LIMITED DISTRIBUTION NOTICE
(NOT TO BE REMOVED FROM DOCUMENT):

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communication and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860

Dror G. Feitelson
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
feit@watson.ibm.com

Bill Nitzberg*
NASA Ames Research Center
Moffett Field, CA 94035
nitzberg@nas.nasa.gov

Abstract

Statistics of a parallel workload on a 128-node iPSC/860 located at NASA Ames are presented. It is shown that while the number of sequential jobs dominates the number of parallel jobs, most of the resources (measured in node-seconds) were consumed by parallel jobs. Moreover, most of the sequential jobs were for system administration. The average runtime of jobs grew with the number of nodes used, so the total resource requirements of large parallel jobs were larger by more than the number of nodes they used. The job submission rate during peak day activity was somewhat lower than one every two minutes, and the average job size was small. At night, submission rate was low but job sizes and system utilization were high, mainly due to NQS. Submission rate and utilization over the weekend were lower than on weekdays. The overall utilization was 50%, after accounting for downtime. About 2/3 of the applications were executed repeatedly, some for a significant number of times.

1 Introduction

Since the dawn of (computer) time, it has been recognized that performance analysis and modeling of computer systems hinges on using a representative workload [9, 1]. The approach is typically to measure the workload on a real system, analyze it, and use the obtained parameters to drive a simulation or as input for analytical modeling. While there are some measurement-based results on workload modeling for uniprocessors [12, 5], there is very little hard evidence from real measurements on parallel machines. The little work that has been done concentrates on modeling single applications, e.g. showing how a specific algorithm leads to changes in the degree of parallelism in different phases of the computation [18, 24, 20]. There is practically no work relating to the mix of different jobs that are found on parallel

*Computer Sciences Corporation, NASA Contract NAS 2-12961.

machines. This makes it hard to study and compare operating system policies for scheduling and processor allocation.

This study makes an attempt to supply such measurements. It is based on accounting traces that were maintained by the system administrators of the 128-node iPSC/860 hypercube located at NASA Ames. All the jobs run on the system during the fourth quarter of 1993 are included. The trace includes two types of records: job records and special records. Job records include a pseudo user ID (to protect privacy), a pseudo command name representing the executable file name, the number of nodes used, the runtime (in seconds), and the start time and date. NQS jobs have similar records, except for the command name that is just a serial number. System commands are identified explicitly (this includes `cat`, `cp`, `grep`, `ls`, `nsh`, `ps`, `pwd`, `rcp`, and `rm`; `nsh` is a remote shell used most often to look at CFS files). Special records include a code identifying the event (dedicated time, scheduled or unscheduled downtime, hardware or software failures, or other), the duration, and start time and date.

The next section describes the scheduling mechanisms used on the traced system. Subsequent sections present the results relating to distribution of job sizes and types, the system utilization and multiprogramming level, the distribution of runtimes and the distinction between interactive and batch jobs, the job submission rate, and user activity and application usage.

2 System Background

The iPSC/860 is the third in a series of hypercube computers from Intel. The original iPSC was a direct outgrowth of a licensing agreement with Caltech regarding technology developed for the Cosmic Cube. However, the original 8086/8087 processors were replaced by the 80286/80287 chip set. The iPSC/2 upgraded to the 80386/80387 set, and also improved the communication performance by replacing the store-and-forward mechanism with hardware routing. The iPSC/860 upgraded to the i860 RISC processor (the prototype for this machine was called the Touchstone Gamma). This machine has since been superseded by the Paragon, which replaces the hypercube interconnection topology by a mesh (the prototype for Paragon was the Touchstone Delta at Caltech). In total, a few hundreds of iPSC machines were sold, and many of them are still in use, including the 128-node machine at NASA Ames that was used in this study.

Both applications and communication libraries for the iPSC often make use of the hypercube topology. Consequently applications cannot run on arbitrary subsets of nodes, but only on subcubes, i.e. sets of nodes that form a hypercube of smaller dimension than the whole machine. Multiprogramming is achieved by using space-slicing (also known as space sharing), that is partitioning the machine into subcubes and running a different job on each (see Fig. 1). The partition sizes are always powers of two. The operating system will always allocate a subcube of 2^i nodes starting with a node whose number is a multiple of 2^i. First consecutive node allocations are tried (0 to 2^i-1, 2^i to 2×2^i-1, etc.). If this fails, other combinations are tried, e.g. nodes 0–15 and 64–79 can be used to satisfy a request

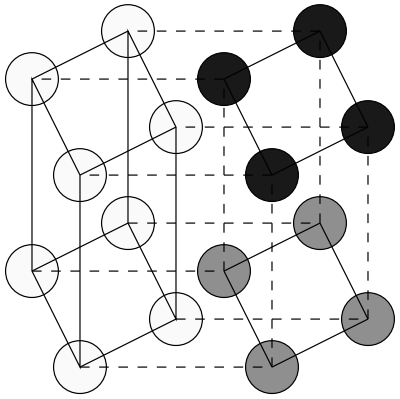

Figure 1: *Example of partitioning a 4-cube into a 3-cube and a pair of 2-cubes.*

time	*number of nodes*			
limit	16	32	64	128
20 minutes	`q16s`	`q32s`	`q64s`	`q128s`
1 hour	`q16m`	`q32m`	`q64m`	`q128m`
3 hours	`q16l`	`q32l`	`q64l`	`q128l`

Table I: *NQS queues used on the traced machine.*

for a subcube with 32 nodes. However, due to the restriction on the starting point, not all possible subcubes are recognized. For example, if a request for 32 nodes is received when only nodes 16–31 and 48–63 are free, the request will be rejected even though these nodes constitute a 32-node subcube. The system has a limit of 10 simultaneous partitions. One of these is used by the Concurrent File System (CFS), so the maximal multiprogramming level supported is 9.

Jobs can be submitted directly or through the NQS batch queueing facility. NQS is integrated with MACS (Multi-user Accounting, Control, and Scheduling utilities), which is part of the system software on the iPSC/860 [13]. To use NQS, system administrators define multiple queues with different resource limits. The combinations used on the iPSC/860 at NASA Ames are shown in Table I: there are 4 possible values for the subcube size, and 3 for the runtime. NQS also allows different priorities and limits on file sizes, but these features are not used. The MACS job-mix scheduler uses a complex algorithm to prioritize the jobs in the NQS queues, and to schedule them on the machine's batch partition (64 nodes on weekdays, 128 nodes at nights and weekends). In certain circumstances, it may even kill direct (interactive) jobs that exceed their resource limits [13].

Only queues `q16s`, `q16m`, `q32s`, and `q32m` are enabled during prime time (from 6 AM to 8 PM), effectively limiting jobs to 32 nodes and 1 hour. Any larger jobs that are submitted are queued until the other queues become active at night. The four 20-minute queues are enabled during the day on weekends. These policies are sometimes modified to accommodate user

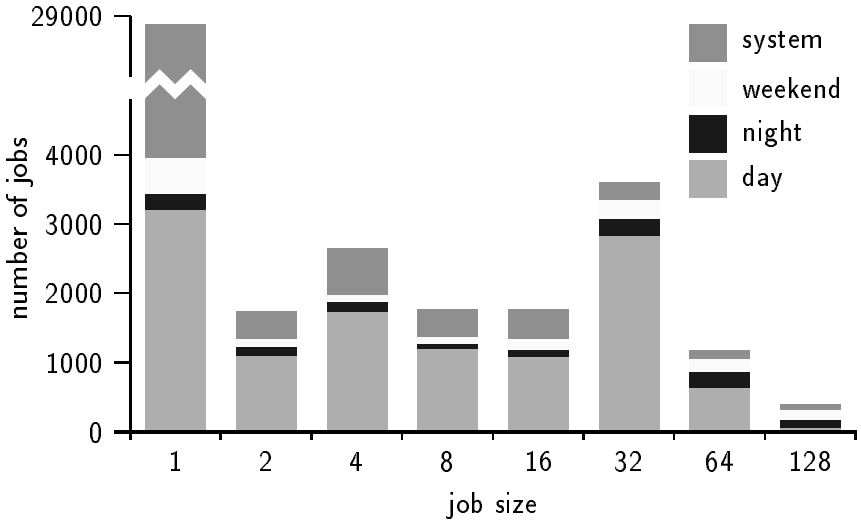

Figure 2: *Histogram of jobs with different degrees of parallelism, classified by period.*

needs. The machine can be dedicated to a certain user, effectively making it unaccessible to others. This is done manually by the system support staff.

