Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization

James Patton Jones[1] and Bill Nitzberg[1]

MRJ Technology Solutions
NASA Ames Research Center, M/S 258-6
Moffett Field, CA 94035-1000

jjones@nas.nasa.gov

Abstract.The NAS facility has operated parallel supercomputers for the past 11 years, including the Intel iPSC/860, Intel Paragon, Thinking Machines CM-5, IBM SP-2, and Cray Origin 2000. Across this wide variety of machine architectures, across a span of 10 years, across a large number of different users, and through thousands of minor configuration and policy changes, the utilization of these machines shows three general trends: (1) scheduling using a naive FCFS first-fit policy results in 40-60% utilization, (2) switching to the more sophisticated dynamic backfilling scheduling algorithm improves utilization by about 15 percentage points (yielding about 70% utilization), and (3) reducing the maximum allowable job size further increases utilization. Most surprising is the consistency of these trends. Over the lifetime of the NAS parallel systems, we made hundreds, perhaps thousands, of small changes to hardware, software, and policy, yet utilization was affected little. In particular, these results show that the goal of achieving near 100% utilization while supporting a real parallel supercomputing workload is unrealistic.

1.0 Introduction

The Numerical Aerospace Simulation (NAS) supercomputer facility, located at NASA Ames Research Center, serves in the role of pathfinder in high performance computing for NASA. In the late 1980s, we began exploring the use of highly parallel systems for supporting scientific and technical computing [1]. Today, it is commonly accepted that "supercomputing" is synonymous with "parallel supercomputing".

Supercomputing means running "big" jobs or applications which cannot be run on small or average-sized systems. Big, of course, is a relative term; we generally consider a job big if it is using at least half of the available resources of a big system. (We leave the definition of "big system" to the reader.)

1. Work performed under NASA contract NAS2-14303, Moffett Field, CA 94035-1000

Traditional vector supercomputers (e.g., the Cray C90) are capable of sustaining nearly 100% utilization while supporting big jobs and running a varied workload [2]. Our experience has shown that this level of utilization is not attainable when running a supercomputing workload on a parallel supercomputer.

2.0 The NAS Parallel Supercomputing Workload

The NAS facility supports research and development in computational aerosciences. Hundreds of research projects are funded annually which use the parallel supercomputers at NAS to perform high-end scientific and technical computing. Over the past 11 years, the NAS parallel workload, priorities, and approach have been consistent.

The workload consists of a mix of:

- hundreds of users; new users are constantly added
- scientific and technical computing for aerospace applications
- code development, debugging, scaling and performance analysis
- “production” runs of existing applications

At the same time, the NAS scheduling policy has consistently striven for (in order of priority):

1. Overnight turn-around for big jobs, and
2. Good machine utilization.

The first priority supports “supercomputing”, the second supports efficient use of resources. NAS supports supercomputing by favoring supercomputer-sized jobs (big ones, typically those that cannot run on any other system within NASA) over smaller jobs. In general, the system configuration, user allocations, and scheduling policies are tuned so that big jobs get overnight turn-around.

In apparent conflict to the first priority is the second. Good machine utilization has historically meant 99% on traditional vector supercomputers, and the stakeholders (those whose money purchased the machines) have traditionally used utilization as a measure of success. As we show, parallel supercomputing does not achieve 99% utilization. It should be noted that machine utilization is arguably not the best measure of the “goodness” or value of a computing system. This issue is discussed further in section 5 below. System utilization is used as the basis of comparison in this paper primarily because utilization was the single largest continuous dataset available for the systems under discussion.

The system configuration and the mechanisms by which we let users run jobs has also been consistent throughout the past 11 years. The systems are all space-shared (partitioned), and batch scheduled. Interactive use is permitted, but it must take place by allocating resources via the batch system, then using those resources interactively. This approach to using parallel computers has prevailed, despite the availability of good time-sharing and gang-scheduling facilities on several systems, for two reasons: the need for consistency of timings and efficiency of execution. Analysis of algorithms, exact megaflop rates, and scaling are major component of the NAS workload. Approaches other than strict, partitioned space sharing don't support this. Furthermore, systems such as the Intel Paragon and Cray Origin 2000 suffer from an interference problem (discussed below), in which it is possible for jobs to "overlap" in such a way as to slow each other down by far more than would be expected by the simple sharing of resources.

Most of the applications run at NAS are statically balanced (applications which require a well balanced load across all nodes). Statically balanced applications strive to give an equal amount of work to each processor. A single slow process in a statically load-balanced application can completely ruin the performance of the application, as other processes will have to wait for it. Another issue arises from message-passing synchronization. Even if we overlay parallel jobs to avoid load-balancing problems, tightly synchronized applications can incur an extra synchronization delay for messages because processes are not *gang-scheduled* (scheduled to run at the same time across all their assigned nodes). These constraints are consistent across typical parallel supercomputing workloads. (For further discussion of parallel supercomputing workloads, see [12].)

