George Mason University

Experimental Comparative Study of Job Management Systems

A Report for the NSA LUCITE Task Order
Productive Use of Distributed Reconfigurable Computing

Tarek El-Ghazawi, P.I.
Kris Gaj, Co-P.I.
Nikitas Alexandridis (GWU), GWU-P.I.
Brian Schott(USC/ISI), Co-P.I.

Graduate Assistants:
Alexandru V Staicu, Jacek R. Radzikowski,
Nguyen Nguyen, Jearanai Vongsaard, and Sébastien Chauvin (GMU),
Preeyapong Samipagdi and Suboh A. Suboh (GWU)

July 20, 2001

OUTLINE:

Executive Summary

Selecting a Job Management System as the basis of a tool for the effective utilization of networked reconfigurable resources is achieved using a number of criteria and processes.

First, the system has to possess minimum functional capabilities that are considered promising for achieving our goal according to a set of well defined requirements. Those functional capabilities were investigated and compared in the conceptual study and translated into 4 candidate systems that were worth pursuing. These were LSF, PBS, Codine, and Condor (time permitting).

Secondly, experience was to be developed with the candidate systems for further analysis and experimental evaluation. The additional analysis, which was based on the gained experience and familiarity with the systems, added with an experimental evaluation study, served as the basis for our final determinations and recommendation.

The additional analysis focused on the ease of extending each of the candidate systems to recognize and deal with reconfigurable resources, as well as some of the auxiliary but desirable features, such as checkpointing.

The experimental evaluation was conducted on testbed created in the lab and composed of several Unix workstations. Worklaods were selected from the NSA HPC benchmarks, the NAS Parallel Benchmark, and the UPC Benchmark, plus selected cryptography applications. A workload generator selected the workloads from the workload lists at random and using random intervals between the arriving job requests, but according to selected probability density functions. Many performance metrics were measured and reported. These include utilization, average turnaround, throughput, and JMS processes overhead.

LSF, PBS, and Codine provide for the needed extension without source code modifications. Condor can be extended by source code modifications. The source code for Condor is available. At the time of this writing, no experimental measurements for Condor were available. However, the general performance behavior of LSF, Codine, and PBS was characterized. In general, LSF has better throughput while PBS has better average turn around. LSF and PBS have comparable performance, while Codine trails by a significant margin. LSF, however, came ahead in many other areas such as operating system support (both Unix and NT), checkpointing, fault tolerance, job migration and load balancing, documentation and customer support. Based on all of the above, we recommend continuing our effort by extending LSF to support the management and effective use of networked reconfigurable resources.

1. Methodology and Findings of the Experimental study

1.1 Methodology

1.1.1 Testbed

A testbed used in our experiments is shown in Fig. 1. The testbed consists of 9 PCs running Linux OS, and 4 workstations Ultra 5, running Solaris 8. The total number of CPUs available in the testbed is 20. The network structure of the testbed is flat, so that every machine can serve as both an execution host and a submission host. In all our experiments, pallj was used as a master host for all Job Management Systems. All 13 hosts, including the master host, were configured as execution hosts. In all our experiments, `pallj` was also employed as a submission host.

1.1.2 Benchmarks

A set of 36 benchmarks has been compiled and installed on all machines of our testbed. These programs belong to the following four classes of benchmarks: NSA HPC Benchmarks, NAS Parallel Benchmarks, UPC Benchmarks, and Cryptographic Benchmarks. The detailed description of all benchmarks is provided in Appendix 1. Each benchmark has been characterized in terms of the CPU time, wall time, and memory requirements using one of the Linux machines.

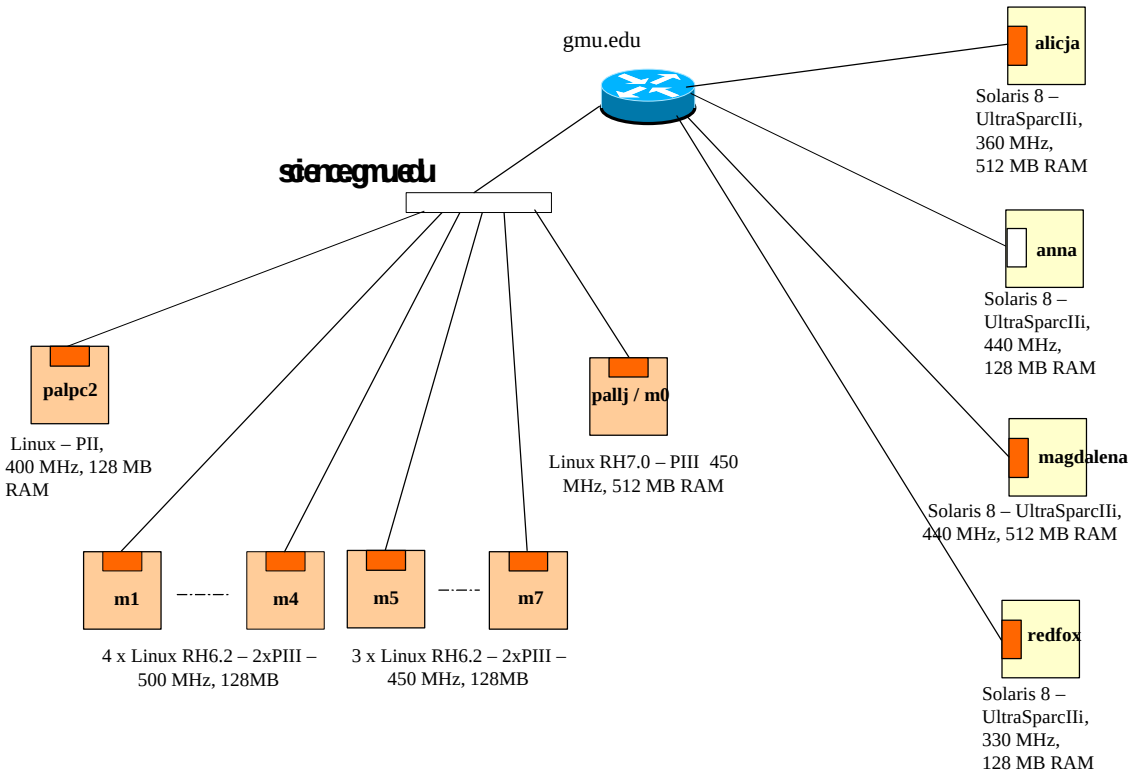

Fig. 1 Testbed used in the experimental study.

All benchmarks have been divided into the following four sets of benchmarks:

1. Set 1 – Short job list – 16 benchmarks with an execution time between 1 second and 2 minutes, and an average execution time equal to 22 seconds.
2. Set 2 – Medium job list – 8 benchmarks with an execution time between 2 minutes and 10 minutes, and an average execution time equal to 7 minutes 22 seconds.
3. Set 3 – Long job list – 6 benchmarks with an execution time between 10 minutes and 31 minutes, and an average execution time equal to 16 minutes 51 seconds.
4. Set 4 – I/O job list – 6 benchmarks with the file input and output, and an average execution time of 1 minute 36 seconds.