The support staff also (manually) register downtime, so the special trace records about such events are only as accurate as manual system monitoring allows. In order to help the support staff notice system problems, a `pwd` command is executed automatically at short intervals. If it fails or hangs, the support staff is notified.

3 General Job Mix

Degrees of parallelism

A total of 42050 jobs appeared in the traces (disregarding those that ran on the service node and those that ran for 0 seconds, i.e. failed immediately). the breakup into different degrees of parallelism is shown in Fig. 2. A distinction is made between user jobs that ran in the day, night, or over the weekend, and jobs run by the system support staff.

The results are that a whopping 28894 jobs (68.7%) ran on a single node, and only 13156 (31.3%) ran on 2 or more nodes. However, 24921 of the sequential jobs (59.3% of the total) were run by the system support staff, and 25561 of them (60.8% of the total) were Unix commands. Both these numbers are dominated by the periodic `pwd` commands that are used to check if the system is up: there were 24013 such jobs (57.1% of the total). It should be noted that the system support staff also executed a significant number of parallel jobs, mostly to check system functionality.

If only the 14794 user jobs are considered, 26.9% were sequential and 73.1% were parallel. Among the parallel jobs, there was a more-or-less uniform distribution across the possible parallel job sizes, which are powers of two (corresponding to subcubes of the iPSC). However, there is a noticeably high count of 32-node jobs. This is not related to the fact that 32 is

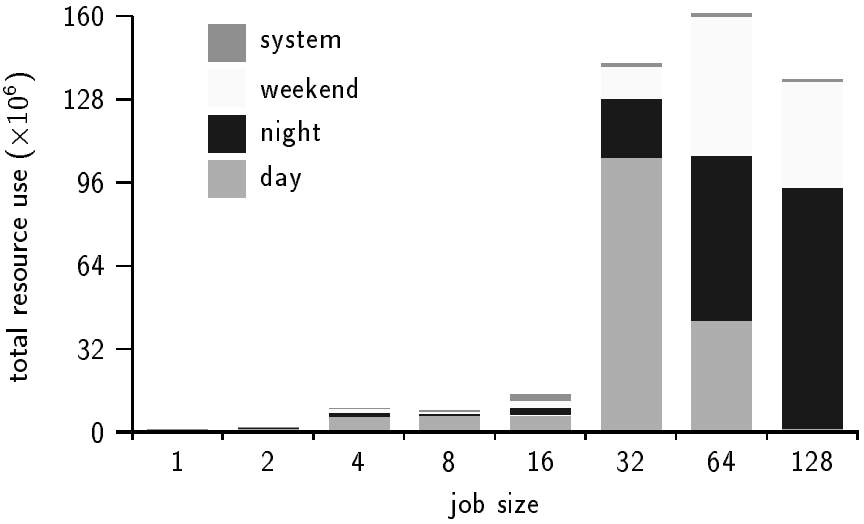

Figure 3: *Total resource use (in node-seconds) by jobs with different degrees of parallelism.*

the maximal size that can be run by NQS during the day, as shown by Fig. 4. There is also a very low count of 128-node jobs, possibly because such jobs can only run if the machine is otherwise idle. Indeed, a relatively large fraction of the 128-node jobs ran at night or over the weekend. For the other sizes, the vast majority of jobs ran during the day. When counting all user jobs, 80.4% ran during the day, 8.5% at night, and 11.1% during the weekend. Our results contradict the often-postulated bimodal distribution, where jobs are expected to require either a single node or all the nodes.

Fig. 2 presents a histogram of the *number* of jobs of each size that were traced. Fig. 3 shows a histogram of the *total resources* consumed by jobs of each size, where resources are measured in node-seconds. It turns out that all the sequential jobs together used only 0.28% of the node-seconds. The large parallel jobs, with 32, 64, and 128 nodes, used 92.5%, in roughly equal portions. Of the resources consumed by user jobs, 36.6% were used during the day, 39.6% during the night, and 23.8% on weekends. These figures should be compared to the 41.7%, 29.8%, and 28.6% of total time that fall into these three time brackets, respectively (disregarding downtime). System support jobs used 1.7% of the resources.

It is noteworthy that 32-node jobs consumed resources mostly during the day, and even 64-node jobs used about a quarter their resources during the day. This is in contrast with 128-node jobs that consumed resources mostly during the night. It was possible because the 32 and 64-node jobs can run within the 64-node batch partition that is defined during prime time. This leaves enough of the machine free for all the small interactive jobs. In fact, interactive users are encouraged to use the `clrcube` command to kill large batch jobs that get in their way [13]. More on the resource use of interactive vs. batch is given below.

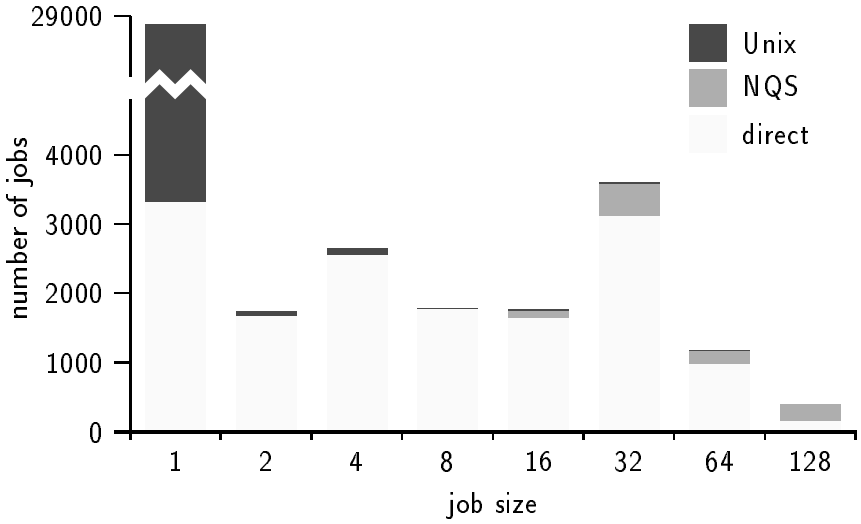

Figure 4: *Histogram of different type jobs.*

Job type

The jobs in the system can be classified into three types: Unix commands, direct user jobs, and NQS jobs. The breakdown in terms of number of jobs is shown in Fig. 4, and their resource requirements are shown in Fig. 5. These include both jobs submitted by users and jobs submitted by the system support staff.