3.0 Supercomputer Resource Sharing

In a supercomputer, the two resources most visible to the user are the CPU and the memory. In parallel supercomputers, these resources are generally grouped together as compute nodes. This paper focuses on node utilization. There are two methods of sharing CPUs in a large MPP system: *time sharing* and *space sharing*. Time sharing allows different programs to run on the same node simultaneously. The operating system is responsible for scheduling different programs to run, each for a certain time slice (quantum). Space sharing (also known as *tiling*) gives a parallel application exclusive access to a set of compute nodes on which to run.

The five systems under review have a mixture of node sharing methods, as shown in Table 1.

Parallel Systems	**Intel IPSC/860**	**TMC CM5**	**Intel Paragon**	**IBM SP-2**	**SGI Origin2000**
Gang Scheduling		✔	unusable		
Time Sharing		✔	unusable	✔	✔
Space Sharing	✔	✔	✔	✔	✔

Table 1. MPP Node Sharing Methods

With the exception of the IPSC/860, all these systems support timesharing. Timesharing works well for serial jobs, which can fit into the memory of a single node. But at NAS we are interested in running parallel applications that cannot run on a single node because of the resources they require (such as more memory than is available, or so much CPU that the single-node run time would be too long). It is often assumed that parallel jobs can be timeshared automatically by the operating system. In reality, however, the issues of load balancing and synchronization make timesharing unacceptable for the parallel applications in the NAS workload.

The issues of timesharing the NAS workload on parallel systems was discussed in detail several years ago [10]. The following excerpt is relevant here:

> Among the statically balanced applications is a very important class of tightly synchronized communication-intensive codes. These applications form a large part of the NAS work load, and tightly synchronized applications are common in other fields. The reason these applications cannot be efficiently timeshared is the accumulation of communication delays created by the uncoordinated scheduling of processes across the nodes. In tightly synchronized applications, information flows between nodes as the calculation progresses. Even when there is only nearest neighbor communication, information flows from neighbor to neighbor eventually reaching all nodes. Every time that information flow is disrupted the entire application slows down.

One solution to this problem is to coordinate time sharing across the nodes of a parallel application, so that all processes of a given application run at the same time. This is called *gang-scheduling* or *co-scheduling* [9]. Gang-scheduling requires operating system support, and a scheme for handling communication in progress (i.e. no messages should be lost when processes are swapped) [3]. Although not necessary, it is easier to implement gang scheduling if nodes are grouped into fixed size partitions, though jobs then do not have flexibility in how many nodes they run on. Since all nodes in a partition run the same number of processes, the scheduler does not have to deal with unbalanced scheduling. Gang-scheduling in fixed-size partitions is an effective way to deal with the problems of timesharing.

Unfortunately, only the Paragon and the CM5 supported gang scheduling. The Paragon's implementation of gang scheduling added so much instability to the system that it proved unusable. On the CM5 we determined that the performance impact outweighed any benefits gang scheduling would have provided.

The second scheduling method is *space sharing*. In this model, a parallel application is given exclusive access to a set of compute nodes on which to run. If any timesharing occurs, it is with the mostly inactive UNIX daemons. The parallel computer can then be divided between different parallel applications, without one competing with another for resources. Space sharing, however, does increase the difficulty of batch job scheduling. Space sharing can also lower the ultilization of other system resources. Since jobs have exclusive control over the nodes, space sharing controls the

CPU together with other system resources, like memory, disks, and interconnect bandwidth. If the application is not using those resources, it is wasted.

Given the vast proportion of the NAS workload that consists of message-pasing applications, space sharing has repeatably been chosen for the NAS parallel supercomputers in order to deliver the greatest per-application performance and consistent runtimes.

4.0 Analysis of NAS Parallel Supercomputer Utilization

In this section we describe the hardware configuration of five NAS parallel supercomputers and discuss batch job scheduling and utilization for each. (For a discussion of batch job scheduling requirements for large parallel supercomputers, like those at NAS, see [10].) During the analysis, several trends begin to emerge. These are discussed as they become apparent in the data presented.

Figures 1-6 show the weekly *available* node utilization for each of the five NAS MPP systems under review, from installation (or when the system was stable enough to put users on) until decommission. By “available” node utilization, we mean that both scheduled and unscheduled outages have been taken into consideration when calculating the percentage utilization.

