Analysis of the Early Workload on the Cornell Theory Center IBM SP2

Steven Hotovy David Schneider
Timothy O'Donnell
Cornell Theory Center
{shotovy,schneid,odonnell}@tc.cornell.edu
Theory Center Technical Report: 96TR234

January 31, 1996

Abstract

Parallel computers have matured to the point where they are capable of running a significant production workload. Characterizing this workload, however, is far more complicated than for the single-processor case. Besides the varying number of processors that may be invoked, the nodes themselves may provide differing computational resources (memory size, for example). In addition, the batch schedulers may introduce further categories of service which must be considered in the analysis.

The Cornell Theory Center (CTC) put a 512-node IBM SP2 system into production in early 1995. Extended traces of batch jobs began to be collected in mid-1995 when the usage base became sufficiently large. This paper offers an analysis of this early batch workload.

1 Introduction

1.1 Background

As the use of parallel computers spreads from research settings to commercial establishments, resource management of these machines becomes a more pressing issue. Job scheduling is an important aspect of this work, and much research on the topic of job scheduling for parallel computers has been done in recent years. Some have focused on distributed-memory machines [7, 8, 9, 11, 14], others on more general systems [3, 6, 10, 15]. Many different strategies (static vs. dynamic, time-sharing vs. space-sharing, etc.) have been investigated and compared. All these studies make underlying assumptions about the workload in order to quantify their results. Some of these studies use synthetic workloads whose characteristics can be controlled by key parameters. Others make use of popular parallel benchmark suites, notably the NAS Parallel Benchmarks [2].

To date, however, there has been very little published on actual parallel workloads. One obvious reason is that, until recently, parallel computers were used principally for research and development, not production. Nonetheless, some interesting research in workload characterization has been done. Cypher and co-workers [4] studied the memory, communication, computing and I/O resource requirements and utilization patterns of a number of scientific codes on distributed memory machines. In the Joint NSF-NASA Initiative on Evaluation (JNNIE) project [12], five NASA centers and the four NSF Supercomputer Centers identified 22 key applications and ran parallel versions of these programs on a diverse set of parallel machines. These applications involve key disciplines supported by these agencies, but are not meant to represent a typical workload. The most complete study of a production parallel workload was that done by Feitelson and Nitzberg [5]. They analyzed NQS trace records over the fourth quarter of 1993 for the 128-processor iPSC/860 at NASA Ames. In an earlier paper [13], the Cornell Theory Center (CTC) identified six classes of future key applications, and estimated the amount of resources these types of jobs would likely consume. These data represented an anticipated load, not the current workload, however.

This study extends this previous work in a number of ways. The number of node-hours generated by the workload under consideration is about double that found in [5], and consists of a broader class of applications. The heterogeneous nature of the CTC SP2 system [1] (node memory, batch classes, etc.) allows us to do more detailed analyses based on the particular resources requested. Since serial work is a non-negligible portion of the workload, the study separates out the serial and parallel components for many of the analyses. The study also includes an analysis of wait times, a topic not traditionally addressed in workload characterization studies.

The Cornell Theory Center put its 512-node IBM SP2 into production in early 1995. Users of the system are computational scientists who have been granted allocations of time via a peer review process. Many different disciplines are represented: those which have historically made use of computational resources, like computational chemistry and fluid dynamics, and newer ones like protein folding and ecological modeling.

Management of the batch queues for the SP2 is done by IBM's LoadLeveler software [16]. LoadLeveler has the ability to log trace records for each job. This feature was activated in mid-June, 1995, and extended traces have been collected since this time. Our workload analysis is based on data collected between June 18th and September 2, 1995, representing over 240,000 node-hours of execution time.

Node Type	128MB	256MB	512MB	1024MB	2048MB
Thin	352	30	0	0	0
Wide	0	22	21	4	1
Total	352	52	21	4	1

Table 1: Configuration of Batch System by Physical Memory and Node Type

1.2 SP2 Hardware Configuration

The Cornell Theory Center IBM SP2 system contains 512 nodes with an aggregate peak speed of 137 Gflops and a combined physical memory of 79 GBytes. The system, which was made available for production use in early 1995, is heterogeneous in that it contains a mixture of *wide* nodes and *thin* nodes. On the CTC SP2, *wide* nodes have been configured with 4 times the data cache and 4 times the data bandwidth between cache and memory compared to *thin* nodes. The nodes also vary in the amount of physical memory attached, from 128 MB up to 2048 MB.

The 512 processors themselves are segregated by the services they provide. The bulk of the processors, 430 in number, are devoted to running batch jobs. The remainder are used for interactive use, I/O gateways, special projects, and system testing.

The pool of batch nodes consist of both wide and thin nodes, and come in a variety of memory sizes. Table 1 details the node type/memory combinations.

The SP2 is also equipped with a High-Performance Switch, which directs data communication among the nodes. This switch runs under two protocols: the standard IP protocol (HPS-IP), and a more efficient one which resides in user space (HPS-USER).

Each node has 2GB of disk capacity. None of this storage is used for permanent files, however. Rather, this storage is used for swap space, caching of permanent files, and temporary space for users. Permanent data are stored on external AFS file servers or on the NSL Unitree mass storage system.