The detailed contents and timing and memory characteristics of all benchmark sets is given in Appendix 1.

1.1.3 Description of experiments

Each experiment consists of running N jobs chosen pseudorandomly from the given set of benchmarks, and submitted one at a time to a given JMS in the pseudorandom time intervals. All jobs were submitted from the same machine, `pallj`, and belonged to a single user of the system. The rate of the job submissions was chosen to have a Poisson distribution.

The total number of jobs submitted to a system, N, was chosen based on the expected total time of an experiment, the average execution time of jobs from the given list, and the number of machines in our testbed. In Experiments 1, 2, 4, and 5, regarding short, medium, and I/O job lists, the total number of jobs was set to 150, which led to a total experiment time of about two hours. In Experiment 3, regarding the long job list, the total number of jobs was reduced to 75 to keep the time of each experiment within the range of 2 hours.

Each experiment was repeated for three JMSes, under exactly the same initial conditions, including the same initial seeds of the pseudorandom generators. Additionally, all experiments were repeated 3 times for the same JMS to minimize the effects of random events in all machines participating in the experiment.

Additionally, each experiment was repeated for several different average time intervals between consecutive job submissions. These intervals have been chosen experimentally in such a way that they correspond to qualitatively different JMS loads. For the longest time interval, each system is very lightly loaded. Only a subset of all available CPUs is utilized at any point in time. Any new job submitted to the system can be immediately dispatched to one of the execution hosts. For the shortest time interval, a majority of CPUs are busy all the time, and almost any new job submitted to a JMS must spend some time in a queue before being dispatched to one of the execution hosts.

Experi-ment Number	Benchmark Set	Average CPU time / Job	Average Time Intervals Between Job Submissions	Total Number of Jobs	Special Assumptions
1	**Set 2, Medium job list**	**7 min 22 s**	**30 s, 15 s, 5 s**	**150**	**one job / CPU**
2	**Set 2, Medium job list**	**7 min 22 s**	**15 s**	**150**	**two jobs / CPU**
3	**Set 3, Long job list**	**16 min 51 s**	**2 min, 30 s**	**75**	**one job / CPU**
4	**Set 1, Short job list**	**22 s**	**15 s, 10 s, 5 s**	**150**	**one job / CPU**
5	**Set 4, I/O job list**	**1 min 36 s**	**15 s**	**150**	**one job / CPU**

Table 1. Characteristic features of experiments performed during our study.

The characteristic features of five experiments performed during our experimental study are summarized in Table 1. Experiments 1-4 have been completed. Experiments 5 needs to be done for one more JMS, and two more experiments are fully prepared, and are planned to be completed in the near future. Experiment 6 will investigate the behavior of each JMS in the configuration with multiple submission hosts, and Experiment 7 would attempt to quantify the fault tolerance of each JMS.

1.1.4 Settings of Job Management Systems

An attempt was made to set all JMSes to an equivalent configuration, using the following major configuration settings:

A. Number of Jobs per CPU

In all experiments, except Experiment 2, a maximum number of jobs assigned simultaneously to each CPU was set to one. In other words, no timesharing of CPUs was allowed.
This setting was chosen as an optimum because of the numerical character of benchmarks used in our study. All benchmarks from the short, medium, and long job lists have no input or output. For this kind of benchmarks, timesharing can improve only the response time, but has a negative effect on two most important performance parameters: turn-around time and throughput. This deteriorating effect of timesharing was clearly demonstrated in our Experiment 2, where two jobs were allowed to share the same CPU.

The following parameters of three investigated JMSes were used to limit the number of jobs per execution host:

LSF: `MXJ` parameter in `lsb.hosts`
PBS: `np` parameter in `nodes`
CODINE: `slots` parameter set using `qmon` or `qconf`.

B. CPU factor of execution hosts

The CPU factors determine the relative performance of execution hosts for a given type of load. Based on the recommendations given in the JMS manuals, CPU factors were set based on the relative performance of benchmarks reflecting a typical load. For each list of benchmarks, two representative benchmarks were selected, and run on machines representing all machine models. The CPU factors were set based on an average ratio of the execution time on the slowest machine to the execution time on the machine for which the CPU factor was determined.

Based on this procedure, the slowest machine had always a CPU factor equal to 1.0. The CPU factors of remaining machines varied in the range from 1.2 to 1.7 for a small job list, and from 1.4 to 1.95 for the medium job list.

The CPU factors in LSF and CODINE affect the operation of the scheduler. In PBS, the equivalent parameter has no effect on scheduling, and affects only accounting and time limit enforcements.

The following parameters of three investigated JMSes were used to set the CPU factors:
LSF: `CPUFACTOR` parameter in `lsf.shared`
PBS: `cputmult` parameter in the MOM `config` file
CODINE: `cpu` parameter in the `load_scaling` table, set using `Host Configuration` of `qmon` or `qconf`.

C. Dispatching interval

The dispatching interval determines how often the JMS scheduler attempts to dispatch pending jobs. This parameter clearly affects an average response time, as well as scheduler overhead. It may also influence the remaining performance parameters.

LSF on one side, and PBS and CODINE on the other side use clearly a different definition of this parameter. In all three systems, this parameter describes the maximum time in seconds between subsequent attempts to schedule jobs. However, in PBS and CODINE, the attempts to schedule jobs also occur whenever a new job is submitted, and whenever a running batch job terminates. The same is not the case for LSF. On the other hand, LSF has two additional parameters that can be used to limit the time spent by the job in the queue, and thus reduce the response time.

The following parameters of three investigated JMSes were used to set the job dispatching interval:
LSF: `MSB_SLEEP_TIME` in `lsb.params`
`NEW_JOB_SCHED_DELAY` in `lsb.queues`

JOB_ACCEPT_INTERVAL in lsb.queues
PBS: scheduler_iteration in the server configuration, set using qmgr
CODINE: Schedule interval in the Scheduler Configuration.

D. Job requirements

The only job requirement used in our experiments and specified during the job submission was the amount of available memory.

No information about the expected execution time, or limits on the wall or CPU time were specified.

F. Scheduling policies

No changes to the parameters describing scheduling policies were made, which means that the default First Come First Serve (FCFS) scheduling policy was used for all systems.

One should be however aware that within this policy, a different ranking of hosts fulfilling the job requirements might be used by different JMSes.
In LSF, this ranking is determined based on a normalized average 1-minute average CPU run queue length (updated every 15 s) and the paging activity of the execution hosts.
In CODINE, a job is dispatched to the least loaded host, but various parameters are provided to normalize and adjust a value of the host load depending on the capabilities and relative performance of available execution hosts.
In PBS, in the default configuration, an average load is not taken into account during scheduling.