4.1 Intel iPSC/860 (Jan. 1990 to Sept. 1994)

The Intel iPSC/860 (also known as the Touchstone Gamma System) is a MIMD parallel computer. The system at NAS consisted of 128 compute nodes (each with a single 40 MHz i860 XR processor and eight megabytes of physical memory), ten I/O nodes (each with an i386 processor and four megabytes of memory), one service node (with a single i386 processor, four megabytes of memory, and an ethernet interface), and an i386-based PC front end with eight megabytes of memory. The compute nodes are connected via a wormhole-routed hypercube network, which delivers 2.8 megabytes per second per link.

The Network Queueing System (NQS, [8]) was used as the batch system, implementing queue-level “first-come first-serve first-fit” (FCFS-FF) scheduling with different size priorities during the day (i.e. big jobs had priority at night, small jobs during the day). The FCFS-First-Fit algorithm works as follows: batch jobs are evaluated in FCFS order in the queue, i.e. oldest job first. For each job, the batch system first checked if there were enough nodes available to run the job, and if so, then compared the job requirements (walltime and node count) to the current scheduling policy. If either of these two checks failed, the scheduler skipped to the next job. If both were successful, the scheduler ran the job and removed it from the list. This process continued until all the jobs were evaluated.

Scheduling on the iPSC/860 was relatively simple, as the system itself was very inflexible. The architecture divided the system into “partitions” each of a power-of-2 number of nodes. Batch jobs were then run in the smallest possible partition size. This made scheduling easier, but forced idle time when running medium- sized jobs. For example, a 65-node job could run only in a 128-node partition.

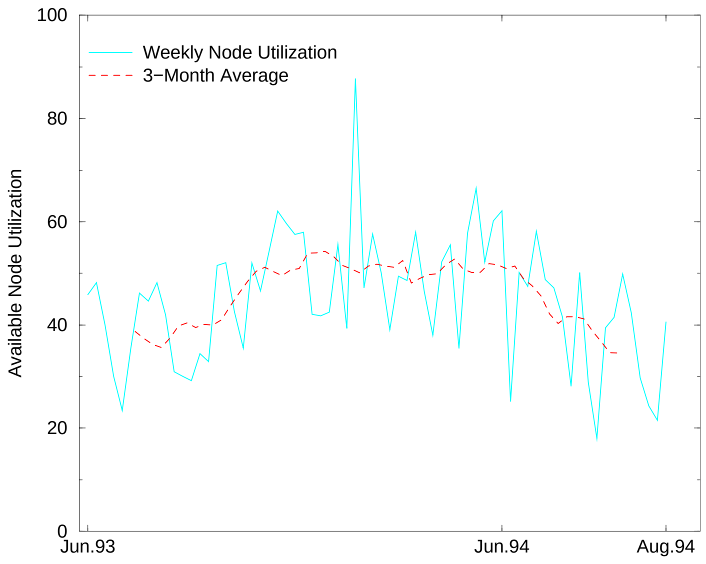

Fig. 1. iPSC/860 Utilization

Since there was no timesharing available, this forced the remaining 63 nodes to be left idle. Furthermore, there existed a system limit of a maximum of ten concurrent partitions. This limit also had the potential for forcing idle time, even when there was a backlog of work. For example, if the iPSC/860 was running ten 2-node jobs, the remaining 108 nodes would be idle. But given the typical job size in the NAS workload, the maximum partition limit was rarely exceeded. (The system ran out of nodes well before it allocated ten partitions.)

The iPSC/860 was fairly unreliable during the first two years at NAS. The first year the system was thoroughly investigated by NAS staff, during which time a variety of benchmarks were developed and run on the system. Figure 1 shows the node utilization starting in mid-1993. (Full accounting data for the first two years is unavailable.) At the time, the utilization shown was considered an impressive improvement over that of previous years, and is primarily attributable to two factors. The first was a significant increase in system stability. Second, in early 1993, users had begun to shift from application debugging to running their codes as “production” batch jobs. Notice that the utilization ranged between 40 and 60 percent for most of the period shown.

4.2 TMC CM-5 (Jan. 1993 to Mar. 1995)

The Thinking Machines Corporation (TMC) CM-5 is a MIMD parallel computer, although it retains some properties of its SIMD predecessor, the CM-2. Notably, each processing node of the CM-5 can be thought of as a small SIMD parallel computer, each with a master SPARC processor sequencing a 2 x 2 array of custom vector units. Furthermore, via a dedicated control network, the processing nodes can also operate in a synchronized SPMD fashion.

The system at NAS consisted of 128 compute nodes (each with one 33 MHz SPARC processor, four vector units, and 32 megabytes of physical memory), four control nodes, one I/O node (which manages the attached RAID), and one system-manager node. Each of these nodes was a standard Sparc-2 workstation (with 64 megabytes of physical memory). The nodes are interconnected via a 4-ary fat-tree data network, which supplies 20 megabytes per second per link, and a bisection bandwidth of 655 megabytes per second.