1.3 LoadLeveler Considerations

LoadLeveler provides a number of keywords to assist in the scheduling and execution of jobs. The most important ones from the standpoint of workload characterization are:

- Batch queue name
- The minimum number of processors required
- The maximum number of processors that can be used
- The switch protocol (USER or IP) to be used for parallel jobs

- The amount of memory requested
- The node type (THIN or WIDE)

LoadLeveler allows for dynamic definition of batch queues. At the Theory Center we have defined the following queues, all based on a maximum execution (wall-clock) time for a job:

- 15-minute
- 3-hour
- 6-hour
- 12-hour
- 18-hour

Within a given queue, there are no inherent limits to other resources, such as the number of processors, which the user may request. LoadLeveler will automatically cancel jobs that exceed the wall-clock limit for the queue, however. The LoadLeveler job scheduler is given control of assigning nodes from the entire pool of 430 processors. All queues draw from this same pool.

LoadLeveler employs space-sharing in the allocation of parallel jobs, i.e., once a node is assigned to a particular job, it is not available for further work until the job is completed. The scheduler uses a least heavily loaded algorithm to select candidate nodes. LoadLeveler imposes no limit on the number of jobs that may be queued in a given queue. LoadLeveler also employs an aging algorithm that increases the priority of jobs which have been waiting longer to receive service.

The Cornell Theory Center has established several policies which affect batch throughput. It should be noted that these are policy decisions, not limitations in the LoadLeveler software. To equalize access to the SP2, the Center originally established a maximum of 2 jobs per user that could be executing simultaneously (although there was no limit per user for the number of jobs waiting in the batch queues). In mid-1995, this limit was raised to 4 simultaneous jobs. Another policy is that all batch queues, except for the 15-minute one, will be given the same initial priority. The 15-minute queue is assigned a higher initial priority because it is intended for program development and debugging where turnaround time is more critical.

2 Data Collection and Interpretation

Extended LoadLeveler logs began to be collected in mid-June, 1995. Each trace record contains header information about the job itself and a sequence of short records for each node of a parallel job. The header information contains two timestamps: submittal time and completion time. The header also lists the job user time (roughly the aggregate CPU time for each participating node) and the job system time. The per-node report lists the timestamp when the node began processing, the timestamp when it was completed, the node user time and the node system time.

When LoadLeveler selects a job from the wait queue for processing, the following steps take place:

1. The LoadLeveler Central Manager accumulates the machines necessary to run the job, ensuring that all resource requirements are met;
2. The Central Manager notifies the host from which the job was submitted that it may begin to initiate the run;
3. The host sends the participating nodes a copy of the executable and asks for confirmation of ready state;
4. The host informs each node that it may begin to execute the job.

The *begin processing* timestamp for each node reflects Step 4. We define the **Job Start Time** as the earliest of the individual node *begin processing* timestamps. It should be noted that the difference between the first node's timestamp and the last node's is at most a few seconds. The *completion* timestamp for each node reflects the time when execution stops. We define the **Job Completion Time** as the latest of the individual node *completion* timestamps. Part of our characterization is based on **Job Duration** or **service time**, which is defined to be the difference between job start time and job completion time. It should be noted that this is a measure of elapsed wall-clock time, not related to CPU time.

Another important quantity in our analysis is the cumulative amount of time the nodes were busy running a job. For each node, the natural measure is the difference between the *begin processing* and *completed* timestamps. The cumulative time, referred to as **User Node Time**, is defined to be the sum of time differences from each node. This quantity is related to **service demand**, a key paramenter in the analyses of various scheduling policies [6, 10]. Like Job Duration, User Node Time is a measure of elapsed wall-clock time.

While all jobs were used in the arrival time analyses, some screening was needed for those analyses which assume a completed job. Complications come form two sources: jobs which were terminated abnormally and "Dead On Arrival" (DOA) jobs (i.e., those which fail immediately).

The LoadLeveler header provides information about the job's status. For the vast majority of jobs this status is *completed*, and such jobs were considered for analysis. However, other values, like *removed*, were possible. This represents the situation where a user cancelled his job, or where LoadLeveler began to run the job, but had difficulty in acquiring nodes. Such jobs were filtered out.

Queue	Number of Jobs		User Node Hours	
	Serial	Parallel	Serial	Parallel
15-minute	3306	6859	187.338	12708.722
3-hour	3530	3865	1714.430	29627.482
6-hour	1662	732	5071.119	14094.566
12-hour	1637	1436	9158.028	90165.279
18-hour	2418	534	18525.010	59165.575

Table 2: Number of Jobs and User Node Time by Queue.

The DOA problem refers to jobs with a *completed* status, but which did no useful work. This could be caused by an error in the script which controlled the batch job, a faulty executable, an incorrectly named input file, or the like.

There was no clearly discernible characteristic of the LoadLeveler traces that would allow us to distinguish true DOA jobs from successful, short-running programs. In the paper, though, we do attempt to identify DOA jobs and to estimate their impact on the workload.

Data for batch jobs were accumulated for the 11-week period from June 18, 1995 to September 2, 1995. During this time the machine was relatively stable, with no significant down times other than regularly scheduled ones on Wednesday and Friday mornings.

3 Serial/Parallel Load Analysis

Of the total workload, 48.4 percent of the jobs were serial and 51.6 percent were parallel. When one looks at User Node Time, however, only 14.5 percent was devoted to serial jobs and 85.5 percent to parallel jobs. Thus while the number of serial and parallel jobs were roughly the same, parallel jobs account for the vast majority of the computational work.