E. Stage-in and stage-out

In Experiment 5, the stage-in and stage-out capabilities of LSF and PBS were investigated. For each job from our i/o list of benchmarks, the job submission scripts contained a list of options to transfer respective input and output files between a submission host and an execution host, immediately before, and immediately after the job execution.

Neither LSF or PBS provide a capability to use conditional stage-in or stage_out, in which a name of the transferred file would depend on the architecture of the execution host. CODINE does not support stage-in or stage-out at all, other than using the prolog and epilogue scripts associated with the run queues (rather than jobs).

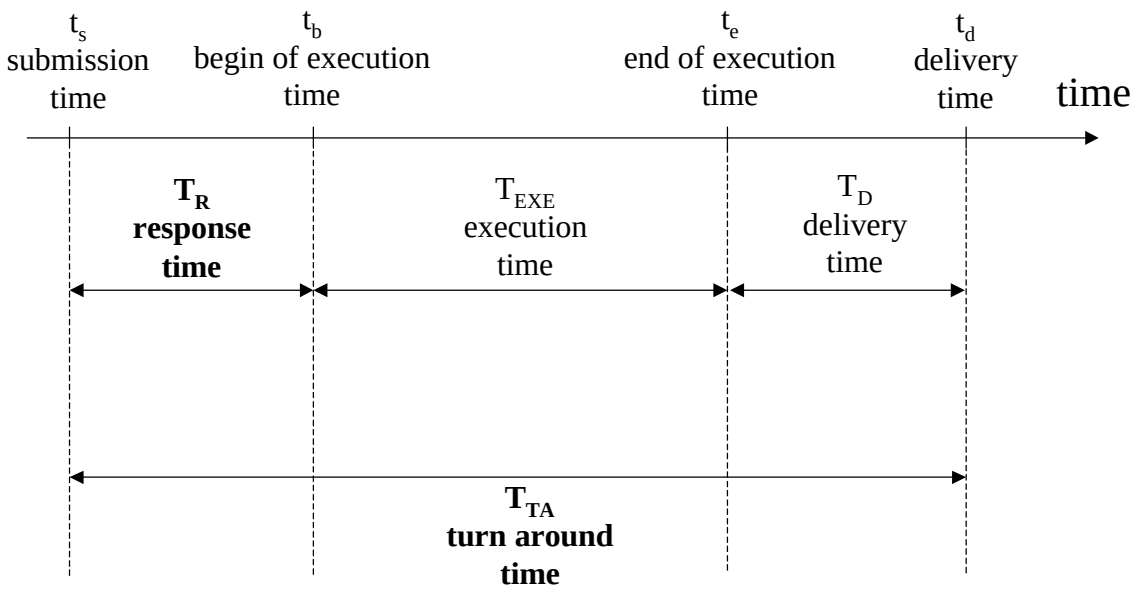

Fig. 2 Definition of timing parameters

1.1.5 Definitions of performance measures

The following performance measures were investigated in our study:

Average response time *is the average amount of time between submitting the job on one of the submission hosts and starting the job on the execution host assigned by JMS (see Fig. 2). The absolute response time is measured in time units, and the relative response time is measured as a ratio to the execution time of the job.*

Average turn around time *is the average time from submitting a job till completing it. If the job generates an output and this output is to be transferred back to the submission host, then the job is considered completed only when the entire output has been transferred to the submission host (see Fig. 2).*

Throughput *is an overall amount of work performed per time unit. In our experiments, throughput was measured using the number of jobs completed (including the transfer of the output to the submission host) in a unit of time.*

Utilization *is a ratio of a busy time span to the available time span. Time span is measured in time*resource units. In our experiment, we measured the utilization by measuring the average percentage of the CPU time used by all JMS jobs on each execution host. These average machine utilizations were then averaged over all execution hosts (see Fig. 3).*

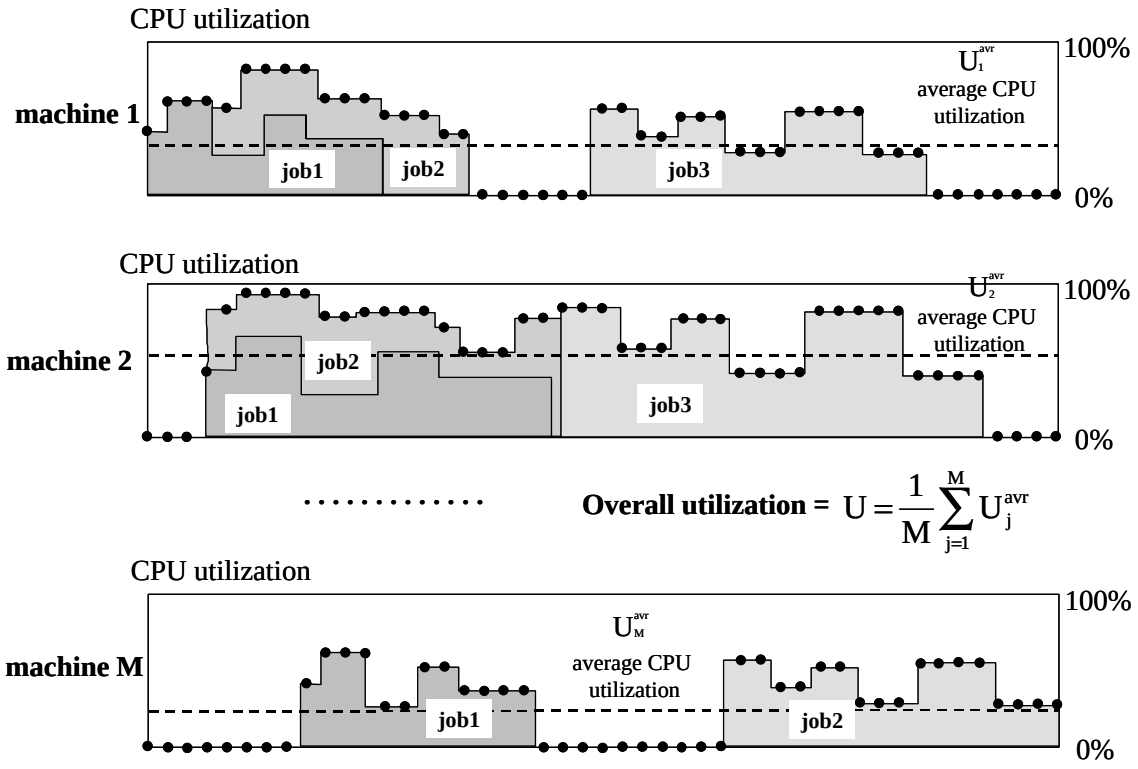

Fig. 3 Determining the system utilization using `top`

Load imbalance *is measured as the variability in the utilization of system resources. In our experiment, we measured load imbalance as a standard deviation of the average CPU utilization of all execution hosts.*

Scheduler overhead *is measured as an average percentage of the CPU time used by JMS daemons. In our experiments, the scheduler overhead was measured separately for the master host daemons and the execution host daemons.*

The exact equations describing all these performance measures are provided in Appendix 2.