The CM-5 was scheduled using the Distributed Job Manager (DJM) which also implemented a size-priority FCFS-FF algorithm that was time-of-day sensitive. Like the iPSC/860, the CM-5 architecture restricted all partitions to a power-of-2 number of nodes. However, the CM-5 further restricted the partition size to a minimum of 32 nodes, and the partition size could not be changed without a reboot of the entire system. During the day, the CM-5 was run with one 64-node partition and two 32-node partitions. Each night, the system was reconfigured into a single 128-node partition to allow large jobs to run.

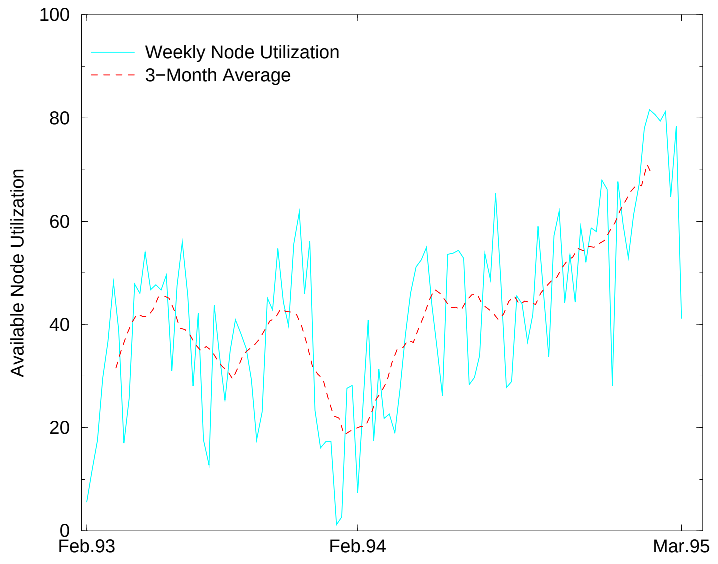

Fig. 2. CM-5 Utilization

The CM-5 followed quite a different life cycle than the iPSC/860. Initially only NAS staff had access for benchmarking and evaluation purposes, and to work to stabilize the system. But rather than taking several years as with the iPSC/860, we had to wait only several weeks before putting “real” users on. Figure 2 shows how quickly the scientists put the CM-5 to work. Part of the reason for the short ramp-up was that most of the researchers were migrating to the CM-5 from the previous generation CM-200 (which had been previously upgraded from a CM-2) at NAS. Many of these users already had codes that ran well on this architecture.

Like the iPSC/860 much of the increased usage of the CM-5 in its final year was due to users completing the debugging cycle, and moving to running production codes. Halfway through the second year of the system’s stay at NAS, in an effort to increase the utilization of the machine, space sharing was relaxed on the small partitions during the day to allow two jobs to timeshare within a given partition. Doing so resulted in a gradual increase of utilization; however, it also resulted in a 20 percent slowdown in both timeshared applications.

4.3 Intel Paragon XP/S-15 (Feb. 1993 to July 1995)

The Intel Paragon XP/S-15 is a MIMD parallel computer. The system at NAS consisted of 208 compute nodes (each with two 50 MHz i860 XP processors and 32 megabytes of physical memory), four service nodes (which make up the service partition and provide an interface to the outside world, serving as a “front end” to the system), eight disk I/O nodes, three HIPPI nodes, and four general-purpose nodes. The compute nodes are connected via a wormhole-routed 2D mesh network, which delivers 175 megabytes per second per link. The Paragon is the successor of the Delta machine.

Using NQS, we implemented queue-level FCFS-FF scheduling with different size priorities, as on the iPSC/860. Scheduling the Paragon, however, was more difficult than scheduling the previous systems because power-of-2 job sizes were no longer required. The resulting wide variety of job sizes decreased the scheduling efficiency.

The Paragon, like the CM-5, had a relatively short shake-out period before we started adding users onto the system. These were primarily users from the iPSC/860 who wanted to try out the new system. Once on, many chose to return to the iPSC/860 until the Paragon stabilized.

The utilization shown in Figure 3 for the first half of 1993 is based on UNIX SAR (system activity report) and load average data. Some data for 1993 were lost (thus the apparent zero usage). Following this, the MACS accounting software was installed, enabling more accurate utilization tracking. This is also the time when the remaining iPSC/860 users began to migrate over to the Paragon in order to continue their work. (Compare the iPSC/860 and the Paragon utilization graphs in Figure 6 to see more clearly the drop in the older system corresponding to an increase in the newer system.)