Obviously, the vast majority of jobs were Unix jobs, mainly due to the periodic `pwd` commands. Disregarding these, we find that most jobs were submitted directly: this was 93.8% of the non-Unix jobs, whereas NQS accounted only for 6.2%. There were no NQS jobs

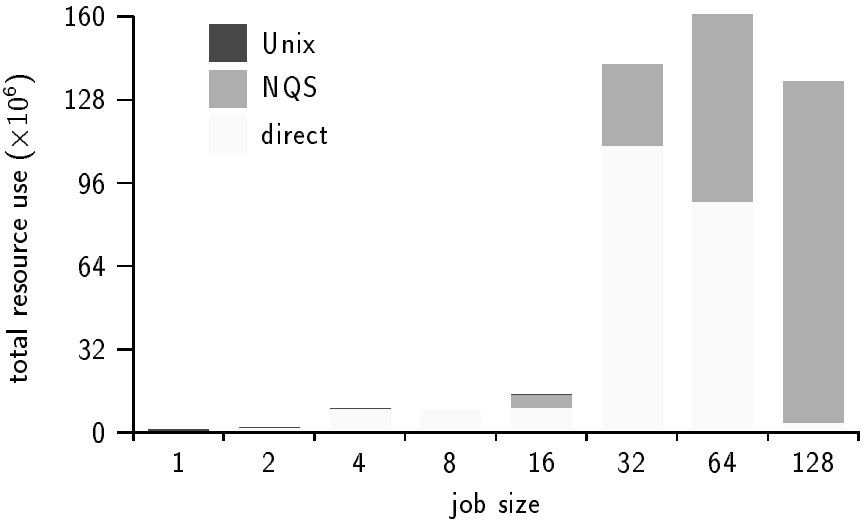

Figure 5: *Resource usage (in node-seconds) by different types of jobs.*

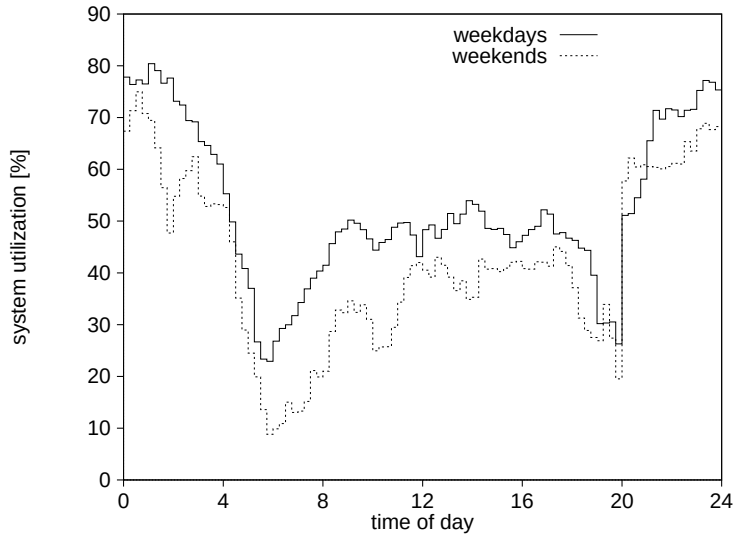

Figure 6: *System utilization as function of time of day.*

with less than 16 nodes. Surprisingly, there were some cases in which Unix commands were run on multiple nodes, and even one case where `pwd` was run on all 128 nodes. The division of resources is quite different from the job count. Nearly all the resources used by 128-node jobs, and large chunks of the resources used by 32 and 64-node jobs, were consumed by jobs submitted through NQS. In total, NQS jobs accounted for 50.5% of the resource usage, direct jobs for 49.3%, and Unix jobs for 0.2%.

4 System Utilization

Fig. 6 shows the system utilization as a function of time of day. Two domains are easily distinguishable. During the workday, the utilization approximately follows the submission rate (see Fig. 12), as the load is primarily composed of small jobs. During the night, the utilization is significantly higher than during the day. This is the result of running fewer jobs, where each one has a higher degree of parallelism and runs longer on average. There is a sharp rise in utilization at 8 PM, due to the NQS night shift. The transition from night to day is more gradual, as NQS might runs out of queued jobs before the night is over. This daily cycle should be contrasted with the cycle observed on interactive workstations, which displays very low utilization at night [16].

After accounting for downtime, the overall average utilization according to the traces was 52.4% on weekdays, 42.1% on weekends, and 50.0% overall. This is considerably lower than the 80% utilization reported for the Touchstone Delta, also accounting for downtime [22]. However, it is in line with the results of [17], which are specific to subcube allocation. It is much higher than the 9% average utilization measured for workstations [16].

The reason for the difference from the Delta is that in general there are not enough small jobs to fill in the gaps left by large ones. In hypercube machines, jobs require partitions that

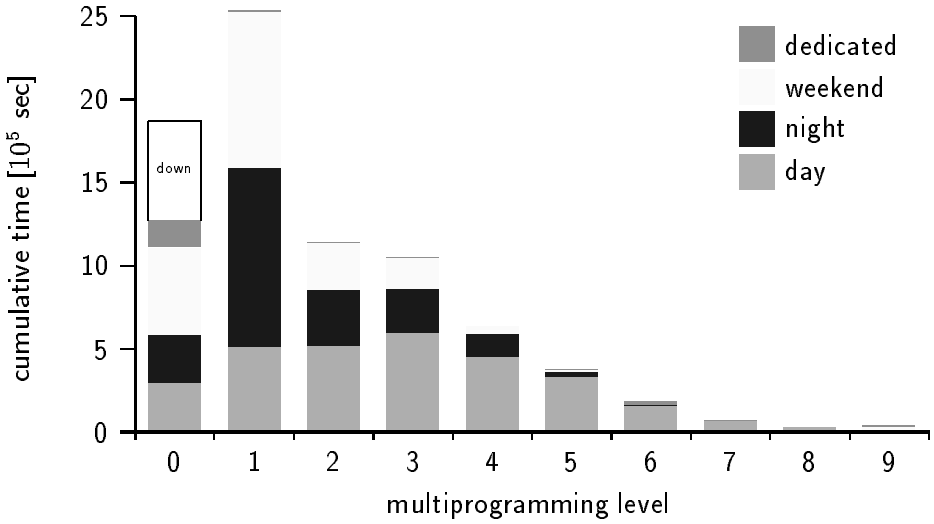

Figure 7: *Cumulative time spent in each multiprogramming level.*

are subcubes. Thus if a 4-node job is running, a 128-node job is blocked. If a 2-node job and a 64-node job are running, another 64-node job cannot run. On a mesh like the Delta, jobs can usually run on different mesh sizes with different proportions, and with less restrictions. Thus if a 128-node partition is not available, 124 nodes can be used instead. If 64 nodes are not available, a partition of 60 can still be used.

In principle, the utilization of hypercubes can be boosted if time-slicing is used [8]. However, this requires gang scheduling to be implemented, and increases the requirement for memory at each node, because multiple applications have to be memory-resident at the same time.

Multiprogramming level

On uniprocessors, multiprogramming is used to improve system utilization by overlapping the execution of jobs with complementary requirements. The iPSC can be multiprogrammed by partitioning it into subcubes, and letting different jobs run on different subcubes. A histogram of the multiprogramming level is shown in Fig. 7, with a distinction between different periods. Dedicated time is distinguished from general availability. A multiprogramming level of zero indicates idle time.

The machine was idle for 16.0% of the time, and down for another 7.5% of the time (it is possible that some of the idle time was actually down time, because down time was registered manually when the system operators noticed it). The machine ran a single job for 32.0% of the time. Of this, 13.4% was spent running the 128-node jobs, which cannot be multiprogrammed with another job. This was the dominant level of multiprogramming at night and on weekends, but on weekends the machine was also idle a significant amount of time. During workdays, the most likely multiprogramming level was 3 jobs, by a small margin. The maximum observed was a multiprogramming level of 9 concurrent jobs, which

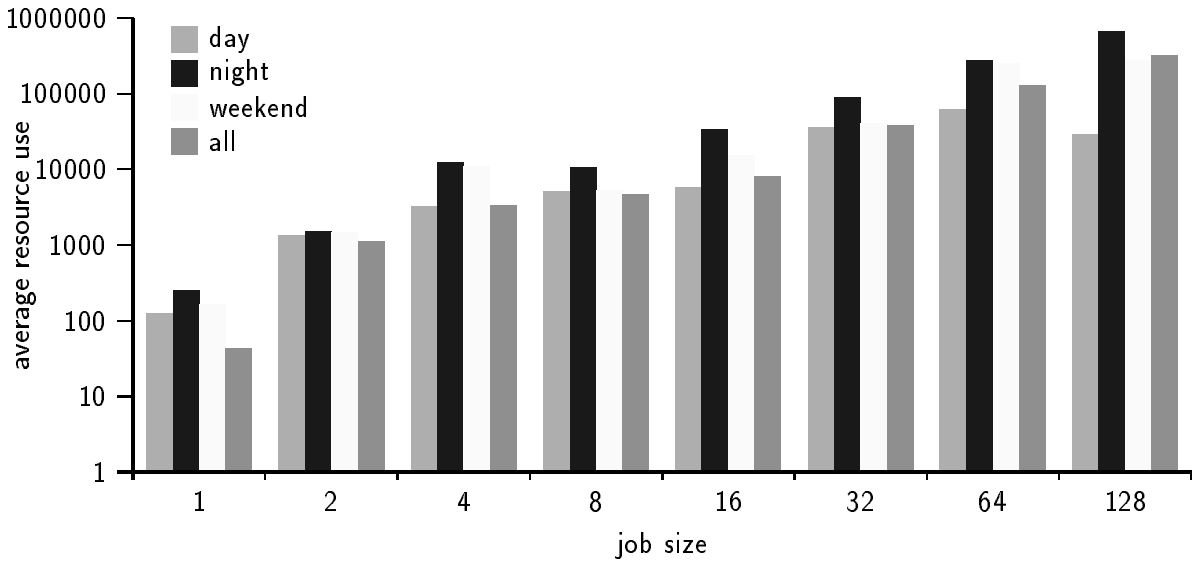

Figure 8: *Average resource use by jobs with different degrees of parallelism.*

matches the system limit. Dedicated time was mostly wasted, but it is possible that some of the time dedicated to Intel personnel was actually downtime.