1.1.6 Methodology for measuring performance parameters

Each experiment was aimed at determining values of all performance measures defined in the proceeding section and described in detail in Appendix 2. All parameters were measured in the same way for all JMSes, using utilities and mechanisms of the operating systems only.

In particular, timestamps generated using the C function gettimeofday(), were used to determine the exact time of a job submission, as well as the begin and end of the execution time. The Unix protocol NTP was used to synchronize clocks of all machines

of our testbed with the required time accuracy. The operation of NTP and the gettimeofday() function is described in Appendix 3.

In order to determine the JMS utilization, load imbalance, and JMS overhead, the Unix `top` utility was used (see Appendix 3). This utility records an approximate percentage of the CPU time used by each process running on a single machine averaged over a short period of time, e.g., 15 seconds (see Fig. 3). For each point in time, the sum of percentages corresponding to all JMS jobs is computed. These sums are then averaged over the entire duration of an experiment, to determine an average utilization of each machine by all JMS jobs The execution host utilizations averaged over all execution hosts determine the overall utilization of a JMS.

A C++ program has been written to emulate a random submission of jobs from a given host. This program takes as an input a list of jobs, a total number of submissions, an average interval between two consecutive submissions, and the name of a JMS used in a given experiment. Two postprocessing Perl scripts have been developed to process log files generated by benchmarks and the `top` utility. These scripts generate exhaustive reports including values of all performance measures separately for every execution host, and jointly for the entire testbed.

1.2 Findings

Two most important parameters determining the performance of a Job Management System from the user's point of view are turnaround time and throughput.
Turnaround time is particularly important during the day, when a user tends to work in a pseudo-interactive mode and awaits results of each subsequent experiment.
Throughput is particularly important during the night, when a user submits a large batch of jobs to a JMS and expects them to be all done by the next morning.
Additionally, *average response time* might be important in case of interactive jobs that require user input and thus a constant presence of the user.

In Experiment 1, with the medium job list, substantial differences among the relative performance of three compared systems were observed (see Apendix 4, Tables A4-1 A-C). PBS was consistently the best in terms of the turnaround time, while LSF was the best in terms of the throughput. CODINE was consistently trailing LSF and PBS according to both performance metrics.
Considering the turnaround time, PBS was better than LSF by 3 to 20 %, depending on an average interval between job submissions. At the same time, PBS outperformed CODINE by 31 to 36 %.
In terms of the throughput, LSF was consistently the best, with an advantage of 1 to 14% over PBS, and an advantage of 12 to 18% over CODINE.
The master and execution host scheduler overheads were negligible (less than 1%) for all three job management systems, except CODINE, where the master host overhead was in the range of 6 to 8%.

In Experiment 2 (see Appendix 4, Table A4-2), with the medium job list, the number of jobs per CPU was increased to two. The average turnaround time and throughput deteriorated for each JMS compared to Experiment 1 with one job per CPU. The average increase in the turn-around time was 14% for LSF, 15% for CODINE, and 35% for PBS. The throughput decreased by 2% for LSF, 12% for PBS, and 18% for CODINE.

In Experiment 3 (see Appendix 4, Tables A4-3 A-B), with the long job list, PBS outperformed LSF and CODINE in terms of both the turnaround time and throughput. Nevertheless, the performance of PBS and LSF were extremely close, and both of these systems outperformed CODINE by a large margin. The turnaround time was the best for PBS by a margin of only 1 to 6 % over LSF, and 57 to 76% over CODINE. The throughput of PBS was the best by a margin of 4 to 8 % over LSF, and 32 to 45% over CODINE. The master host scheduler overhead was the largest for CODINE, where it averaged between 5 and 8 %, compared to the negligible scheduler overhead for LSF and PBS (less than 1 %).

In Experiment 4 (see Appendix 4, Tables A4-4 A-C), with the short job list, the ranking in terms of the throughput was consistently the following: LSF first, CODINE close second, PBS third. The largest differences did not exceed 22%. In terms of the turnaround time the ranking was CODINE first, PBS close second, and LSF third. CODINE outperformed PBS by a margin of 11 to 15%, and LSF by a margin of 22 to 34%.

Thus, in majority of experiments, LSF demonstrated the best performance in terms of the throughput, and PBS in terms of the turnaround time. Except Experiment 4, with the short job list, CODINE was consistently trailing both PBS and LSF in both throughput and turnaround time.

1.3 Limitations of the study

Despite our best efforts to make our performance measurements as accurate and representative as possible, we realize and acknowledge here the limitations of our study. These limitations concern in particular:

a. the size and configuration of our testbed

In actual applications, the number of execution nodes may easily exceed 1000, and these nodes may be connected into a complex hierarchical structure.

b. number and type of operating systems

Only two relatively similar operating systems were represented in our study, Linux and Solaris.

c. network connections among system nodes

All computers in our testbed were connected using a fast local LAN. No WAN (in particular Internet) connections were used in our study.

d. type of benchmarks

Only relatively similar numerical benchmarks, with limited input/output were used in our study. No strong requirements on the amount of available memory, temporary and swap disk space, or network activity were imposed.

e. number of JMS configuration settings

Only a very limited number of possible JMS configurations was taken into account in our study. In particular, only one, default scheduling policy was considered. No checkpointing, job migration, or load balancing were used. No special requirements or limits on the program execution times were specified.

f. frequency and distribution of the workload

Only a relatively simple, Poisson-based, job submission distribution was investigated.

g. number of users and user groups

Only one user and no user groups were used in our study. All jobs were submitted with the same priority.

h. new versions of systems

Only within the duration of our experiment a substantially extended version of PBS was published. New versions of each JMS may have substantially different performance characteristics.

Each of these various factors may by itself dramatically change results and conclusions from any experimental comparative study. Investigating all these various options is by far beyond the scope of this project, and requires a joined effort of many research and industry groups.

Why the results of our study are still valid and useful?

Despite these limitations, we believe that our study has given an interesting insight into the relative performance of three leading Job Management Systems. The practical value of the study comes, among the other, from the following factors:

1. Even though our benchmarks and experiment times seem to be relatively short compared to the real-life scenarios, we make up for that by setting the average time between job submissions to the relatively small values. As a result, the systems are fully exercised, and our results are likely to scale for more realistic loads with proportionally longer job execution times and longer times between job submissions.
2. Typical users rarely use all capabilities of any complicated system, such as JMS. Instead, majority of Job Management Systems deployed in the field use the default values of majority of configuration parameters.