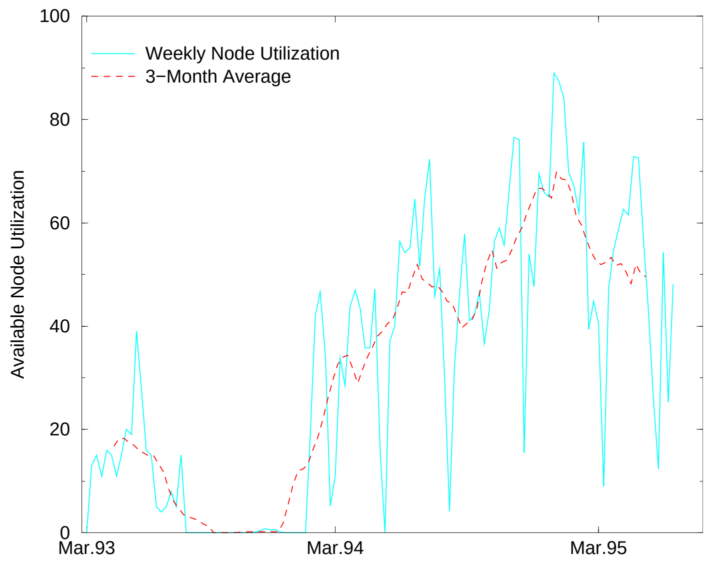

Fig. 3. Paragon Utilization

The periodic dips in the Paragon utilization correspond to testing and regular operating system upgrades. During these times, though the system was available to users, many opted off the machine to avoid the frustrations of trying to use a system in flux. From the utilization graph, we see that the FCFS-FF algorithm maintained the utilization between and 40 and 60 percent for most of the "production" lifetime of the system.

4.4 IBM SP-2 (July 1994 to Sept. 1997)

The IBM SP-2 is a MIMD parallel computer. The system at NAS consisted of 160 compute nodes. Each node is an IBM RS6000/590 workstation powered with a single 66.7 MHz POWER2 processor and at least 128 megabytes of physical memory. (Six of the nodes had 512 megabytes of memory). The nodes of an SP-2 are connected by a packet-switched, multi-stage omega network (a hierarchy of crossbar switches) utilizing buffered wormhole-routing. The switch can deliver 40 megabytes per second bidirectionally.

The system arrived with IBM's LoadLeveler batch system, which provided simple FCFS scheduling. Based on what we had learned with previous parallel systems, we predicted that a FCFS-FF scheduling algorithm would result in a system node utilization of around 50 percent. However, the Loadleveler batch system used a simple FCFS algorithm which was achieving roughly 25 percent utilization. After six months, we replaced LoadLeveler with the Portable Batch System (PBS) utilizing a FCFS-FF algorithm [5]. System utilization immediately doubled (see Figure 4), averaging 50 percent. This level of utilization continued the entire time we used the FCFS-FF scheduler.

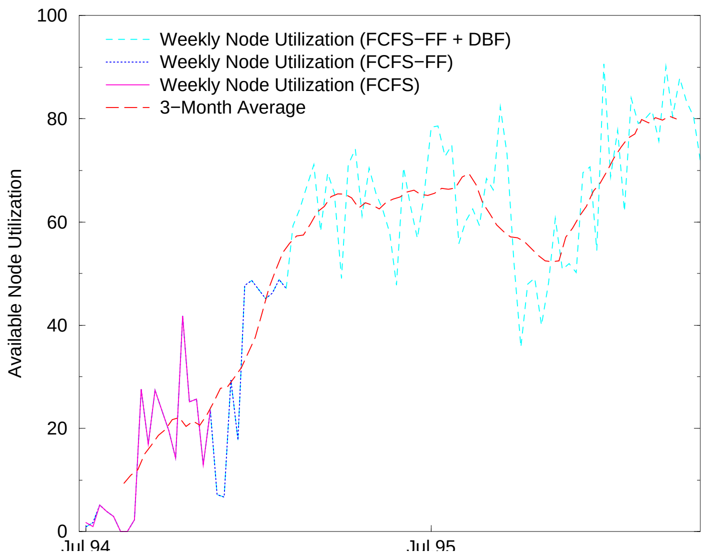

Fig. 4. SP-2 Utilization

One problem with the First-Fit algorithm is that it can continually take away nodes from a large waiting job in order to run smaller jobs. As a result, large jobs tend to "starve" in the queue, always waiting for resources to become available, but never receiving them. One attempt to remedy this situation is to periodically "drain" the system in order to free up enough nodes to be able to run the large waiting jobs. However, draining the system is an expensive operation since a large number of nodes may need to be kept idle to ensure that a particular job can run. For example, suppose we want to run a 128-node job, but there are only 127 nodes available, and there is a 5-hour job running on the last node. We have to keep 127 nodes idle for 5 hours to guarantee that this job will run. While this simplistic approach works, it is obvious that it does not lead to the best system utilization possible.