It should be noted that a package for inter-partition communication has been developed at NASA Ames, and is used on this system. This package allows distinct jobs, running concurrently on disjoint subcubes, to communicate with each other [3]. Use of this package obviously boosts the measured multiprogramming level, because sets of jobs that are actually part of the same application are measured as independent jobs.

Downtime

As noted above, the system was down for 7.5% of the time during the three months that were traced, and it is possible that some idle time and dedicated time were actually downtime. A large part of the downtime (6.4%) was attributed to the scheduled six-day annual shutdown from December 24 to 29 (the normal shutdown is about one day, mainly for air conditioning maintenance. In 1993, chiller problems required an extended shutdown). Other downtime was attributed to 69 software failures, 7 hardware failures, and one case of tripping over the power line. Thus the average uptime was just over one day. the longest period of continuous availability was 4.8 days.

5 Runtime Distribution and Resource Use

Fig. 8 shows the average resource usage (i.e. the average number of node-seconds consumed) for user jobs of different sizes, using a logarithmic scale. Generally speaking, the average resource usage grows with the number of nodes used by the job. Night-time jobs require up to an order of magnitude more resources than daytime jobs with the same degree of parallelism.

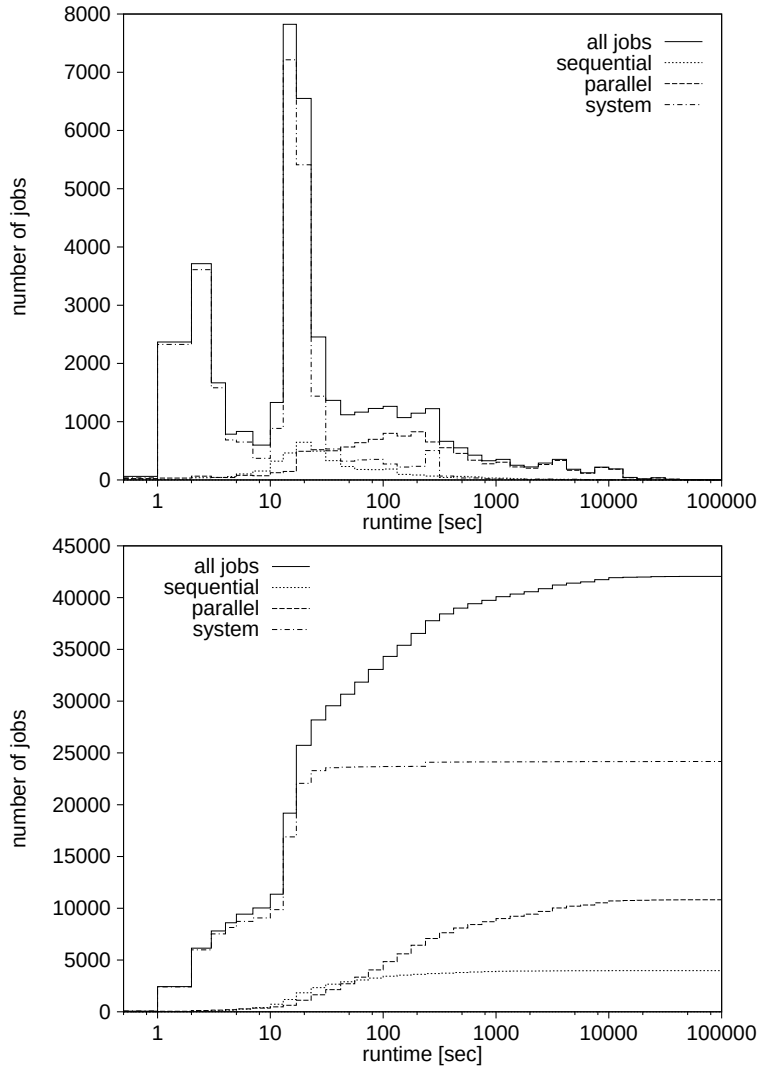

Figure 9: *Pointwise and cumulative distribution of runtimes.*

The "all" column is an average over all jobs at all times, including system support. This is the reason for the rather low value for single-node jobs.

The evidence from Fig. 8 suggests that different job sizes have approximately the same mean execution time. However, closer inspection shows some interesting deviations. Fig. 9 shows the distribution of the runtimes of all traced jobs, and also differentiates between sequential and parallel jobs. It is important to distinguish between user sequential jobs and system support jobs, which are also mostly sequential. The system jobs have two very large peaks at about 2 seconds and about 12 seconds. User sequential jobs have a much wider distribution, with the peak at about 12 seconds. User jobs in general lack weight in the region of low runtimes. In fact, very few user jobs ran for less than 10 seconds. The peak of the distribution for parallel user jobs is around 150 seconds or so, and it has a rather long

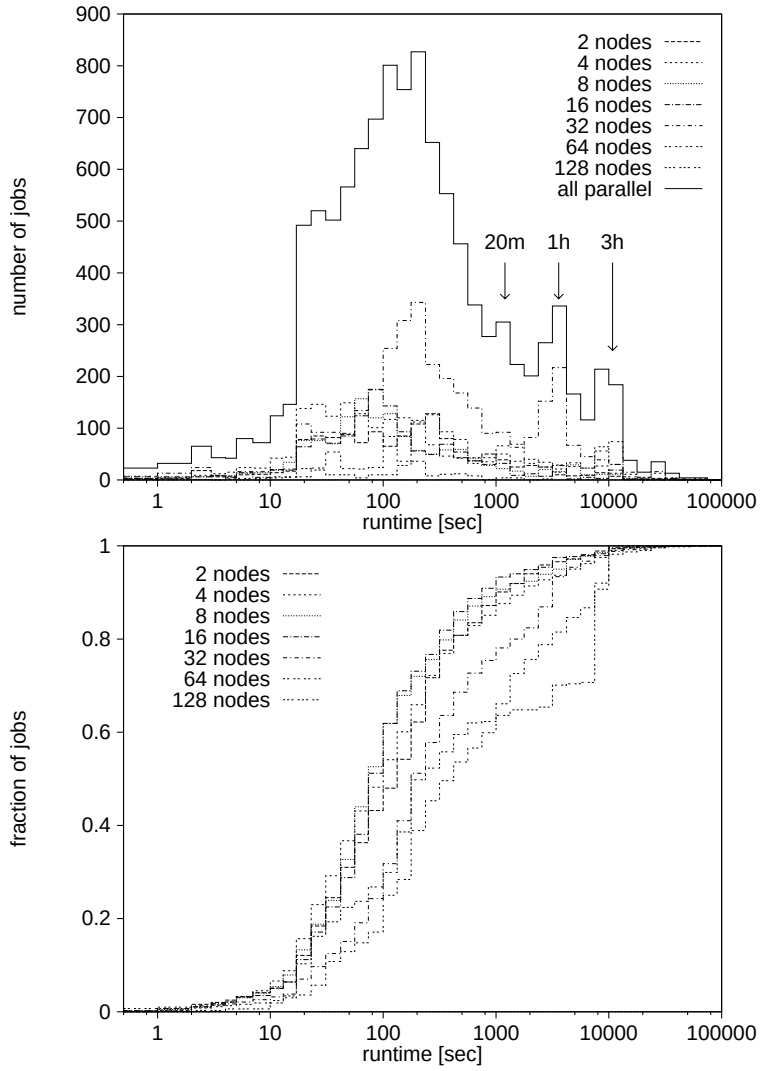

Figure 10: *Distribution of runtimes for jobs with different degrees of parallelism, and normalized cumulative distributions of runtimes.*

tail.