The load on the system showed minor fluctuations from week to week, with a gradual increase later in the reporting period. We did observe that the serial load remained almost constant during this time; all variations were due to the parallel load. Much of this serial load comes from legacy applications that had been running on the CTC IBM ES/9000 prior to its removal in March, 1995 and which continue to be run on individual nodes of the SP2.

3.1 Analysis by Queue

It is instructive to look at this breakdown as a function of submitted queue. Table 2 shows the results. The 15-minute queue accounted for 39.1 percent of submitted jobs, but only 5.4 percent of the User Node Time. The table also shows that the proportion of serial jobs increases as the time limit of the queue increases, as does the volume of serial User Node Time. In fact, for

Memory	Number of Jobs		User Node Hours	
	Serial	Parallel	Serial	Parallel
128MB	8890	13297	19327.022	193230.538
256MB	1710	117	9462.956	12695.560
512MB	1618	20	4476.585	40.157
1024MB	220	0	728.871	0.000
2048MB	136	0	821.547	0.000

Table 3: Number of Jobs and User Node Time by Requested Memory.

the production queues (6, 12, and 18 hours), serial jobs are twice as numerous as parallel ones. This highlights one particular use of the SP2 as a powerful computational engine for large serial jobs.

One can make two other observations about this data:

- Despite significant serial use of the system, parallel User Node Time dominates for all queues;
- The 6, 12 and 18-hour queues account for an overwhelming 81.5 percent of the load, showing that the SP2 is being used effectively for production work.

3.2 Analysis by Requested Memory

Another interesting dimension to the serial/parallel breakdown is the split as a function of requested memory. Table 3 shows the proportion of serial and parallel jobs and User Node Time. From this table one can see that the vast majority of work is done by jobs requesting 128 MB of memory (85.3 percent of the jobs, 88.3 percent of the User Node Time). Another important trend is that serial work accounts for an increasingly larger share of the work as the node memory size increases.

There are several reasons for this phenomenon. One factor is that there are fewer nodes having larger memory (see Table 1). This, in effect, discourages parallel use because if one requests a large memory resource, LoadLeveler will guarantee that ALL nodes have that resource. Since such resources can be scarce, the user would face a long wait time before they would be freed up.

There is another important reason, though, for the large memory characteristics of serial jobs. Prior to the installation of the SP2, the computational workhorse of the Theory Center was an IBM ES/9000. This machine had a large physical memory and even larger extended memory; researchers made use of these features in writing their programs. An analysis of the ES/9000 workload done a year ago indicated the need for large amounts of real memory on SP2 nodes, since paging under AIX is very inefficient for codes with large working sets. The resulting SP2 workload bears out the analysis.

Node Type	128MB	256MB	512MB	1024MB	2048MB
Percent Requested	33.68	23.77	12.00	10.16	45.82

Table 4: Fraction of Node Time Requested, Categorized by Physical Memory

It is not surprising that the majority of work on the SP2 requests smaller amounts of memory – 82 percent of the batch nodes have 128 MB of memory. To adjust for this factor, Table 4 shows the fraction of a given type of node's available time that was requested. (Please note that this does not represent how the load was actually distributed by LoadLeveler. For example, a parallel job requesting 128 MB of memory may actually be assigned several 256MB nodes if all 128 MB nodes are busy.) This tells a somewhat different story. The single 2048 MB node is busiest, being active nearly half the time servicing serial jobs. The 128 MB nodes are active about 35 percent of the time, with a decrease in relative activity as the node memory increases to a low of 12 percent utilitzation for the 4 1048 MB nodes. It is not clear why the 512 MB and 1024 MB nodes are relatively unused by parallel jobs. It may be that there is demand for these nodes, but use is being discouraged because there are so few of them. On the other hand, there may be no such latent demand.

3.3 Analysis by Job Duration

Still one more way to analyze the data is based on the length of time (wall-clock time) the job ran, i.e., job duration. Figure 1 displays the relative number of serial and parallel jobs as a function of job duration, which is aggregated by half-hour increments. Figure 2 gives the same kind of breakdown for User Node Time.

As is evident in Figure 1, the majority of jobs (73.8 percent) run in 1 hour or less. All jobs from the 15-minute queue fall in this category, as do shorter-running jobs from the other queues. DOA jobs from all queues also contribute. From that point on, there is a general trend toward fewer, longer-running jobs, although this trend is not uniform.

One can also observe that the proportion of serial jobs increases as job duration increases. This is not surprising (and is good news, in fact) because one of the major inducements to parallelize an application is to reduce job run time.

Figure 2 tells a somewhat different story. While more User Node Time is spent in jobs running an hour or less than in any other bracket of time, it represents only 12.6 percent of the total. Large amounts of User Node Time are accumulated by jobs running between 4 and 9 hours. The spike at 19 hours reflects jobs in the 18-hour queue that were automatically terminated by LoadLeveler. Such jobs periodically write restart files so that they may begin reprocessing near the point of termination. One can also observe that, unlike in

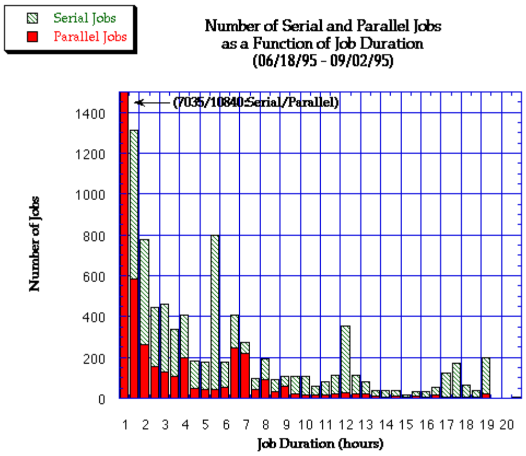

Figure 1: Number of Jobs as a Function of Job Duration.