A better solution is to have the scheduler recognize that we will not be able run our job for 5 hours, but we can use the 127 nodes to run any jobs that can complete in 5 hours, using a *backfilling* method. *Static-Backfilling* fixes the starting time of the high-priority job at the earliest time possible (i.e., the earliest time when the required nodes will be available). In our previous example, this was 5 hours. A *Dynamic-Backfilling* (DBF) algorithm, rather than fixing the time at the earliest time that it can, will instead determine the most appropriate time to start the job within a starting window [4, 11]. The earliest time possible may in some cases not lead to the best result. Using our previous example, let's assume that we also had a 125-node job (for 5 hours 30 minutes) queued. Using static-backfilling, we could not run this job as we only have 5 hours to backfill. But with dynamic-backfilling, we should recognize that by shifting the starting time by 30 minutes, we will be able to fill 125 nodes, significantly increasing resource utilization. Thus the DBF algorithm attempts continually to balance both the need to run high-priority jobs and the need to maintain as many nodes as possible in-use, by providing the ability to drain the system efficiently and to reserve nodes for the top-ranked job.

As soon as we installed PBS, we began implementing the DBF algorithm. Approximately two months later, the PBS FCFS-FF scheduler module was replaced with our PBS DBF scheduler module. (This DBF scheduling module is now included in the PBS distribution.) System utilization again jumped, this time roughly 20 percentage points, to 70 percent. Over time, as users began to run fewer debugging jobs and started scaling up the problem size of their applications, the utilization slowly crept up to 80 percent. The system continued at this level of usage until it was allocated to the NASA Metacenter project. (Given that the Metacenter project introduced many new variables, and thereby substantially changing the userbase, scheduling model and workload flow of the system, we do not report those data here. Further discussion of the Metacenter project and its meta-scheduling experiments is included in [6, 7].)

4.5 Cray Origin2000 (Jan. 1997 -)

The SGI/Cray Origin2000 is a MIMD computer based on the cache-coherent non uniform memory architecture (ccNUMA) providing a distributed shared memory (DSM) system. Each node contains two MIPS RISC 64-bit R10000 processors and a configurable amount of memory; nodes are connected via a modified hypercube network

In January 1997, NAS received its first 32-processor SGI Origin2000. (Systems larger than 32-processors receive the “Cray” label.) One of the most useful features of the Origin2000 is the single-system image. Users can utilize the system as a large symmetrical multiprocessor (SMP) rather than having to be concerned with the distributed nature of the architecture. Therefore when the system first arrived, we decided to see if we could schedule it like a true timeshared SMP. We installed PBS with a FCFS-FF SMP scheduling module that we had been running on our cluster of four Cray J90s.

In spite of its single system image, the attempt to schedule this distributed memory system quickly proved problematic, as both the underlying architecture and the interference between jobs resulted in severe performance degradation and varied runtimes. Specifically, since the hardware components are distributed across the interconnected hypercubes, the number of network routers between any two nodes within the system increases with the distance between those nodes. This alone translates into a variable latency in communication for message passing applications. Since applications could be started on any set of nodes within the system, the runtimes of a given application varied from run to run.

In addition, the operating system attempts to start applications on contiguous nodes, but as the system "fills up" with work the nodes quickly become fragmented since every application has a different runtime. As a result, applications are started on nodes that are scattered throughout the system. As this tends to increase the distance between the nodes assigned to a given job, latency again increases, as do runtimes.

The third problem we identified was in the "sharing" of nodes between jobs. Remember that an Origin2000 "node" has two processors which have equal access to locally shared memory. Scheduling multiple applications onto the same node makes it possible for these processes to compete for the memory on that node, delaying message-passing and thereby further increasing the runtime of the application as a whole. While many sites run applications which are tolerant of these conditions, the applications run at NAS displayed a range of variation in runtimes, from 30% on a lightly loaded system up to 300% on a fully loaded system.

Needless to say, we quickly turned to another solution. We switched to software partitioning of the system, where we assigned sets of processors to specific partitions. Each partition was assigned an SGI NODEMASK which identified which nodes belonged to that partition. Batch jobs were then scheduled into the smallest possible partition. (A NODEMASK is similar to a "processor set", except it is node-based rather than CPU-based. While not a perfect solution, the NODEMASK capability proved quite functional, even though it was only advisory to the kernel. Some of this functionality will be made available in the SGI MISER kernel scheduler.)