The distributions for jobs with different degrees of parallelism are shown in Fig. 10. The differences are not very clear, but there is some tendency for larger jobs to run longer. This is easier to see in the cumulative distribution plot: The plots for jobs using 2, 4, 8, and 16 nodes are to the left and above those for jobs using 32, 64, and 128 nodes. A similar classification is seen when precise values for the average runtimes are computed, as shown in Table II.

Three small peaks appear to the right of the main peak in Fig. 10. These correspond to the runtime limitations of different NQS queues (20 minutes, 1 hour, and 3 hours). The 1 and 3 hour peaks probably include jobs that were killed by NQS when their time ran out.

job size	*average runtime*	*standard deviation*	*coefficient of variation*
1	140.6	736.0	5.2
2	714.2	2422.3	3.4
4	1116.7	4171.5	3.7
8	705.2	2344.3	3.3
16	569.3	1970.9	3.5
32	1305.3	3311.6	2.5
64	2350.8	4155.8	1.8
128	3280.1	4408.1	1.3
system	44.1	438.0	9.9

Table II: *Runtime statistics for jobs with different degrees of parallelism.*

The 20 minute peak is actually unrelated to NQS, as shown in Fig. 11.

Table II shows the mean, standard deviation, and coefficient of variation[1] of the measured runtimes. As can be expected from the relatively wide distributions, the standard deviations have high values. Consequently, the coefficient of variation is always larger than 1, lending credence to a hyperexponential model, as is typically expected [15]. The mean is especially small, and the coefficient of variation especially large, for system support jobs. due to the large number of these jobs, it is important to exclude them when investigating statistics of user activity.

Given the lack of previously published results relating to the distribution of runtimes for parallel jobs, performance studies have assumed various "reasonable" distributions. These are related to the model of how jobs scale with increased available parallelism. Three models have been proposed [27, 25]:

- *Fixed work.* This assumes that the work done by a job is fixed, and parallelism is used to solve the same problems faster. Therefore the runtime is assumed to be inversely proportional to the degree of parallelism. This model is the basis for Amdahl's law [2].

- *Fixed time* [10, 11]. Here it is assumed that parallelism is used to solve increasingly larger problems, under the constraint that the total runtime stays fixed. In this case, the runtime distribution is independent of the degree of parallelism.

- *Memory bound* [26]. If the problem size is increased to fill the available memory on the larger machine, the amount of productive work typically grows at least linearly with the parallelism. The overheads associated with parallelism always grow superlinearly. Thus the total execution time actually increases with added parallelism.

[1]The coefficient of variation is the ratio of the standard deviation to the mean. In the exponential distribution, it has a value of 1.

threshold [sec]	*jobs below threshold*	*resource use*
1	0.2%	0.0002%
3	1.1%	0.001%
10	5.9%	0.007%
31	26.9%	0.12%
100	49.6%	0.69%
316	72.8%	3.73%
1000	85.0%	8.30%
3162	92.2%	19.47%

Table III: *Division of resources between interactive and batch jobs (user jobs only).*

The first two models are typically used in studies of multiprocessor scheduling [21, 19, 17, 6]. Our results indicate that the third is closer to the truth (at least for the measured system). An independent study of 23 CFD applications (which is the typical application domain for the studied system) also found that large jobs tend to run longer, despite the fact that they use more processors [23].

Interactive vs. batch

The distribution of runtimes also allows one to distinguish between interactive and batch jobs. For example, jobs taking less than 10 seconds could be called interactive, and those taking more time called batch. Table III shows the percentage of jobs of each type, and their aggregate resource use, for different threshold values. The results are that for reasonable thresholds, in the range of tens of seconds, up to half the jobs are interactive, but they use less than 0.7% of the resources. This is because so many of the short jobs use only one node, whereas most of the long jobs have high degrees of parallelism. If, however, we decide that a job taking around 15 minutes is still considered interactive, the results are that nearly 90% of the jobs are interactive, and they consume somewhat less than 10% of the resources.

Another distinction that is often made is based on the job submission mechanism, where direct jobs are considered interactive and NQS jobs are batch. To check this dichotomy, we plot the runtime distribution for the two types in Fig. 11. The results are that the bulk of direct jobs indeed have runtimes of between 20 and 200 seconds, whereas NQS jobs had two peaks at 1 and 3 hours (corresponding to the queue limits). However, the total number of NQS jobs was small, so the number of long running direct jobs was larger than the number of NQS jobs. In addition, direct jobs were not limited to 3 hours, so the longest running jobs were direct jobs. It is interesting to note that was no noticeable NQS peak at 20 minutes, indicating that the 20 minute queues were either underutilized, or used for really short jobs that did not reach their time limit. Thus NQS was also used for interactive jobs.

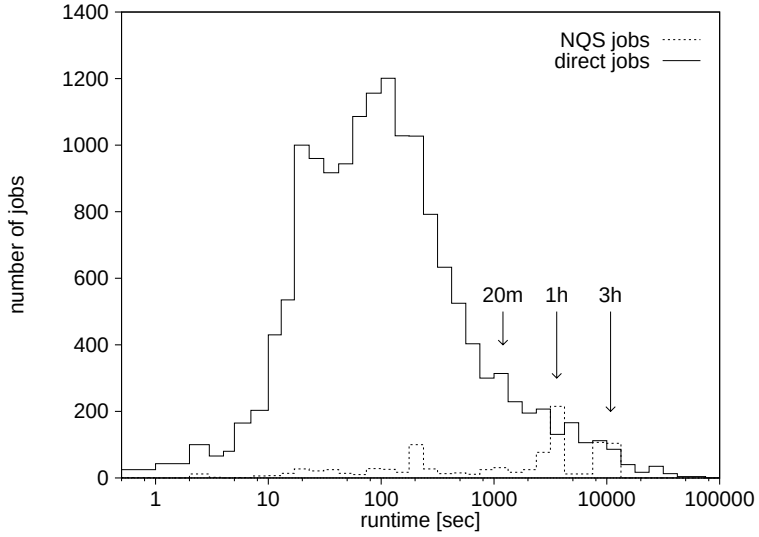

Figure 11: *Distribution of runtimes for direct and NQS jobs.*

6 Job Submission Rate

The traced system was in production use by a number of users. This included both interactive use, and the execution of relatively long jobs. Jobs were submitted directly or through NQS. In this section we study the rate at which jobs were presented to the system for execution; for NQS jobs, this is the rate at which NQS executes them, not the rate at which users submit to NQS.