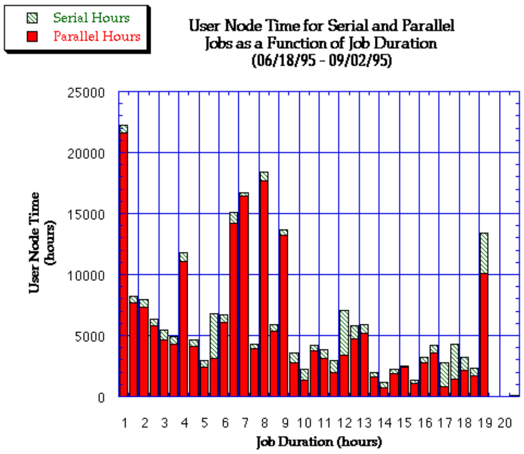

Figure 2: User Node Time as a Function of Job Duration.

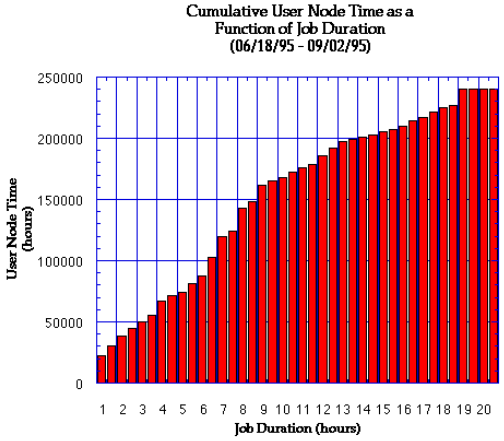

Figure 3: Cumulative User Node Time as a Function of Job Duration.

Figure 1, serial user node hours do not dominate for longer-running jobs. This is as expected – one 16-way parallel job running for 19 hours generates 16 times as much User Node Time as does a corresponding serial job.

Figure 3 shows a cumulative distribution of User Node Time. One can see the relatively steep slope between 4 and 9 hours, with a slight leveling off beyond that point. From this figure one can observe that the median value is approximately 8 hours. This means that half the User Node Time comes from jobs that run in 8 hours or more, half from 8 hours or less.

Figures 4, 5 and 6 show corresponding data for jobs in the 15-minute queue. We look at this queue in detail because its purpose is to support program development and debugging jobs, whereas the other queues are targeted for production work. Figure 4 exhibits a clear, exponential-looking, dropoff on the number of jobs as job duration increases. In contrast to the general workload in the previous set of figures, the proportion of serial jobs decreases as the job duration increases. One hypothesis for this behavior is that serial jobs can use this queue strictly for debugging and small test runs, while parallel jobs can get some production work done.

Figure 5 shows a varied distribution of User Node Time across the range of job duration times for this queue. Jobs which run in 1 minute or less comprise 39.6 percent of the jobs, yet only 1.7 percent of the User Node Time. It should be noted that User Node Time is overwhelmingly parallel (98.4 percent). This emphasizes the fact that the 15-minute queue is being used for real program development and debugging, not routine tasks which are much more easily done on the interactive portion of the SP2.

Figure 6 displays the cumulative User Node Time for the 15-minute queue.

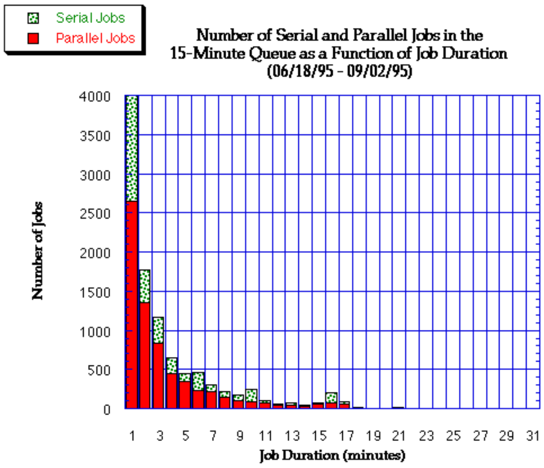

Figure 4: Number of Jobs as a Function of Job Duration in the 15-Minute Queue.

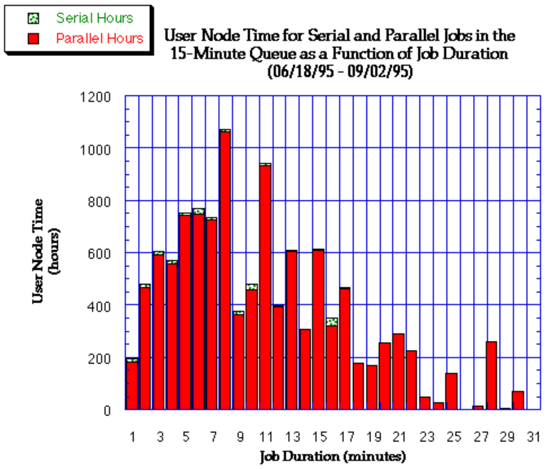

Figure 5: User Node Time as a Function of Job Duration in the 15-Minute Queue.