Additionally, our methodology and tools developed as a result of this project may be used by other groups to extend our understanding of similarities and differences among behavior and performance of existing Job Management Systems.

2. Comparative Analysis of Selected Job Management Systems Using the Critical Evaluation Factors

Experimental results

Performance:

Turn-around time:

Very good: PBS
Good: LSF
Average: CODINE

Unknown: Condor

Throughput:

Very good: LSF
Good: PBS
Average: CODINE

Unknown: Condor

Stage-in / Stage-out

Yes: LSF, PBS
Limited: Condor
No: CODINE

Stage-in and stage-out is supported by LSF and PBS and has been experimentally verified during our study. Stage-in and stage-out in Condor has several limitations, including no support for transfer of subdirectories. Stage-in and stage-out capabilities are not included in CODINE. Special scripts would need to be written to support the equivalent functionality.

Functional considerations

Ease of extension to reconfigurable hardware:

Excellent: LSF, PBS, CODINE

More difficult: Condor

LSF, CODINE, and the most recent version of PBS (5.1.1) all support defining and managing new dynamic resources using configuration files only, without any need for changes in the source code of JMS programs. This feature can be used to extend each of these three JMSes to manage FPGA-based accelerator boards. The new resource that needs to be added to a given JMS would represent the availability of the accelerator board for JMS users.
The primary component of this extension would a board monitoring program or script that controls the status of an accelerator board, and periodically communicates this status to a local resource manager running on each execution host. This board monitoring program/script should be written according to the specification for

- ELIM, External Load Information Manager in LSF
- Load sensor in CODINE, and
- Shell escape to the MOM configuration file in PBS.

This program would be started by a local resource manager (LIM in LSF, cod_execd in CODINE, and MOM in PBS), and would communicate with this manager using standard output. The local resource manager would transfer the information periodically or by request to a JMS scheduler, who would use this information to match each job that requires acceleration with an appropriate host. The job requirements regarding this new reconfigurable resource would be specified during a job submission and enforced by a scheduler the same way as requirements regarding default built-in resources.
Extending Condor to provide the similar functionality would require changes in the source code of this JMS.

Operating Systems:

UNIX and NT: LSF
Unix & limited NT: Condor
UNIX only: CODINE, PBS

LSF is the only JMS that fully supports Windows NT. In Condor, there is a limitation that jobs submitted from Windows NT can only be executed on machines running Windows NT.
CODINE and PBS currently do not support Windows NT.

Parallel job support

Full: LSF, CODINE
Limited: PBS, Condor

A full parallel job support is offered only by LSF and CODINE. Both these systems support programming, testing, execution, and run-time management of parallel applications. Both JMSes provide full fine-grained resource control over parallel applications provided that these applications are compiled with the vendor-supplied message passing libraries (e.g. MPI or PVM). This resource control includes dynamic resource discovery and allocation, transparent invocation of parallel components, monitoring, and accounting.
PBS has a limited parallel job support. PBS cannot control or monitor resource usage of all processes belonging to a parallel job. PBS can only make a list of allocated nodes available to the parallel job and hope that the vendor and the user make use of this list and stay within the allocated nodes.
Condor can act as a resource manager for the PVM (Parallel Virtual Machine) daemon. Whenever a PVM program asks for nodes, the request is forwarded to Condor. Condor finds a machine in the Condor pool and adds it to the virtual machine. The Condor-PVM environment is most suitable for parallel jobs that follow the master-worker paradigm. The current version of Condor does not support MPI.

Checkpointing:

Strong: LSF, Condor
Limited: CODINE
Weak: PBS

LSF supports kernel-level checkpointing on all operating systems that provide it. LSF also supplies a user-level library to provide checkpointing on operating systems that do not provide kernel-level control. This checkpointing can have two forms: the user-level checkpointing, in which application object files must be relinked with a set of libraries provided by LSF, and the application level checkpointing, which implies changes in the application source files. The user-level checkpointing has multiple limitations including the architecture-specific checkpoint files and no support for parallel jobs.
Checkpointing in Condor can be done at the user and the application levels, but it has multiple limitations, including no support for parallel jobs or interprocess communication. Checkpointing in CODINE can be done at the same three levels (kernel, application, and user) as checkpointing in LSF, however the application and user-level checkpointing must be based on foreign checkpointing libraries, such as Condor libraries.
In PBS, only the kernel-level checkpointing is supported on a small number of operating systems that provide this mechanism.

Job migration and Dynamic load balancing:

Yes: LSF, CODINE, Condor
No: PBS

In LSF, CODINE, and Condor jobs can be checkpointed and migrated to another machine with a lighter load. If no checkpointing is supported by a given job (e.g.,

commercial software), and the job is preempted from a heavily loaded host, this job can be rerun from the beginning on another machine. In LSF and CODINE a job can be automatically migrated if it is suspended for an extended period of time to assure load balancing. In CODINE the migration may also be caused by exceeding the load value that forces migration.
PBS does not support either job migration or load balancing.

Fault tolerance (other than checkpointing):

Good: LSF, CODINE
Average: PBS, Condor

In LSF and CODINE, if the master host becomes unavailable or its daemons fail, another host takes over automatically. In LSF, the number of transitions is limited only by the total number of hosts in the system; in CODINE, only one additional host is specified as a shadow master host. In all JMSes the jobs can be automatically rerun if the execution host fails.

Scalability:

Excellent: LSF
Good: CODINE, PBS, Condor

LSF Multicluster allows dividing computational resources of a very large or geographically dispersed organization into a number of autonomous clusters. This configuration enables load sharing and batch processing among clusters, and can support systems with multiple thousands of nodes. The similar capability is not available in any other JMS.

Documentation:

Excellent: LSF
Good: PBS, CODINE, Condor

The documentation of LSF is superior compared to the documentation of other JMS systems from the point of view of organization, completeness, readability, and clear division of information into parts required by LSF administrators, users, and programmers. Despite a larger volume of this documentation, searching for the required information is easier and faster than in other systems.

Technical support:

Excellent: LSF
Good: PBS
Average: Condor

Unknown: CODINE

Technical support of Platform Computing Corporation (LSF), Veridian Systems Inc. (PBS) and the Condor Team at the University of Wisconsin-Madison were contacted several times during our experimental study. The typical issues concerned problems with installation and configuration of the systems, and clarifications regarding the capabilities of each JMS. The technical support of Sun Grid Engine was not used during our study.
The technical support of LSF was found to be excellent. A typical response time was in the range of 2-3 hours. The answers to our questions were complete and accurate. Additionally, the LSF team volunteered to give a review of the capabilities of the system at the beginning of the experimental phase of our project.
The technical support of PBS was good. A typical response time was in the range of 24 hours. Nevertheless, the follow-up communication was sometimes necessary to clarify answers to our questions.
The technical support of CODINE was average. A typical response time was in the range of 2-3 days, and selected answers required a follow-up communication.