Fig. 12 shows the submission rate of new jobs, as a function of time of day. Only user

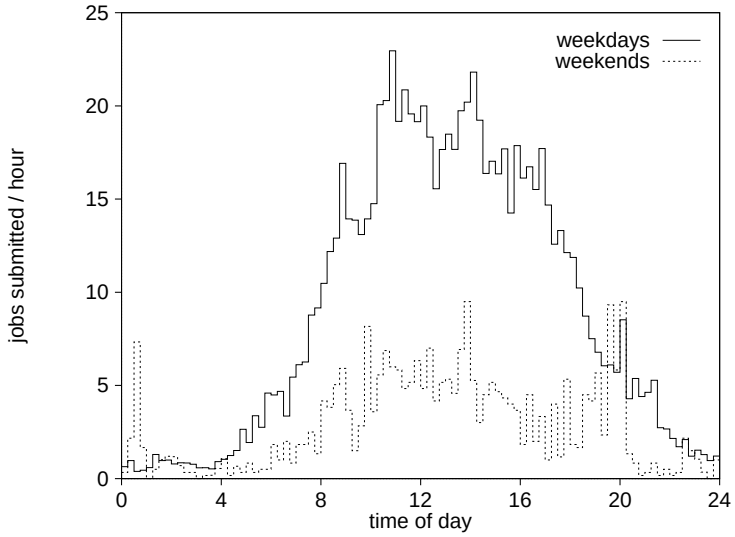

Figure 12: *Job submission rate as function of time of day.*

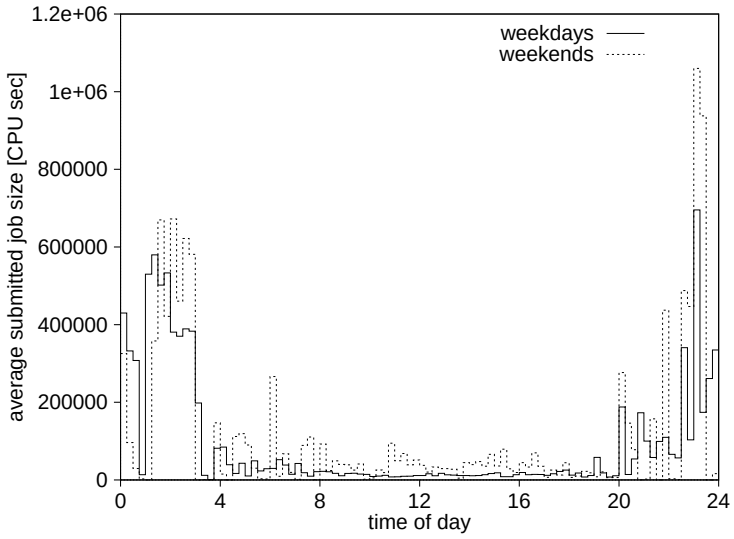

Figure 13: *Submitted job size as function of time of day.*

jobs are included (i.e. system support jobs are not). On weekdays, a daily cycle is obvious, with a high submission rate during the working day and a low rate at night. The peak is in the late morning, with a noticeable drop during lunch. This pattern is similar to known results from uniprocessor interactive systems [4]. The peak value is about one job every 2.6 minutes, on average. The implication is that the load on the system does not change too often, so scheduling schemes can afford to optimize the way resources are shared.

The gradual rise in activity before 8 AM is attributed to users from the East Coast (three time zones away), rather than to early risers in the West Coast. On weekends the submission rate during the day is much lower than on work days, but still higher than during the night. It should be noted that the graph only shows user activity. When system support activity is included, the periodic `pwd` commands cause a significant increase in the job submission rate. However, the numbers of these commands are not completely deterministic. There are many more of them between 4 AM and 4 PM (average 15.5 per hour) than between 4 PM and 4 AM (average 7.9 per hour). Their effect is therefore especially noticeable between 4 and 8 AM, when system activity is otherwise rather low.

The number of submitted jobs reflects the interactive workload, but not the whole load on the system. It is also important to consider the size of the jobs. Fig. 13 shows the average submitted job size for different hours of the day. It is seen that during the workday the average job size is very small. Starting at about 8 PM, the average submitted job size becomes significantly larger, with the largest jobs submitted at the beginning of the night. Much of this is due to the fact that the large NQS queues become active at 8 PM. Jobs submitted during the weekend tend to be larger than those submitted on work days.

Another way to look at the job submission process is via the interarrival times. The distribution of interarrival times is shown in Fig. 14. This distribution is often assumed to be exponential. The measurements cast some doubts as to the validity of this assumption,

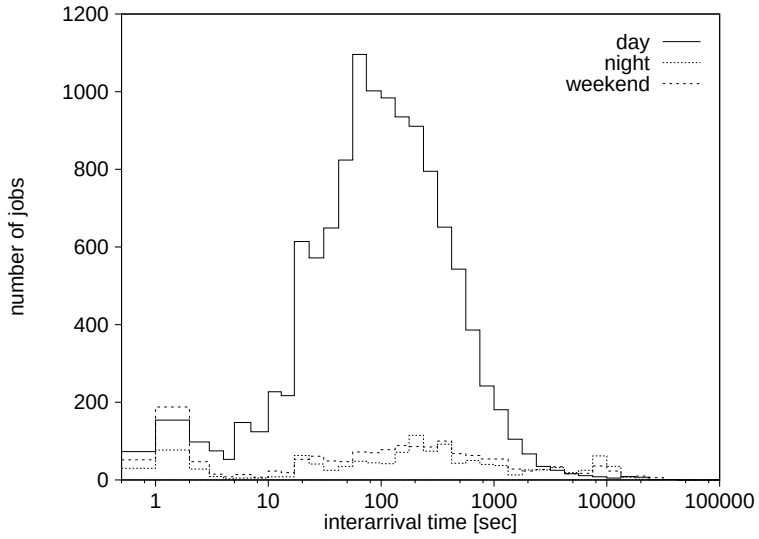

Figure 14: *Distribution of interarrival times.*

period	*mean*	*standard deviation*	*coefficient of variation*
day	269.6	958.2	3.56
night	1536.6	3240.5	2.11
weekend	1161.3	3287.3	2.83

Table IV: *Measured parameters of distribution of interarrival times.*

as the coefficient of variation is significantly higher than 1, indicating a larger spread than in an exponential distribution (see Table IV). Note that the apparent lack of weight at low values is an artifact of using a logarithmic scale, and does not indicate a departure from an exponential distribution. The mean values at night and in the weekend are much larger than during workdays, probably because there are much more long batch jobs and much less interactive work.

7 User Activity and Application Usage

The trace included activity of 50 users, in addition to system support staff. Some of the users demonstrated a very low level of activity (7 had less than 10 jobs each), while others were very active. The distribution of levels of activity is shown in Fig. 15. The most active user submitted 2607 jobs during the three months that were traced. These jobs included 336 NQS jobs, and the rest were repeated executions of only 5 different applications. The most versatile user had 78 applications, which where executed a total of 652 times. In general, many users ran the same application many times. Note that we only have such statistics for jobs that were submitted directly, as the application was not recorded for NQS jobs.