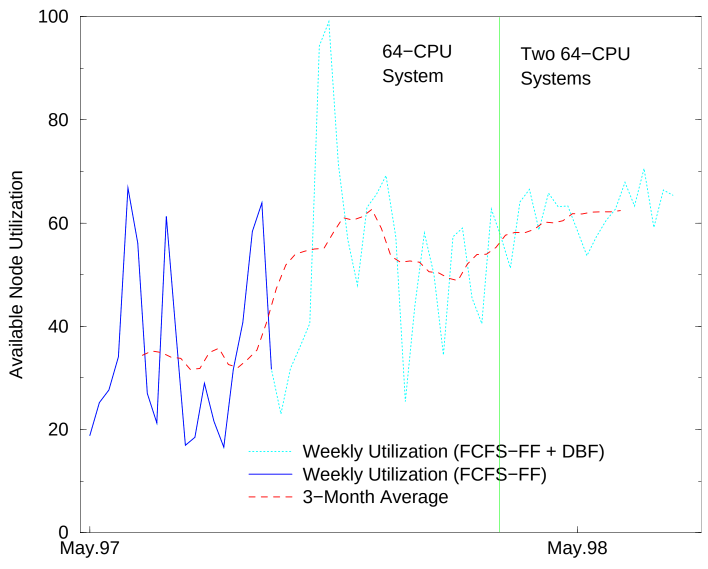

Fig. 5. Origin2000 Utilization

In March 1997, we doubled the size of our Origin2000 system, creating a 64-processor parallel supercomputer running under a single system image. Figure 5 shows the utilization starting with the installation of this system.

We scheduled each of the partitions within the Origin2000 as we did on the previous parallel systems. Figure 5 shows the system utilization that resulted: a rough average of 35 percent. From our previous experience, we predicted that adding dynamic backfilling to the scheduling algorithm would add another 15 percentage points to the utilization. Our prediction was again borne out: average system utilization increased from about 35 percent to about 55 percent. Nearly a year later NAS purchased a second 64-processor Origin2000 system. Rather than being run as a separate system, it was configured to share the workload of the first system.

Another trend we have noticed is that if you change the definition of “big jobs” in relation to the maximum number of nodes available, utilization will increase. We predicted that by adding this second 64-processor system to the compute pool (thereby doubling the number of processors available for computation) but maintaining the maximum job size at 64-processors, utilization should increase. We were surprised at how smoothly the utilization curve in Figure 5 spanned the doubling of the resources without increasing the maximum job size. This appears to be in part a result of the second system being identical to the first. The amount of resources doubled, and the users responded by submitting twice as many jobs as before. Turnaround time and utilization remained constant, but throughput doubled..

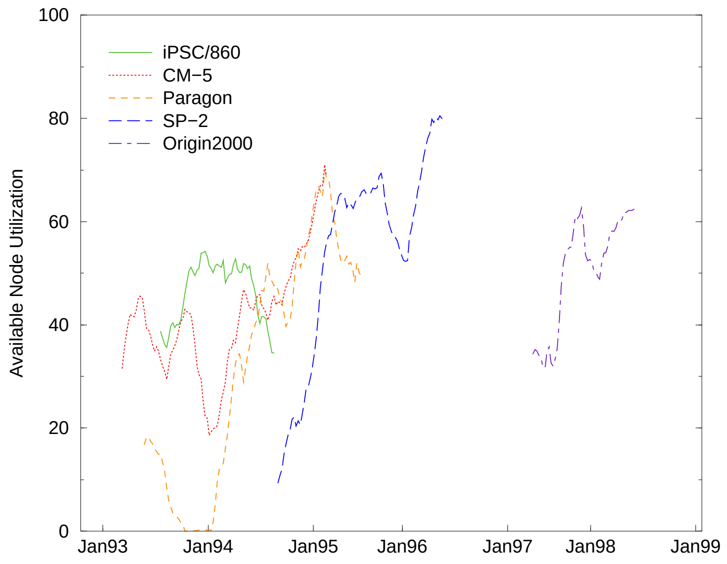

Fig. 6. Comparison of Parallel Supercomputer Utilization

Figure 5 ends at the point of installation of two additional 128-processor Origin2000 systems. Given that the full user base was not given immediate access, there are insufficient data to make a fair comparison between these new systems and those already reported. Then, in mid-November, the two 128-processor systems were merged into the first 256-processor Origin2000, and the full NAS parallel system user base was given access. Accounting, scheduling policy, and the scheduler itself have been continually changing since that time. Not until the system changes are stabilized will we be able to compare the 256-processor origin to the other systems reviewed herein

In order to make the comparison of the various system utilization graphs easier, a composite graph is included above. Figure 6 shows the lifetime utilization of each of the five MPP systems, along a single timeline. It is useful to compare the arrival and decommission of the systems with each other. From such a composite graph, it is easier to see that the iPSC/860 utilization began dropping as the Paragon started rising.