3. Summary of Findings

LSF

Cons	Pros
o **Second best performance in terms of turnaround time**	o **Ease of extension to support reconfigurable hardware** o **Best throughput** o **Excellent documentation** o **Excellent technical support** o **Strong checkpointing** o **Job migration and dynamic load balancing** o **Strong support for parallel jobs** o **Very good fault tolerance** o **Excellent scalability (multiclustering)**

CODINE / Sun Grid Engine

Cons	Pros
o **Average performance** o **No support for Windows NT** o **No stage-in / stage-out** o **Externally supported checkpointing only**	o **Free binary version (not counting technical support)** o **Ease of extension to support reconfigurable hardware** o **Parallel job support** o **Job migration and dynamic load balancing** o **Very good fault tolerance**

PBS

Cons	Pros
o **No support for Windows NT** o **No checkpointing** o **No dynamic load balancing** o **Limited parallel job support**	o **Ease of extension to support reconfigurable hardware (but only in PBS Pro 5.1.1)** o **Best turn-around time**

Condor

Cons	Pros
o **Difficulty of extension to support reconfigurable hardware** o **No interactive job support** o **Limited parallel job support** o **Average technical support**	o **Free binary and source version** o **Strong checkpointing**

The comparative analysis of four investigated job management systems is summarized in the table above.

LSF was classified in the top group according to every investigated feature, except the average turn-around time. This performance metrics could be possibly improved by non-default JMS parameter settings, targeting the given load characteristics and the testbed used during our experiments. Even despite these suboptimal settings, LSF consistently demonstrated the highest throughput. In all other respects, LSF performed extremely well, outperforming all other Job Management Systems in terms of the operating system support, scalability, documentation, and technical support. It was also one of only two systems that fully support parallel jobs, checkpointing, and offer strong resistance against the master host failure.

CODINE performed extremely well in multiple categories such as parallel job support, job migration, load balancing, and resistance against the master host failure. It is also the only investigated system, apart from Condor, which is available for free. The major drawbacks of CODINE include a relatively long turn-around time, low throughput, lack of support for Windows NT, no support for stage-in and stage-out, and only externally supported checkpointing.

PBS demonstrated an excellent performance in terms of the turn-around time, and the second best in terms of the throughput. The primary weaknesses of this system include no support for Windows NT, very limited checkpointing, no job migration or load balancing, and limited parallel job support.

Condor distinguished itself from other systems in terms of the strong checkpointing. It is also one of the oldest and the best understood job management systems, and it is available for free, including its sources. The main weaknesses of Condor include no support for interactive jobs, the limited support for parallel jobs, and average technical support. Because of the timing constraints, and difficulties with installing Condor in our environment, we were not able to experimentally evaluate the performance of Condor compared to other job management systems. We are planning to complete experiments involving Condor in the near future.

Surprisingly, an ease of extension to support FPGA-based accelerator boards appeared to be a minor factor in comparison. Three out of four systems, LSF, CODINE, and PBSPro v. 5.1, seems to be easily extendable without the need for any changes in their source code. The fourth system, Condor, can also be relatively easily extended, taken into account the full access to its source files.

4. Recommendation

Taking into account the combined results of our conceptual and experimental study we recommend choosing **LSF** as a primary system to be extended to support FPGA-based accelerator boards.

At the same time, it should be understood that the primary task that needs to accomplished during the planned extension, the development of an FPGA-board monitor, is independent of the choice of the job management system. The primary function of this monitor is to control the status of the board and resolve any conflicts regarding the access to the board by JMS jobs and local users. After this board monitor is developed, it can be relatively easily adjusted to work with the three investigated job management systems, LSF, CODINE, and PBSPro v. 5.1.

The specific features of job management systems that support extension to reconfigurable hardware include

- capability to define new dynamic resources,
- strong support for stage-in and stage-out in order to allow an easy transfer of the FPGA configuration bitstreams, data inputs, and results between the submission host and the execution host with reconfigurable hardware;
- strong support for checkpointing, job migration, and load balancing to minimize the time penalty caused by interrupting FPGA-accelerated jobs by local users.

Among the three earlier identified systems, these two features are best supported by LSF. CODINE is the second best with limited support for stage-in and stage-out and only external checkpointing libraries, and PBSPro is the least suitable because of the limited support for checkpointing and no support for job migration and load balancing.

Appendix 1

Descriptions of Benchmarks

A. NAS Parallel Benchmarks (NPB)

3-D Fast Fourier Transform Partial Differential Equation (FT) Benchmark:

In this benchmark a 3-D partial differential equation is numerically solved using forward and inverse FFTs. This kernel performs the essence of many "spectral" methods.

The implementation of the 3-D FFT PDE benchmark follows a fairly standard scheme. The 3-D DFT is computed using a 3-D fast Fourier transform (FFT) routine. The result is multiplied by certain exponentials and is then performed an inverse 3-D DFT. Note that the 3-D FFT steps require considerable communication for operations such as array transpositions.

Class	Problem Size	Program Name	MFLOPS	CPU Usage	Wall Time [s]	Memory Usage
S	64x64x64 6 iterations	`ft.S`	40	80%	4.4	2.5%
W	128x128x64 6 iterations	`ft.W`	38.80	98%	9.6	5%
A	256x256x128 6 iterations	`ft.A`	37.37	97%	191	76%

Table A1-1: Performance characteristics of the 3-D FFT PDF benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

MultiGrid (MG) Benchmark:

The MG benchmark is a simplified multigrid kernel. It is used to obtain an approximate solution to a scalar Poisson problem on a discrete 3D grid with periodic boundary conditions. This problem is simplified in the sense that it has constant rather than variable coefficients as in a more realistic application.

It implements a V-cycle multigrid algorithm in order to arrive at an approximate solution to a Poisson problem using a 256x256x256 grid with periodic boundary conditions. Four iterations are used to obtain an approximate solution. The algorithm works continuously on a set of grids that are made between coarse and fine. The residual is first restricted from the fine grid to the coarse with the projection operator. An approximate solution is then calculated on the coarsest grid, followed by the prolongation of the solution from the coarse grid to the fine. At each step of the prolongation, the residual is calculated and a smoother is applied.

Class	Problem Size	Program Name	MFLOPS	CPU Usage	Wall Time [s]	Memory Usage
S	32x32x32 4 iterations	`mg.S`	50.83		0.15	
W	64x64x64 40 iterations	`mg.W`	47.87	96%	11	1.5%
A	256x256x256 4 iterations	`mg.A`	36.31	95%	106	86%

Table A1-2: Performance characteristics of the MG benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

The Embarrassingly Parallel (EP) Benchmark:

The EP Benchmark is a typical problem of many Monte Carlo simulation applications. It is intended to provide an estimate of the upper achievable limits for floating performance regardless of interprocessor communication. To reach this goal pairs of Gaussian random deviates are generated and tabulated in successive square annuli. The data dependency is minimal, hence the problem is called Embarrassingly Parallel. The result is a table of 10 integer values, which are counts of how many pseudorandom pairs satisfy some specified criteria.