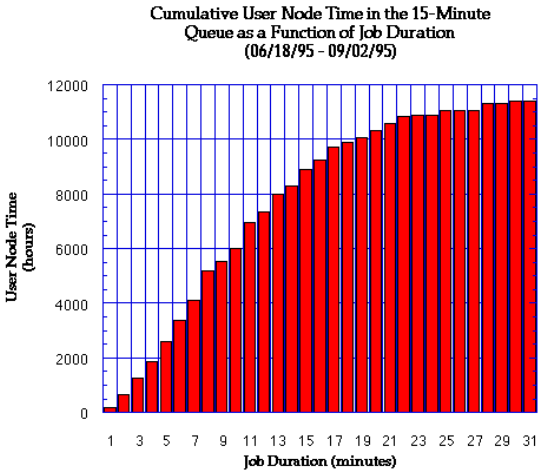

Figure 6: Cumulative User Node Time as a Function of Job Duration in the 15-Minute Queue.

From it one can see that the median point is 8 minutes, indicating that the queue is being used effectively.

3.4 DOA Analysis

As was mentioned in the introduction, there was no clearly discernible characteristic of DOA jobs that would allow us to distinguish them from short-running, successful jobs. In looking at job duration as a function of number of processors requested, we found no clustering of data at the low end of the spectrum. We did observe, though, that the minimum job duration increased as the number of processors increased. We attempted, then, to establish a minimum job duration of a successful job for a given number of processors. This was done by submitting a series of "Hello, world" parallel jobs over a range of processors, and selecting the shortest job duration for each category.

With this baseline, we define DOA jobs as those whose job duration is less than the successful-job minimum for the given number of processors. The results appear in Table 5. Under this somewhat generous definition, DOA activity constitutes a noticeable fraction of those jobs running in 1 hour or less – 15.5 percent of the jobs and 13.4 percent of the User Node Time. In light of the entire workload, DOA jobs still account for 11.4 percent of jobs submitted, but only a bit more than 1 percent of the User Node Time. This confirms the hypothesis that the number of such jobs is noticeable, reflecting program development and debugging work, but their impact on the load is negligible.

Serial Jobs	773
Parallel Jobs	2004
User Node Time (hours)	2948.03

Table 5: Dead-on-Arrival Statistics

4 Wait Time Analysis

During the period when data was being collected, the batch nodes were busy about 40 percent of the time. One would expect, then, that wait times would be consistently small. Yet in a highly parallel, heterogeneous system, with a wide range of user-requestable resources, it is possible that some may become scarce, leading to long tails in the waiting time distribution.

The key resources and LoadLeveler constructs that could potentially impact wait times include:

- Down times
- LoadLeveler queue
- Requested memory
- Number of processors requested

System downs have an obvious affect on wait times. The SP2 was relatively stable throughout the period when trace data was being collected. The only extended outages came from scheduled down times. In the wait time analysis, we subtracted out the time for scheduled downs for those jobs submitted prior to a down but whose execution did not begin until after.

The submitted queue will affect the wait time of jobs because, as was mentioned above, the 15-minute queue receives a preferential initial priority. Thus, if the workload were uniform over the queues, one would expect to find a lower average wait time for the 15-minute queue and another, slightly higher, value for the remaining queues.

Table 6 shows the results of our analysis. One positive note is that, with the exception of the 15-minute queue, the wait times for each queue are substantially less than the maximum running time allowed. The best performance is shown by the 18-hour queue where the average wait time is only 110 minutes, or 10 percent of the maximum running time allowed. This is especially notable because the 18-hour queue accounts for over 32 percent of the total User Node Time (see Table 2).

There are some surprising results, however. For example, the mean wait time for the 15-minute queue is *longer* than for the 3 and 6-hour queues. Among the longer-running queues, the wait times for the 3 and 6-hour queues are similar, the 12-hour queue somewhat longer, but much longer for the 18-hour queue.

Queue Class	Job Type	Mean	Std. Dev.	C.V.
15-Minute	Serial	20.7	114.6	5.5
	Parallel	26.0	110.7	4.3
	All Jobs	24.3	112.0	4.6
3-Hour	Serial	16.4	98.3	6.0
	Parallel	28.2	114.3	4.1
	All Jobs	22.5	107.1	4.8
6-Hour	Serial	16.3	92.5	5.7
	Parallel	25.4	94.2	3.7
	All Jobs	19.1	93.1	4.9
12-Hour	Serial	44.1	182.4	4.1
	Parallel	36.8	129.5	3.5
	All Jobs	40.7	159.9	3.9
18-Hour	Serial	116.1	394.4	3.4
	Parallel	76.2	291.8	3.8
	All Jobs	109.9	378.1	3.5

Table 6: Average Wait Times (in minutes) as a Function of Queue Class

The long wait times for the 15-minute queue are surprising since this queue accounts for only 5 percent of the User Node Time (see Table 2). The extremely high coefficient of variation suggests that the wait time distribution is highly skewed. A more detailed look at the data confirmed this hypothesis. We discovered that:

- The median wait time is 7 seconds;
- Nearly 88 percent of the jobs have a wait time less than the mean.

We took a closer look at the LoadLeveler traces for the 10 longest waiting jobs in this queue (between 22 and 64 hours). Most were parallel, with a range of processors requested, while some were serial. One requested licensed software which runs only on selected nodes. None were submitted during times of heavy load. A few traces indicated LoadLeveler instabilities. In general, we could find no characteristics of the jobs, nor of the resources they requested, that would explain these inordinately long wait times. Some possible explanations include:

- LoadLeveler instabilities
- Jobs waiting because the user already has 4 active jobs
- Action taken by users or systems staff to put the job on hold

Table 7 displays wait times as a function of requested memory. One can see that the shortest waiting time is for 128 MB nodes, which are most numerous.

