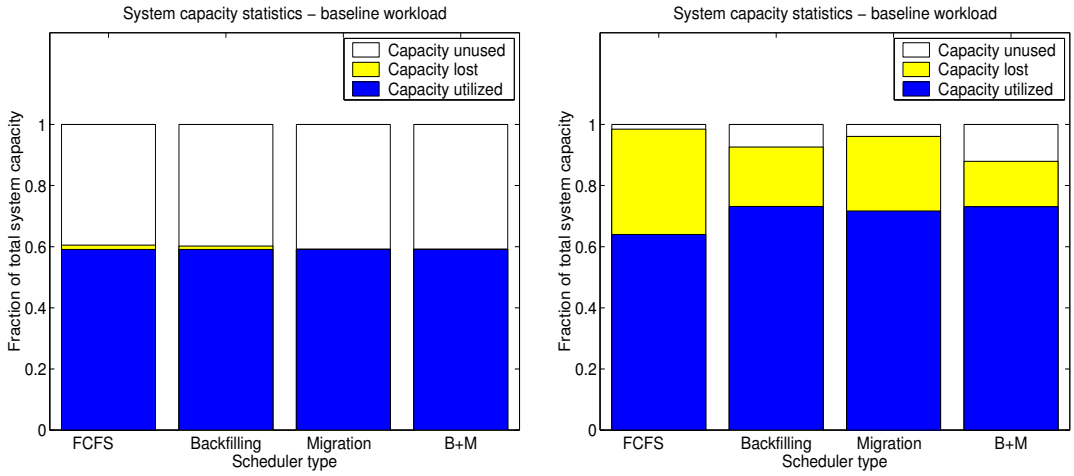
For the NASA log, the number of migrations increases linearly while the average time between these migrations varies from about 90 to 30 minutes, depending on the utilization level and its effect on the amount of fragmentation in the torus. In contrast to the NASA log, the number of migrations in the SDSC log do not increase linearly as utilization levels increase. Instead, the relationship is closer to an elongated bell curve. As utilization levels increase, at first migration attempts and successes also increase slightly to a fairly steady level. Around the first signs of saturation the migrations tend to decrease (*i.e.*, at around 70% utilization for the Migration scheduler and 77% for B+M). Even though the number of successful migrations is greater for the SDSC log, the average time between migrations is still longer as a result of the larger average job execution time.



(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-5: Mean job wait time *vs* utilization for the NASA and SDSC logs, comparing the standard migration policy to a full migration policy that always attempts to migrate.

Most of the benefit of migration is achieved when we only perform migration according to our parameters. Applying these parameters has three main advantages: we reduce the frequency of migration attempts so as not to always suffer the required overhead of migration, we increase the percentage of migration attempts that are successful, and additionally we increase the average benefits of a successful migration. This third advantage is apparent when we compare the mean job wait time results for our standard FN_{tor} and FN_{max} settings to that of the scheduler that always attempts to migrate. Even though the maximum possible number of successful migrations is sometimes twice as many as our actual number of successes, Figure 5-5 reveals that the additional benefit of these successful migrations is very small.