5.0 Caveats and Conclusions

First, a caveat. There are hundreds of variables which contribute to effective scheduling of a parallel supercomputer. We have ignored all but one—the scheduling algorithm—in our analysis. Further, we make some broad, sweeping generalizations which are heavily dependent on the user workload of the system. Although experience has shown that the NAS facility is a prototypical supercomputing center, one should not totally discount the possibility that the unique features of this facility contribute to these results.

Data gathered over 11 years of operating parallel supercomputers (including the Intel iPSC/860, Intel Paragon, Thinking Machines CM-5, IBM SP-2, and Cray Origin 2000) show three distinct trends:

- scheduling using a naive FCFS first-fit policy results in 40-60% utilization,
- switching to the more sophisticated dynamic-backfilling scheduling algorithm improves utilization by about 15 percentage points (yielding about 70% utilization), and
- reducing the maximum allowable job size increases utilization.

Most surprising is the consistency of these trends. Over the lifetime of the NAS parallel systems, we made hundreds, perhaps thousands, of small changes to hardware, software, and policy. Yet, utilization was affected little, in general increasing slowly over the lifetime of the system, with the few significant increases attributable to improvements in the scheduling algorithms. In particular, these results show that the goal of achieving 100% utilization while supporting a real parallel supercomputing workload is currently unrealistic. The utilization trends are similar irrespective of what system is used, who the users are, and what method is used for partitioning resources.

6.0 Future Work

The “goodness” of a scheduling algorithm and policy is hard to quantify. Although utilization is the most commonly used metric, it is still inadequate. Utilization does not take into account properties of the workload which, even if the jobs were optimally scheduled, would not yield 100% utilization. Such workload parameters as job-size mix, arrival rate of jobs, efficiency of applications, etc., are ignored. A better metric is needed. The utilization we report is simply the amount of time processors are assigned out of the total time processors are available (i.e., we ignore only system down time). We would like to refine utilization to be based not on the total uptime of a system, but on the optimal scheduling of the given workload. Other metrics we would like to explore are throughput measures and comparison of “time-to-solution”.

Seven years of job accounting data for five different parallel supercomputers is a lot of data; we have only scratched the surface. We would like to analyze big-job turnaround times in a fashion similar to our analysis of utilization trends. Further, we would like to investigate correlations between system stability (crashes), user load, turnaround time, workload characteristics, utilization, and, if possible, system culture (e.g., night time vs. day time, conference deadlines, etc.).

References

1. David H. Bailey, *Experiences with Parallel Computers at NASA/Ames* **NAS Technical Report RNR-91-007**, NAS Facility, NASA Ames Research Center, February 1991.

2. Nick Cardo, *Batch Scheduling: A Fresh Approach*, in **Proceedings of Cray’s User Group Conference**, March 1995.

3. **Job Scheduling in Multiprogrammed Parallel Systems** by Dror G. Feitelson, IBM Research Report RC 19790 (87657), October 1994.

4. Dror Feitelson and A. Mu'alem Weil, *Utilization and Predictability in Scheduling the IBM SP2 With Backfilling*, in **Proceedings of 12th International Parallel Processing Symposium.**, pp. 542-546, April 1998.

5. Robert Henderson, *Job Scheduling Under the Portable Batch System*, in **Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing**, Santa Barbara, CA, April 1995.

6. James Patton Jones, *The NASA SP2 Metacenter*, in **Proceedings of the Computational Aerosciences Workshop**, HPCCP, August 1996.

7. James Patton Jones, *Implementation of the NASA Metacenter: Phase 1 Report*, **NAS Technical Report NAS-97-027**, NAS Facility, NASA Ames Research Center, October 1997.

8. B. Kingsbury, *The Network Queuing System*, Sterling Software, Palo Alto, 1985.

9. **Scheduling Techniques for Concurrent Systems** by J.K. Ousterhout, In 3rd *International Conference of Distributed Computing Systems*, pp. 22-30, October 1982.

10. Bill Saphir, Leigh Ann Tanner and Bernard Traversat, *Job Management Requirements for NAS Parallel Systems and Clusters*, in **Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing**, Santa Barbara, CA, April 1995.

11. Bernard Traversat, Ed Hook and James Patton Jones. *A Dynamic-Backfilling Algorithm for Efficient Space-Sharing Scheduling on an IBM SP2*, **NAS Technical Report NAS-98-101**, NAS Facility, NASA Ames Research Center, November 1998.

12. K. Windisch, Virginia Lo, R. Moore, Dror Feitelson, and Bill Nitzberg, *A Comparison of Workload Traces From Two Production Parallel Machines*, in **Proceedings of 6th Synp. Frontiers of Massively Parallel Computing**, pp. 319-326, October 1996.