Memory Class	Job Type	Mean	Std. Dev.	C.V.
128 MB	Serial	26.4	147.9	5.6
	Parallel	29.5	124.7	4.2
	All Jobs	28.2	134.5	4.8
256 MB	Serial	60.2	212.8	3.5
	Parallel	62.6	201.2	3.2
	All Jobs	60.4	212.0	3.5
512 MB	Serial	47.1	175.4	3.7
	Parallel	39.1	103.9	2.7
	All Jobs	47.0	174.7	3.7
1024 MB	Serial	55.4	146.7	2.7
	Parallel	0.0	0.0	0.0
	All Jobs	55.4	146.7	2.7
2048 MB	Serial	592.1	1123.3	1.9
	Parallel	0.0	0.0	0.0
	All Jobs	592.1	1123.3	1.9

Table 7: Average Wait Times (in minutes) as a Function of Memory Requested

The 256, 512 and 1024 MB node types have slightly longer wait times, whereas the single 2048 MB node has a far longer one. Thus there is a correlation between the number of each node type and wait time. The wait times are not consistent, however, with relative load on each node type (see Table 4). Once again the coefficients of variation are extremely large, attributable to a few jobs with artificially long wait times that do not reflect inherent characteristics of the job themselves or the resources they are requesting. We will continue to look at the data (or for other sources of data) that will help us better understand these phenomena.

Figure 7 shows the average wait time based on the number of processors requested. There is a modest correlation between wait time and number of processors, but the large variance suggests that, as before, the data demands more careful scrutiny.

5 Inter-arrival Analysis

Of particular interest to the job scheduling and resource management community is the inter-arrival time between jobs. Figures 8 and 9 describe the job submittal rate as a function of time and the amount of work generated by those jobs.

As one would expect, the job submittal rate is higher during daytime hours for both weekdays and weekends (see Figure 8). The weekday rate appears to be about 3 times that of the weekend rate. The apparent outlier for weekdays at

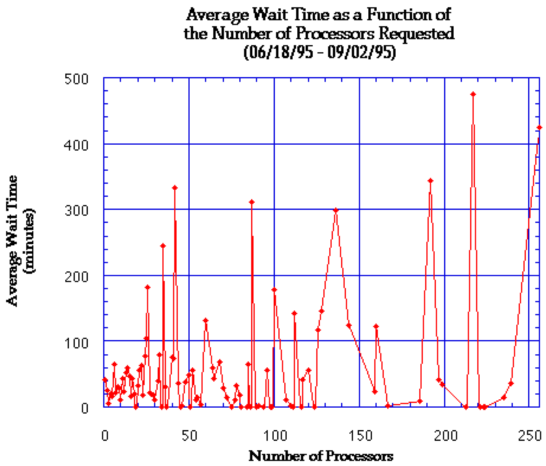

Figure 7: Average Wait Time as a Function of the Number of Processors Requested.

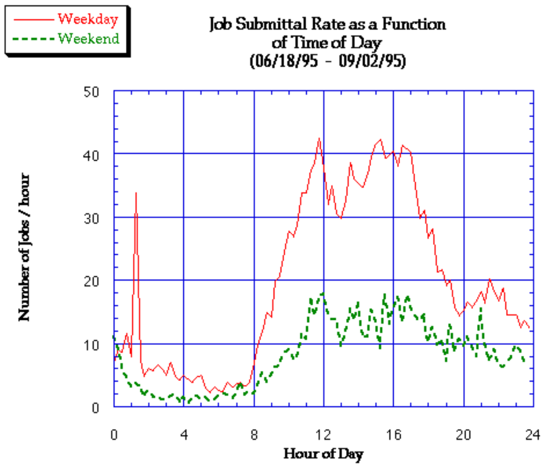

Figure 8: Job Submittal Rate as a Function of Time of Day.

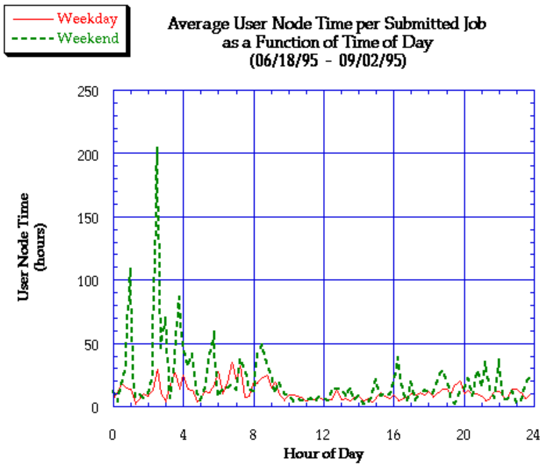

Figure 9: Average User Node Time per Submitted Job as a Function of Time of Day.

1:00 AM is not an error; on June 21, one user submitted over 300 small parallel jobs at one time to run regression tests. Figure 9 highlights the difference between weekday and weekend load. In general, the User Node Time per job is greater on the weekend, significantly so during off-hours. This confirms a pattern of users to submit large jobs during times when the machine is more lightly loaded, thereby getting faster turnaround.