The compute intensive part of the program is a loop over segments of the 2n numbers. For each iteration, a batch of pseudorandom numbers is first generated and then analyzed. The numbers are generated in batches instead of one-by-one because this is normally faster.

Class	Problem Size	Program Name	MOPS rand #/ s	CPU Usage	Wall Time [s]	Memory Usage
S	Pow(2,25) = 33554432	`ep.S`	1.20	95%	27.89	0.2%
W	Pow(2,26)= 67108864	`ep.W`	1.20	95%	55.83	0.2%
A	Pow(2,29)= 536870912	`ep.A`	1.20	98%	447.21 = 7min45s	0.2%
B	Pow(2,30)	`ep.B`	1.20	50% (1 processor)	1785.13= 30min15s	1%
C	Pow(2,32)	`ep.C`	1.20	50%(1 processor)	7131.51= 2h	1%

Table A1-3: Performance characteristics of the EP benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

Large Integer Sort (IS) Benchmark:

The IS kernel benchmark is an implementation of the bucket sort algorithm. It ranks a disordered sequence of integers generated by a sequential pseudorandom key generation algorithm.

Class	Problem Size	Program Name	MOPS Keys Sorted	CPU Usage	Wall Time [s]	Memory Usage
S	65536 10 iterations	`is.S`	inf	98%	0.25	< 1 %
W	1048576 10 iterations	`is.W`	10.49	98%	1	1.3%
A	8388608 10 iterations	`is.A`	6.45	98%	13	50%

Table A1-4: Performance characteristics of the IS benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

LU Benchmark:

LU is a simulated CFD application which uses symmetric successive over-relaxation (SSOR) to solve a block lower triangular-block upper triangular system of equations resulting from an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions.

Class	Problem Size	Program Name	MFLOPS	CPU Usage	Wall Time [s]	Memory Usage
S	12x12x12 50 iterations	`lu.S`	89.07	6.9%	1.1	0.6%
W	33x33x33 300 iterations	`lu.W`	36.94	50%	489	5.3%
A	64x64x64 250 iterations	`lu.A`	56.2	50%	2123	35.9%

Table A1-5: Performance characteristics of the LU benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

SP (pentadiagonal solver) and BT (block tridiagonal solver) Benchmarks:

SP and BT are simulated CFD applications that solve systems of equations resulting from an approximately factored implicit finite-difference discretization of the Navier-Stokes equations. Each solves three sets of uncoupled systems of equations: first in the x, then in the y, multi-partition approach is used is used to solve these systems in both the codes, since it provides good load balance and uses coarse-grained communication.

Class	Problem Size	Program Name	MFLOPS	CPU Usage	Wall Time [s]	Memory Usage
S	12x12x12 100 iterations	`sp.S`	68.6	7.9%	1.4	0.8%
W	36x36x36 400 iterations	`sp.W`	38.61	50%	367 = 6 min	11.8%
A	64x64x64 400 iterations	`sp.A`	39.40	50%	2158 = 36min	63.2%

Table A1-6: Performance characteristics of the SP benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

Class	Problem Size	Program Name	MFLOPS	CPU Usage	Wall Time [s]	Memory Usage
S	12x12x12 60 iterations	`bt.S`	76.58	50%	3.0	2%
W	24x24x24 200 iterations	`bt.W`	67.25	50%	115	13.3%

Table A1-7: Performance characteristics of the BT benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

B. NSA HPC Benchmarks

NQueens:

In the N Queens problem we seek to find all solutions to the problem of placing N queens on an NxN chessboard such that no queen can kill the other. A solution is found when no two queens are placed on the same row, column or diagonal. Each queen will be placed on one row and, as they are as many rows as queens, the problem can be reduced to finding, for each row, the column on which a queen can be placed. The problem is to compute the number of possible solutions for a given N. The algorithm used is a depth-first search through the tree of possible queen placement. Each time a solution is

encountered, a counter is incremented and the algorithm goes back looking for the next solution. A few optimizations have been applied to this basic algorithm. The program does not take any input (beside the parameters) and outputs only the number of solution (the solutions are not saved in the process).

C. UPC Benchmarks

Sobel Edge Detection Benchmark:

Edge detection has many applications in computer vision, including image registration and image compression. One popular way of performing edge detection is using the Sobel operators. The process involves the use of two masks for detecting horizontal and vertical edges. Each mask is placed on top of the image with its center aligned with the currently considered pixel. Each of the underlying image pixels is multiplied by the corresponding mask value and the results are summed. The sums for each mask are squared and summed. The square root of this previous result is stored as an edge value for the local pixel. The input data is taken from a file or generated by a random number generator, and the output data is discarded or written to a file.

Class	Problem Size	Program Name	MOPS	CPU Usage	Wall Time [s]	Memory Usage
1	256x256 pixels	`sobel.256`	Aprox. 10 MUL and 10 ADD for every pixel	100%	4	0.3
2	512x512 pixels	`sobel.512`		100%	17	0.6
3	1024x1024 pixels	`Sobel.1024`		100%	68	1.8

Table A1-8: Performance characteristics of the Sobel Edge benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

Matrix Multiplication:

This implementation of the matrix multiplication multiplies matrices of double precision floating point values. The input matrices are randomly generated or read from a file and the result is discarded or written to a file.

Class	Problem size NxPxM	Program Name	MOPS	CPU Usage	Wall Time [s]	Memory Usage
1	512x512x512	`matrix.1`	We are interested in multiplications (NxMxP)	100%	10.54	4.6%
2	1024x512x512	`matrix.2`		100%	21.05	7.7%
3	2048x512x512	`matrix.3`		100%	39.58	14.4%

Table A1-9: Performance characteristics of the Matrix Multiplication benchmarks. All timing measurements performed using PC with 2 x Pentium III, 450 MHz, 128 MB RAM, Linux OS.