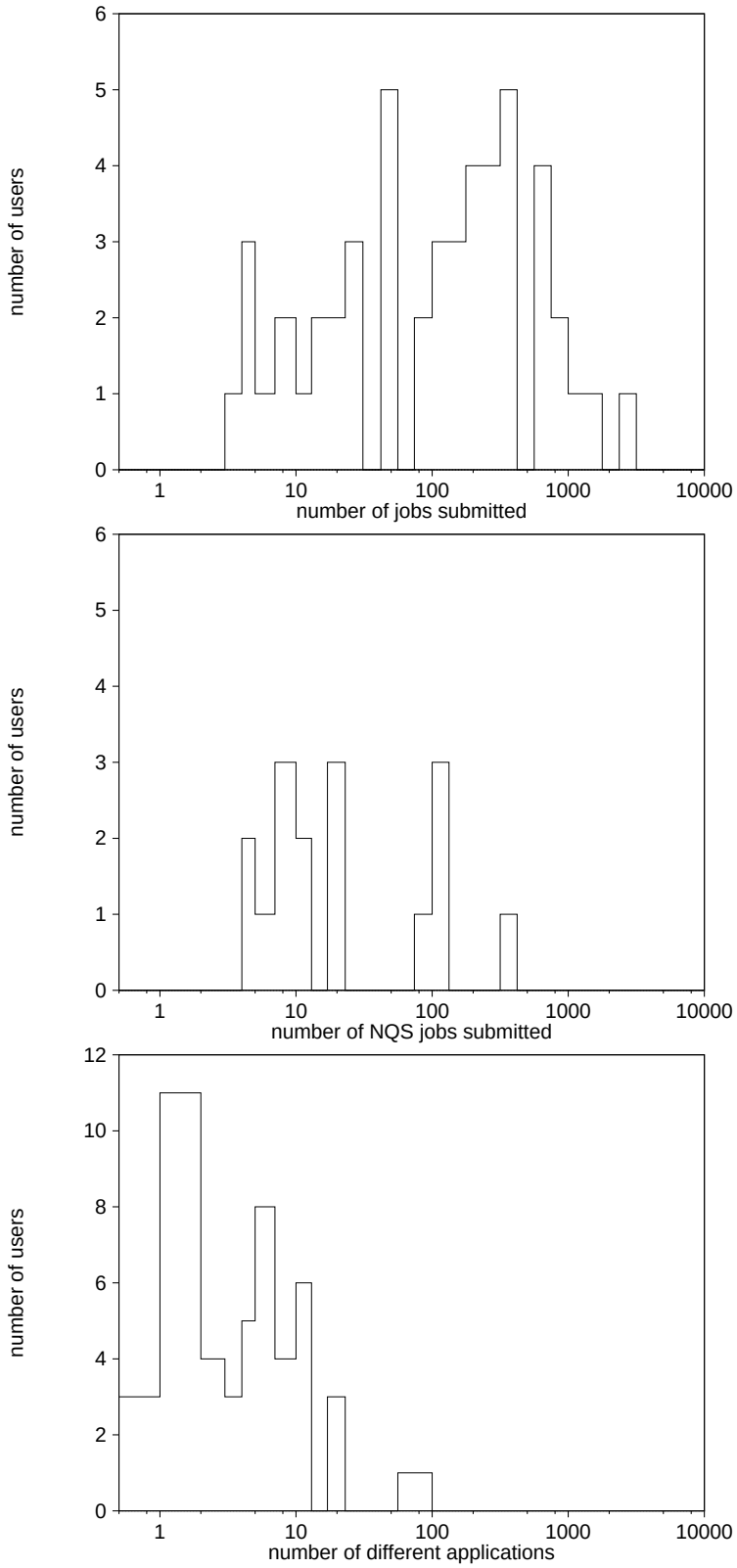

Figure 15: *Profiles of user activity.*

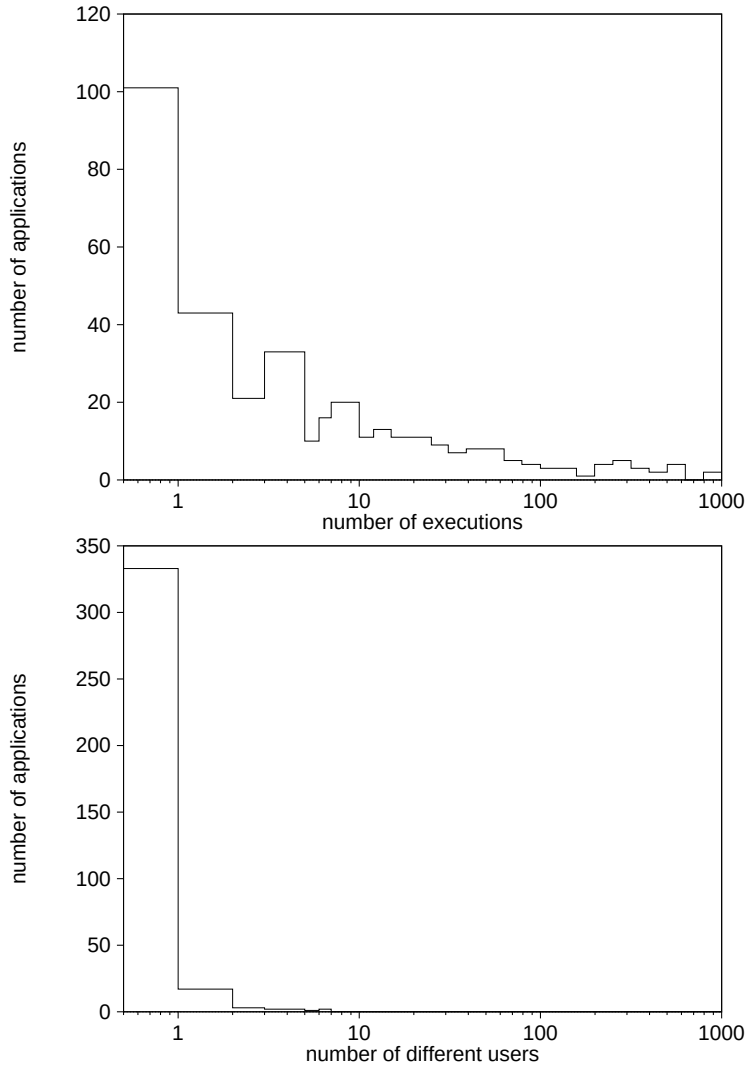

Figure 16: *Profiles of application usage.*

The statistics for applications are shown in Fig. 16. 358 different applications appeared in the trace (excluding NQS jobs, Unix commands, those that ran on the service node, and those that were only executed by system support staff). 101 of them were only executed once. The distribution of the rest is rather wide. The most popular job was executed a total of 1082 times. The distribution for sharing is much narrower. 333 jobs were only executed by a single user. The most widely shared application was executed by 7 different users. Naturally, Unix commands were shared by larger numbers of users (not plotted).

Applications are often coded in a style that allows them to be executed on partitions of different sizes. The traces show extensive use of this option (Fig. 17). Of the 257 applications that were executed more than once, 147 (57.2%) were executed on 2 or more different partition sizes. Three applications were even executed on all 8 possible partition sizes. In

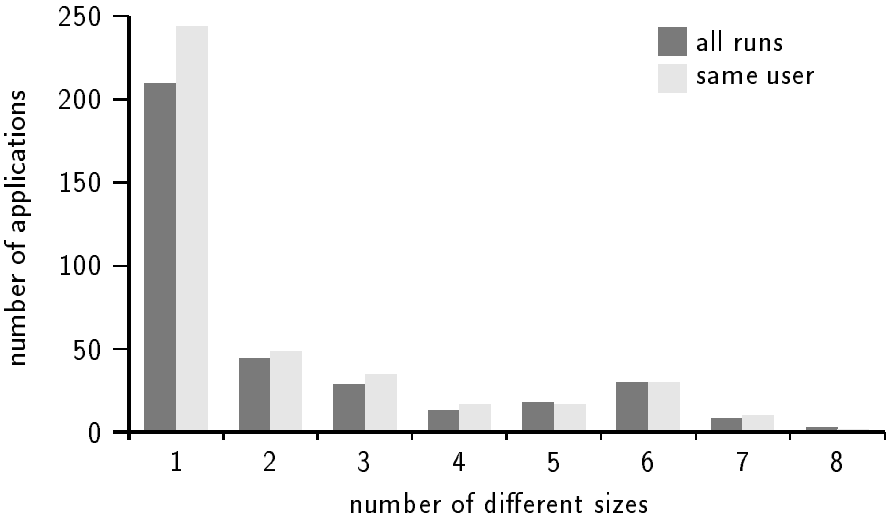

Figure 17: *Histogram of applications executed on different subcube sizes.*

most cases, this was not a result of different users running the same application on different size subcubes; rather, the same user would run on different sizes. The bars for executions by the same user appear higher than those for all runs because applications that were shared by multiple users are counted separately for each one.