A closer look at Figure 8 shows:

- A period of low activity from midnight to 8AM
- A consistent increase in submittals from 8AM until noon, where it reaches its maximum value
- A small decrease in early afternoon followed by a gradual increase to the maximum value at about 4PM
- A consistent decrease till 8PM
- A gradual tapering off during a period of low activity from 8PM to 8AM

It was natural, then, to differentiate the inter-arrival times by:

- Weekday: Monday through Friday, 8AM to 8PM
- Weekend: Saturday 12AM through Monday 12AM
- Weeknight: Remainder of time

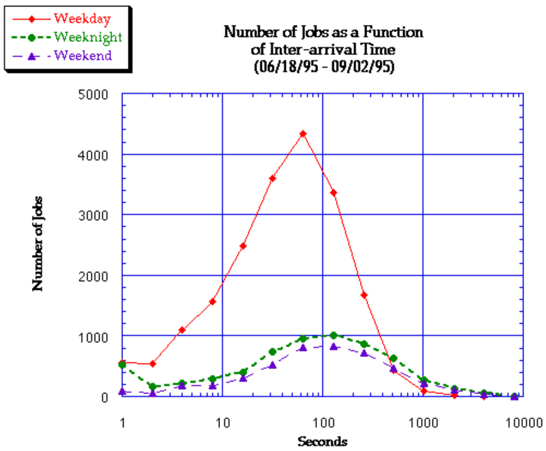

Figure 10: Number of Jobs as a Function of Inter-arrival Time.

Time Period	WeekDay	WeekNight	WeekEnd
Number of Data Points	19714	6205	4589
Mean	114.137	352.269	388.919
Standard Deviation	201.260	877.397	991.756
Coefficient of Variation	1.763	2.491	2.550

Table 8: Inter-Arrival Time Statistics (in seconds).

The graphical results can be seen in Figure 10 and a statistical description in Table 8. In the figure, inter-arrival times are binned by powers of 2; for example, approximately 4300 jobs had an inter-arrival time of between 32 and 64 seconds. Please note that the X-axis is plotted on a log scale.

Nearly 2/3 of the jobs were submitted during weekday hours. Most weekday jobs have an inter-arrival time of between 16 and 128 seconds, with a mean of 114 seconds; very rarely is there more than a 15-minute gap between arrivals. The coefficient of variation is greater than 1.7, indicating that the inter-arrival times do not follow an exponential distribution, as is sometimes assumed in modeling studies.

The weeknight and weekend behaviors are very similar to each other. As expected, the number of jobs submitted are far fewer than during the weekday. The graphs are much smoother, showing a peak at about 100 seconds. The tail, reflecting times of relative inactivity, is more pronounced. The mean inter-arrival time is over triple that of the weekday one, and the distribution is more highly skewed, as one would expect with a more extensive tail.

It is useful to compare these results with those of Feitelson and Nitzberg [5].

Mean	9.248
Standard Deviation	46.786
Coefficient of Variation	5.059

Table 9: Service Demand Statistics (in hours)

There is remarkable similarity in the plots of job submittal rates, both evincing a "double maximum" at noon and 4PM. Both show significantly larger jobs submitted between midnight and 4AM, although our data does not show a similar pattern from 8PM to midnight, whereas theirs does. There are differences, though, in the inter-arrival time data. Their data shows more activity on the weekend than on weekday nights, while our data reflects very little difference. More striking is the difference in coefficient of variation for the weekday submittals. Both are greater than 1, indicating a non-exponential distribution, but their value is twice as large as ours.

6 Application Metrics

One important metric of a workload is the *service demand*. This is defined as the cumulative processing requirements of a particular job. This quantity is a key parameter in the analyses of various scheduling policies [6, 10]. In our terminology, this is User Node Time. Table 9 gives the statistical breakdown of this metric, showing a mean value of over 9 hours. The data is very skewed, with a coefficient of variation greater than 5. This value is consistent with that reported in [3] for the CM-5 at the University of Wisconsin. In some ways, our figure is more striking because the underlying workload is limited to batch jobs, and does not take into account interactive work done on other nodes of the SP2. If the interactive work were incorporated into the data, one would expect an even greater coefficient of variation because the mean service demand of interactive jobs would be far less than the 9 hour average for batch jobs.

Another metric of interest is *average parallelism*. One way of defining this metric is:

$$\frac{\sum_{i=1}^{NJOB} NPROC_i}{NJOB}$$

where NJOB is the number of jobs and NPROC(i) is the number of processors requested by the ith job. For our workload, this Job-based average parallelism is 17.7.

Unfortunately, this definition of average parallelism does not take into account the length of time a job runs. If jobs using more processors have a shorter job duration, then this measure will overstate the degree to which actual parallel work is being done. Conversely, if jobs using more processors run longer, as was found in [5], then this measure will understate it.

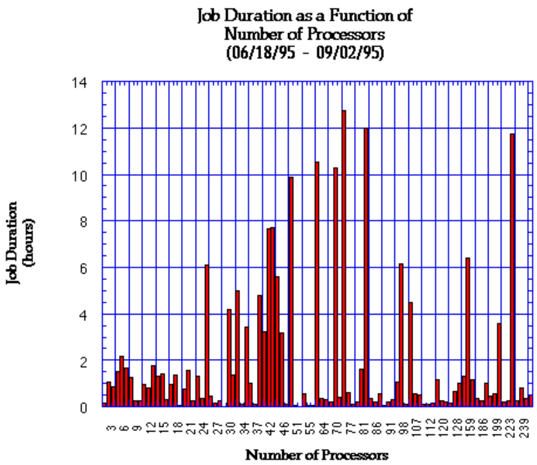

Figure 11: Job Duration as a Function of Number of Processors.