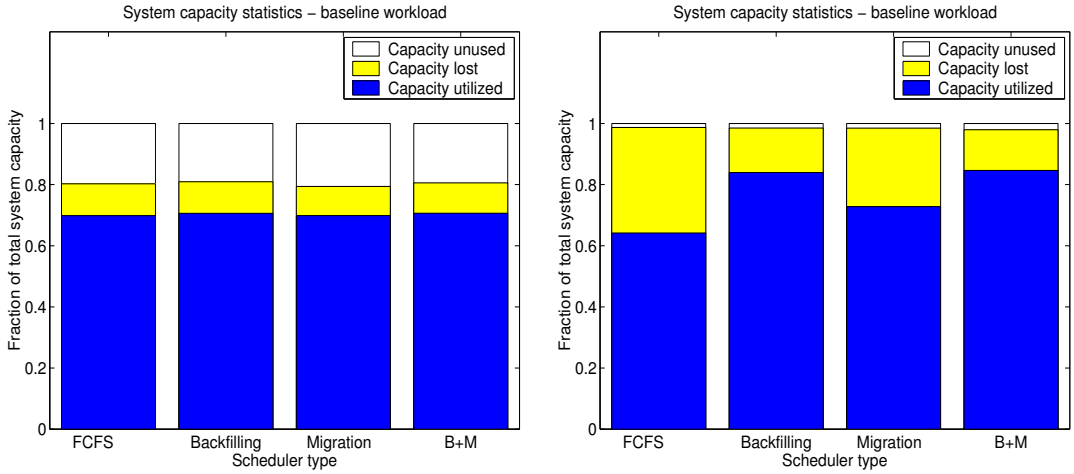


(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-6: Capacity utilized, lost, and unused as a fraction of the total system capacity.

We complete this chapter with an analysis of the results for system capacity utilized, unused capacity, and lost capacity. The results for each scheduler type and both standard job logs, with $c = 1.0$, are plotted in Figure 5-6. The utilization improvements for the NASA log are barely noticeable. The SDSC log, however, shows the greatest improvement when using B+M over FCFS, with a 15% increase in capacity utilized and a 54% decrease in the amount of capacity lost. By themselves, the Backfill and Migration schedulers each increase capacity utilization by 15% and 13%, respectively, while decreasing capacity loss by 44% and 32%. These results show

that B+M is significantly more effective at transforming lost capacity into unused capacity. Under the right circumstances, it should be possible to utilize this unused capacity more effectively.



(a) NASA iPSC/860 (b) SDSC RS/6000 SP

Figure 5-7: Capacity utilized, lost, and unused as a fraction of the total system capacity, with increased workload constant $c = 1.2$.

When we increase the workload by setting $c = 1.2$ and examine the system capacity results, as in Figure 5-7, it is apparent that the SDSC log has already reached the saturation region since essentially none of the capacity is unused. The performance gain for our space-sharing schedulers is also much larger, as the Backfill and B+M schedulers achieve greater than 84% capacity utilization, an improvement over the FCFS scheduler of 34%. The Migration scheduler achieves approximately half of this utilization increase with an improvement of 14%. In contrast, the NASA log is beginning to show only small signs of lost capacity, and our scheduling algorithms have minimal effect when so much of the capacity is unused. This is due to a combination of the job size restrictions that allow jobs to fill the torus more compactly, and also because the initial average NASA workload is 12% of the average SDSC workload.

Chapter 6

Related and Future Work

The topics of our work have been the subject of extensive previous research, much which has already been previously sited. In particular, [9, 16, 19] have shown that backfilling on a flat machine like the IBM RS/6000 SP is an effective means of improving quality of service. The benefits of combining migration and gang-scheduling have been demonstrated both for flat machines [26, 27] and toroidal machines like the Cray T3D [8]. The results in [8] are particularly remarkable, as system utilization was improved from 33%, with a pure space-sharing approach, to 96% with a combination of migration and gang-scheduling. The work in [23] discusses techniques to optimize spatial allocation of jobs in mesh-connected multicomputers, including changing the job size, and how to combine spatial- and time-sharing scheduling algorithms. An efficient job scheduling technique for a three-dimensional torus is described in [2]. This paper, therefore, builds on this previous research by applying a combination of backfilling and migration algorithms, exclusively through space-sharing techniques, to improve system performance on a toroidal-interconnected system.

Future work opportunities can further build on the results of this paper. The impact of different FCFS scheduling heuristics for a torus, besides the largest free partition heuristic currently used, can be studied. It is also important to identify how the current heuristic relates to the optimal solution in different cases. Additional study of the parameters I, FN_{tor}, and FN_{max} may determine further tradeoffs associated with partition size increases and more or less frequent migration attempts.