The main interest in the repeated execution of the same job stems from the possibility of using statistics relating to previous runs to estimate the resource requirements of additional runs. As a simple check of this idea, the coefficient of variation of the runtimes of jobs that

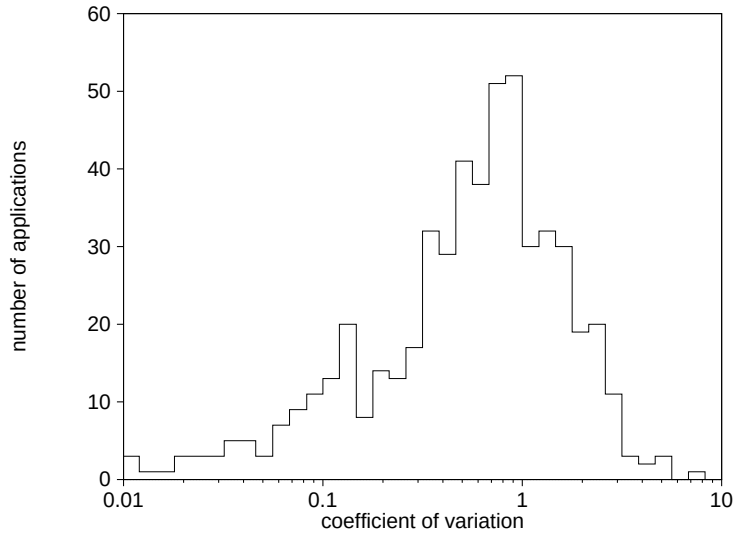

Figure 18: *Histogram of the coefficient of variation of runtimes of applications that were executed repeatedly.*

were executed more than once was tabulated. In cases where the same job was executed by multiple users or on different subcube sizes, each one was considered separately. The results are shown in Fig. 18. The majority of cases had a coefficient of variation smaller than 1, which is promising. The predictive power can be improved by using a more sophisticated model of job behavior, rather than just using the mean of previous runs [7].

8 Conclusions

As parallel machines come into production use, it becomes possible to trace their workloads. In many cases, this is done anyway as part of the accounting procedures. Analyzing such traces can provide a wealth of information about usage patterns, which is invaluable for the design of new and improved systems. The analysis is straightforward, and the traces are small relative to those used in other fields. For example, our whole trace is only 2.15 MB, in ASCII format.

Our results show that there is a roughly uniform distribution of user jobs of different sizes, with the big ones consuming most of the resources. This implies that the big jobs must be matched together for good utilization, as there are insufficient small jobs to fill the gaps. NQS goes a long way in filling this need, but it is used mainly during the night to prevent interference with interactive work. It is expected that achieving high utilization would be less of a problem in multistage or mesh-connected machines, where partitioning is more flexible than in a hypercube. Also, time-slicing may actually improve utilization, despite its increased overhead.

The distribution of runtimes of different jobs indicates that jobs with a higher degree of parallelism tend to run longer. This favors the memory-bound model of application scaling. Both job runtimes and interarrival times have a high coefficient of variation, indicating that a hyperexponential distribution may provide an appropriate model. When multiple executions of the same application on the same number of nodes are considered, the coefficient of variation tends to be less than 1. This indicates that historical information can be used to gauge the resource requirements of applications.

Acknowledgements

This study was initiated using the CHARISMA traces, collected by David Kotz and Nils Nieuwejaar of Dartmouth College to study I/O behavior of applications [14]. These traces were later replaced by an accounting trace collected at NASA Ames, which had the advantage that it included user and job information, and that it covered a much longer period. The generosity and help provided by Dave and Nils at the initial stages of this work are greatly appreciated.

References

[1] A. K. Agrawala, J. M. Mohr, and R. M. Bryant, "*An approach to the workload characterization problem*". *Computer* **9(6)**, pp. 18–32, Jun 1976.

[2] G. M. Amdahl, "*Validity of the single processor approach to achieving large scale computer capabilities*". In *AFIPS Spring Joint Comput. Conf.*, vol. 30, pp. 483–485, Apr 1967.

[3] E. Barszcz, "*Intercube communication for the iPSC/860*". In *Scalable High-Performance Comput. Conf.*, pp. 307–313, 1992.

[4] M. Calzarossa and G. Serazzi, "*A characterization of the variation in time of workload arrival patterns*". *IEEE Trans. Comput.* **C-34(2)**, pp. 156–162, Feb 1985.

[5] M. Calzarossa and G. Serazzi, "*Workload characterization: a survey*". *Proc. IEEE* **81(8)**, pp. 1136–1150, Aug 1993.

[6] S-H. Chiang, R. K. Mansharamani, and M. K. Vernon, "*Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 33–44, May 1994.

[7] M. V. Devarakonda and R. K. Iyer, "*Predictability of process resource usage: a measurement-based study on UNIX*". *IEEE Trans. Softw. Eng.* **15(12)**, pp. 1579–1586, Dec 1989.

[8] D. G. Feitelson and L. Rudolph, "*Wasted resources in gang scheduling*". In 5th *Jerusalem Conf. Information Technology*, pp. 127–136, IEEE Computer Society Press, Oct 1990.

[9] D. Ferrari, "*Workload characterization and selection in computer performance measurement*". *Computer* **5(4)**, pp. 18–24, Jul/Aug 1972.

[10] J. L. Gustafson, "*Reevaluating Amdahl's law*". *Comm. ACM* **31(5)**, pp. 532–533, May 1988. See also *Comm. ACM* **32(2)**, pp. 262–264, Feb 1989, and *Comm. ACM* **32(8)**, pp. 1014–1016, Aug 1989.

[11] J. L. Gustafson, G. R. Montry, and R. E. Benner, "*Development of parallel methods for a 1024-processor hypercube*". *SIAM J. Sci. Statist. Comput.* **9(4)**, pp. 609–638, Jul 1988.

[12] P. Heidelberger and S. S. Lavenberg, "*Computer performance evaluation methodology*". *IEEE Trans. Comput.* **C-33(12)**, pp. 1195–1220, Dec 1984.

[13] Intel Corp., *iPSC/860 Multi-User Accounting, Control, and Scheduling Utilities Manual.* Order number 312261-002, May 1992.

[14] D. Kotz and N. Nieuwejaar, "*Dynamic file-access characteristics of a production parallel scientific workload*". In *Supercomputing '94*, Nov 1994.

[15] S. Krakowiak, *Principles of Operating Systems*. MIT Press, 1988.

[16] P. Krueger and R. Chawla, "*The stealth distributed scheduler*". In 11th *Intl. Conf. Distributed Comput. Syst.*, pp. 336–343, May 1991.

[17] P. Krueger, T-H. Lai, and V. A. Radiya, "*Processor allocation vs. job scheduling on hypercube computers*". In 11th *Intl. Conf. Distributed Comput. Syst.*, pp. 394–401, May 1991.

[18] M. Kumar, "*Measuring parallelism in computation-intensive scientific/engineering applications*". *IEEE Trans. Comput.* **37(9)**, pp. 1088–1098, Sep 1988.

[19] S. T. Leutenegger and M. K. Vernon, "*The performance of multiprogrammed multiprocessor scheduling policies*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 226–236, May 1990.

[20] S. Majumdar, D. L. Eager, and R. B. Bunt, "*Characterisation of programs for scheduling in multiprogrammed parallel systems*". *Performance Evaluation* **13(2)**, pp. 109–130, 1991.

[21] S. Majumdar, D. L. Eager, and R. B. Bunt, "*Scheduling in multiprogrammed parallel systems*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 104–113, May 1988.

[22] P. Messina, "*The Concurrent Supercomputing Consortium: year 1*". *IEEE Parallel & Distributed Technology* **1(1)**, pp. 9–16, Feb 1993.

[23] V. K. Naik, S. K. Setia, and M. S. Squillante, *Performance Analysis of Job Scheduling Policies in Parallel Supercomputer Environments*. Research Report RC 19138 (82333), IBM T. J. Watson Research Center, Sep 1993. Also in *Supercomputing '93*.

[24] K. C. Sevcik, "*Characterization of parallelism in applications and their use in scheduling*". In *SIGMETRICS Conf. Measurement & Modeling of Comput. Syst.*, pp. 171–180, May 1989.

[25] J. P. Singh, J. L. Hennessy, and A. Gupta, "*Scaling parallel programs for multiprocessors: methodology and examples*". *Computer* **26(7)**, pp. 42–50, Jul 1993.

[26] X-H. Sun and L. M. Ni, "*Scalable problems and memory-bounded speedup*". *J. Parallel & Distributed Comput.* **19(1)**, pp. 27–37, Sep 1993.

[27] P. H. Worley, "*The effect of time constraints on scaled speedup*". *SIAM J. Sci. Statist. Comput.* **11(5)**, pp. 838–858, Sep 1990.