We define, then, Usage-based average parallelism as:

$$\frac{\sum_{i=1}^{NJOB} NPROC_i * JobD_i}{\sum_{i=1}^{NJOB} JobD_i}$$

where JobD(i) is the job duration of the ith job. For our workload, this value is 17.9, virtually the same as the Job-based average. This suggests that jobs requesting larger number of processors run just as long as jobs requesting a smaller number. The breakdown can be seen in Figure 11, which shows an erratic pattern of job duration as a function of number of processors. From this one might expect to see a higher usage-based average parallelism since job durations can be large for some requests for a large number of processors. However, the figure is misleading in that these large job durations reflect only a few jobs; the peaks at 75, 84 and 233 processors, for example, are derived from only 1 job in each category.

7 Conclusions

Heterogeneous parallel computers like the Cornell Theory Center IBM SP2 complicate workload characterization because more variables must be taken into consideration when analyzing the data. Fortunately, LoadLeveler traces provide a great deal of useful information, making such analyses possible.

At the most general level, we found that the number of parallel and serial jobs were roughly the same, but that parallel usage was overwhelmingly greater than serial usage. Serial usage remained fairly constant over the period of time when traces were collected; parallel usage showed minor fluctuations, with a gradual increase later in the period.

128 MB nodes account for the bulk of the processing time. When one adjusts for the number of nodes having a given amount of memory, however, we found that the single 2048 MB node had the greatest demand. As for job duration, most long-running jobs were serial, but as before, parallel jobs accounted for the bulk of the User Node Time.

Wait times proved to be difficult to analyze. No matter how the wait times were categorized (by queue, by memory, by number of processors), the coefficients of variation were unexpectedly large. A closer inspection of the data showed that a few jobs were waiting an inordinately long time, little of which was due to the nature of the job or the resources requested.

Inter-arrival times were segregated by time of day. The mean inter-arrival time during weekday periods was about two minutes, with a high coefficient of variation. The mean service demand was about 9 hours with a very large coefficient of variation – over 5. Scheduling policies take these large variations into account are essential.

Note: The traces used in this study are available upon request.

References

[1] T. Agerwala, J.L. Martin, J.H. Mirza, D.C. Sadler, D.M. Dias and M. Snir, "SP2 System Architecture". *IBM Systems Journal*, Vol. 34, No. 2, 1995.

[2] D. Bailey et al., *The NAS Parallel Benchmarks*, Tech. Rep. RNR-91-002, NAS Systems Division, January, 1991.

[3] S. Chiang, R.K. Mansharamani and M.K. Vernon. "Use of Application Characteristics and Limited Preemption for Run-to-Completion Parallel Processor Scheduling Policies". In *Proceedings of ACM SIGMETRICS Conference*, 1994. 22(1): p. 33-44.

[4] R. Cypher, A. Ho, S. Konstantinidou and P. Messina. "Architectural Requirements of Parallel Scientific Applications with Explicit Communication". In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May, 1993. p. 2-13.

[5] D.G. Feitelson and B. Nitzberg. "Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860". *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, April, 1995.

[6] S. Leutenegger and M.K. Vernon. "The Performance of Multiprogrammed Multiprocessor Scheduling Policies". In *Proceedings of ACM SIGMETRICS Conference*, 1990. 22(1): p. 226-236.

[7] C. McCann and J. Zahorjan. "Scheduling Memory Constrained Jobs in Distributed Memory Parallel Computers". In *Proceedings of ACM SIGMETRICS Conference*, 1995. 23(1): p. 208-219.

[8] C. McCann and J. Zahorjan. "Processor Allocation Policies for Message-Passing Parallel Computers". In *Proceedings of ACM SIGMETRICS Conference*, 1994. 22(1): p. 19-32.

[9] V.K. Naik, S.K. Setia and M.S. Squillante. "Scheduling of Large Scientific applications on distributed Memory Multiprocessor Systems". In *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 1993.

[10] E.W. Parsons and K.C. Sevcik. "Multiprocessor Scheduling for High-Variability Service Time Distributions". *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, April, 1995.

[11] V.G.J. Peris, M.S. Squillante and V.K. Naik. "Analysis of the Impact of Memory in Distributed Parallel Processing Systems". In *Proceedings of ACM SIGMETRICS Conference*, 1994. 22(1): p. 5-18.

[12] W. Pfeiffer, S. Hotovy, N.A. Nystrom, D. Rudy, T. Sterling and M. Straka. *JNNIE: The Joint NSF-NASA Initiative on Evaluation*. San Diego Supercomputer Center Technical Report GA-A22123, July, 1995.

[13] M.E. Rosenkrantz, D.J. Schneider, R. Leibensberger, M. Shore and J. Zollweg. "Requirements of the Cornell Theory Center for Resource Management and Process Scheduling". *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, April, 1995.

[14] S.K. Setia, M.S. Squillante and S.K. Tripathi. "Processor Scheduling on Multiprogrammed, Distributed Memory Parallel Computers". In *Proceedings of ACM SIGMETRICS Conference*, 1993. 21(1): p. 158-170.

[15] M.S. Squillante. "On the Benefits and Limitations of Dynamic Partitioning in Parallel Computer Systems". *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing*, April, 1995.

[16] *IBM LoadLeveler Administration Guide*, IBM Document Number SH26-7220-02, October, 1994.