Finally, while we do not attempt to implement complex time-sharing schedulers such as those used in gang-scheduling, a more limited time-sharing feature may be beneficial. Preemption, for example, allows for the suspension of a job until it is resumed at a later time. These time-sharing techniques may provide the means to further enhance the B+M scheduler and make the system performance of a toroidal-interconnected machine more similar to that of a flat machine.

Chapter 7

Conclusions

We have investigated the behavior of various scheduling algorithms to determine their ability to increase processor utilization and decrease job wait time in a toroidal-interconnected system. We have shown that a scheduler which uses only a backfilling algorithm performs better than a scheduler which uses only a migration algorithm, while migration is particularly effective under a workload that produces a large amount of fragmentation (*i.e.*, when many small to mid-sized jobs of varied sizes represent much of the workload). Migration has a significant implementation overhead but it does not require any additional information besides what is required by the FCFS scheduler. Using a selective migration policy based on a few key parameters, we can decrease the overhead of a migration policy and achieve similar results as a full migration policy by only attempting to migrate when the expected benefits of that migration attempt is sufficiently high. Furthermore, a migration scheduler does not sacrifice any measure of fairness. Backfilling, on the other hand, does not have a significant implementation overhead but requires additional information pertaining to the execution time of jobs. By sacrificing a small amount of fairness, the expected benefits of a pure backfilling strategy are usually much greater than a migration scheduler.

Simulations of FCFS, backfilling, and migration space-sharing scheduling algorithms have shown that B+M, a scheduler which implements all of these algorithms, shows a small performance improvement over just FCFS and backfilling. However,

B+M can convert significantly more lost capacity into unused capacity than backfilling alone. Additional enhancements to the B+M scheduler may harness this unused capacity to provide further system improvements. Even with the performance enhancements of backfilling and migration techniques, a toroidal-interconnected machine such as BG/L can only approximate the job scheduling efficiency of a fully connected machine in which all nodes are equidistant.

Bibliography

[1] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir. **SP2 system architecture**. *IBM Systems Journal*, 34(2):152–184, 1995.

[2] H. Choo, S.-M. Yoo, and H. Y. Youn. **Processor Scheduling and Allocation for 3D Torus Multicomputer Systems**. *IEEE Transactions on Parallel and Distributed Systems*, 11(5):475–484, May 2000.

[3] J. E. Moreira E. Krevat, J. G. Castanos. **Job Scheduling for the BlueGene/L System**. In *Proceedings of the 8th International Euro-Par Conference (Research Note)*, pages 207–211, August 2002. LNCS 2400.

[4] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. **A worldwide flock of Condors: Load sharing among workstation clusters**. *Future Generation Computer Systems*, 12(1):53–65, May 1996.

[5] D. G. Feitelson. **A Survey of Scheduling in Multiprogrammed Parallel Systems**. Technical Report RC 19790 (87657), IBM T. J. Watson Research Center, October 1994.

[6] D. G. Feitelson. **Packing schemes for gang scheduling**. In *Job Scheduling Strategies for Parallel Processing, IPPS'96 Workshop*, volume 1162 of *Lecture Notes in Computer Science*, pages 89–110, Berlin, March 1996. Springer-Verlag.

[7] D. G. Feitelson. **Parallel Workloads Archive**. URL: `http://www.cs.huji.ac.il/labs/parallel/workload/index.html`, 2001.

[8] D. G. Feitelson and M. A. Jette. **Improved Utilization and Responsiveness with Gang Scheduling**. In *IPPS'97 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 238–261. Springer-Verlag, April 1997.

[9] D. G. Feitelson and A. M. Weil. **Utilization and predictability in scheduling the IBM SP2 with backfilling**. In *12th International Parallel Processing Symposium*, pages 542–546, April 1998.

[10] H. Franke, J. Jann, J. E. Moreira, and P. Pattnaik. **An Evaluation of Parallel Job Scheduling for ASCI Blue-Pacific**. In *Proceedings of SC99, Portland, OR*, November 1999. IBM Research Report RC21559.

[11] B. Gorda and R. Wolski. **Time Sharing Massively Parallel Machines**. In *International Conference on Parallel Processing*, volume II, pages 214–217, August 1995.

[12] D. Hyatt. **A Beginner's Guide to the Cray T3D/T3E**. URL: `http://www.jics.utk.edu/SUPER_COMPS/T3D/T3D_guide/T3D_guideJul97.html`, July 1997.

[13] H. D. Karatza. **A Simulation-Based Performance Analysis of Gang Scheduling in a Distributed System**. In *Proceedings 32nd Annual Simulation Symposium*, pages 26–33, San Diego, CA, April 11-15 1999.

[14] E. Krevat, J. G. Castanos, and J. E. Moreira. **Job Scheduling for the BlueGene/L System**. In *Job Scheduling Strategies for Parallel Processing, 8th International Workshop*, pages 38–54, July 2002. LNCS 2537.

[15] D. H. Lawrie. **Access and Alignment of Data in an Array Processor**. *IEEE Transactions on Computers*, 24(12):1145–1155, December 1975.

[16] D. Lifka. **The ANL/IBM SP scheduling system**. In *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer-Verlag, April 1995.

[17] J. E. Moreira, W. Chan, L. L. Fong, H. Franke, and M. A. Jette. **An Infrastructure for Efficient Parallel Job Execution in Terascale Computing Environments**. In *Proceedings of SC98, Orlando, FL*, November 1998.

[18] U. Schwiegelshohn and R. Yahyapour. **Improving First-Come-First-Serve Job Scheduling by Gang Scheduling**. In *IPPS'98 Workshop on Job Scheduling Strategies for Parallel Processing*, March 1998.

[19] J. Skovira, W. Chan, H. Zhou, and D. Lifka. **The EASY-LoadLeveler API project**. In *IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 41–47. Springer-Verlag, April 1996.

[20] W. Smith, V. Taylor, and I. Foster. **Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance**. In *Proceedings of the 5th Annual Workshop on Job Scheduling Strategies for Parallel Processing*, April 1999. In conjunction with IPPS/SPDP'99, Condado Plaza Hotel & Casino, San Juan, Puerto Rico.

[21] H. S. Stone. **High-Performance Computer Architecture**. Addison-Wesley, 1993.

[22] C. Z. Xu and F. C. M. Lau. **Load Balancing in Parallel Computers: Theory and Practice**. Kluwer Academic Publishers, Boston, MA, 1996.

[23] B. S. Yoo and C. R. Das. **Processor Management Techniques for Mesh-Connected Multiprocessors**. In *Proceedings of the International Conference on Parallel Processing (ICPP'95)*, volume 2, pages 105–112, August 1995.

[24] K. K. Yue and D. J. Lilja. **Comparing Processor Allocation Strategies in Multiprogrammed Shared-Memory Multiprocessors**. *Journal of Parallel and Distributed Computing*, 49(2):245–258, March 1998.

[25] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. **Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques**. In *Proceedings of IPDPS 2000*, Cancun, Mexico, May 2000.

[26] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. **The Impact of Migration on Parallel Job Scheduling for Distributed Systems**. In *Proceedings of the 6th International Euro-Par Conference*, pages 242–251, August 29 - September 1 2000.

[27] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. **An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration**. In *Job Scheduling Strategies for Parallel Processing, 7th International Workshop*, pages 133–158, June 2001.

[28] B. B. Zhou, R. P. Brent, C. W. Jonhson, and D. Walsh. **Job Re-packing for Enhancing the Performance of Gang Scheduling**. In *Job Scheduling Strategies for Parallel Processing, IPPS'99 Workshop*, pages 129–143, April 1999. LNCS 1659.