D. Cryptographic Benchmarks

Exhaustive Key Search for Mars, RC6, Rijndael, Serpent, and Twofish

The most efficient attack against secret-key ciphers, such as DES (Data Encryption Standard), is an exhaustive key search attack. The assumption in this attack is that an opponent knows a small fragment of a message corresponding to the given ciphertext. By decrypting the ciphertext with all possible keys, and comparing outputs with the known block of the message, he is able to eliminate all incorrect keys and find a small subset of keys that includes the correct one, which was actually used for encryption.
In our benchmarks, five different ciphers, Mars, RC6, Rijndael, Serpent, and Twofish, are used as an object of an exhaustive key search attack. All these ciphers use a different set of internal operations and therefore have different performance characteristics. All five ciphers used in our study were the final candidates in the contest for the new Advanced Encryption Standard. The contest was organized by NIST in 1997-2001 and resulted in the choice of one of these ciphers, Rijndael, as a new U.S. Federal Information Processing Standard (FIPS). All these ciphers are likely to be deployed in real-life applications.
The source codes of all cipher implementations are available in public domain at
http://fp.gladman.plus.com/cryptography_technology/aes/index.htm
The inputs to each benchmark are: the message block, the ciphertext block, the beginning of the key range, and the key range size. The output is the number and the list of matching keys. The time of the benchmark execution can be set to an arbitrary value, since it is directly proportional to the key range size, and almost independent of other parameters.

Benchmark sets

Described above benchmarks were divided into three sets depending on their execution time on a selected Linux PC. The machine `m1` with two Pentium III processors, 450 MHz clock speed, 128 MB RAM, and Linux OS RH6.2 was used in all measurements. The members of each of the benchmark sets are listed in Tables A1-10, 11, 12, together with their respective CPU time, memory usage, and the memory requirements specified during submission of a given job to a JMS. The memory requirement was chosen to be at least 20% greater than the actual memory usage on `m1`. This margin of safety was typically sufficient to assure that the requested amount of memory was sufficient to run the program on all execution hosts available in our testbed.

Selected benchmarks from the short and medium job lists were modified in such a way that the program input is read from the input files, instead of being generated internally by the program, and a full output is written to the output files. These programs constitute the fourth set of benchmarks, we call the I/O job list. This list is specified in Table A1-13, together with the CPU time, memory requirements, and names of respective input and output files. Please, notice that because of the difference in a format of numerical binary files under Linux and Solaris 8, two equivalent input files were created. Both of these files need to be transferred to the execution host during the stage-in procedure because the user writing a submission script for a given job does not know in advance which execution host will be used to run the job.

SHORT JOBS (1 s ≤ execution time ≤ 2 minutes)

No.	Group	Name	Class	Script name	CPU time [s]	Memory Usage [MB]	Memory Requirements [MB]
1	NPB	FT	S	`ft.S.sh`	3.5	3.2	**4**
2	NPB	FT	W	`ft.W.sh`	9.4	6.4	**8**
3	NPB	MG	W	`mg.W.sh`	10.8	1.9	**3**
4	NPB	EP	S	`ep.S.sh`	26.5	0.25	**1**
5	NPB	EP	W	`ep.W.sh`	53.0	0.25	**1**
6	NPB	IS	W	`is.W.sh`	1.0	1.7	**3**
7	NPB	BT	S	`bt.S.sh`	3.0	2.5	**3**
8*	**NPB**	**BT**	**W**	**`bt.W.sh`**	**115**	**17**	**21**
9	NSA	IS	7 mln	`radix.7M.sh`	6	12.8	**16**
10	UPC	Sobel	256	`sobel.256.sh`	4	0.4	**1**
11	UPC	Sobel	512	`sobel.512.sh`	17	0.8	**1**
12*	**UPC**	**Sobel**	**1024**	**`sobel.1024.sh`**	**68**	**2.4**	**3**
13	UPC	MM	512	`matrix.1.sh`	10.5	5.9	**8**
14	UPC	MM	1024	`matrix.2.sh`	21	9.9	**12**
15	UPC	MM	2048	`matrix.3.sh`	40	18.4	**23**
Average					**22.0**		

*** benchmarks used to determine the relative CPU factors of execution hosts**

Table A1-10 Benchmark Set No. 1: Short Job List.

MEDIUM JOBS (2 minutes < Execution Time ≤ 10 minutes)

No.	Group	Name	Class	Script name	CPU time [min:s]	Memory Usage [MB]	Memory Requirements [MB]
1	NPB	EP	A	`ep.A.sh`	7:45	1.3	**3**
2	NPB	LU	W	`lu.W.sh`	8:09	6.8	**9**
3*	**NPB**	**SP**	**W**	**`sp.W.sh`**	**6:07**	**15.1**	**19**
4	Crypto	Mars	M	`crypto.mars.M.sh`	9:21	0.4	**1**
5	Crypto	RC6	M	`crypto.rc6.M.sh`	6:21	0.4	**1**
6	Crypto	Rijndael	M	`crypto.rijndael.M.sh`	4:11	0.4	**1**
7	Crypto	Serpent	M	`crypto.serpent.M.sh`	8:54	0.4	**1**
8*	**Crypto**	**Twofish**	**M**	**`crypto.twofish.M.sh`**	**8:05**	**0.4**	**1**
Average					**7:22**		

*** benchmarks used to determine the relative CPU factors of execution hosts**

Table A1-11 Benchmark Set No. 2: Medium Job List.

LONG JOBS (10 minutes < Execution Time ≤ 30 minutes)

No.	Group	Name	Class	Script name	CPU time [min:s]	Memory Usage [MB]	Memory Requirements [MB]
1*	**NPB**	**EP**	**B**	**`ep.B.sh`**	**30:15**	**5**	**6**
2	Crypto	Mars	L	`crypto.mars.L.sh`	14:55	0.4	**1**
3	Crypto	RC6	L	`crypto.rc6.L.sh`	10:07	0.4	**1**
4	Crypto	Rijndael	L	`crypto.rijndael.L.sh`	10:58	0.4	**1**
5	Crypto	Serpent	L	`crypto.serpent.L.sh`	14:09	0.4	**1**
6*	**Crypto**	**Twofish**	**L**	**`cryto.twofish.L.sh`**	**20:45**	**0.4**	**1**
Average					**16:51**		

*** benchmarks used to determine the relative CPU factors of execution hosts**

Table A1-12 Benchmark Set No. 3: Long Job List.

INPUT/OUTPUT JOBS

No.	Group	Name	Class	Script name	CPU time [min:s]	Memory Usage [MB]	Memory Requirements [MB]	Input files	Output file
1	NPB	FT	S	`ft.S.io.sh`	0:04	3.2	**4**	`fft_64.in_pc` `fft_64.in_sun`	`fft_64.out_pc` `fft_64.out_sun`
2	NPB	FT	W	`ft.W.io.sh`	0:10	6.4	**8**	`fft_128.in_pc` `fft_128.in_sun`	`fft_128.out_pc` `fft_128.out_sun`
3	UPC	MM	512	`matrix.1.io.sh`	0:11	5.9	**8**	`mat_512.in_pc` `mat_512.in_sun`	`mat_512.out_pc` `mat_512.out_sun`
4	UPC	MM	1024	`matrix.2.io.sh`	0:21	9.9	**12**	`mat_1024.in_pc` `mat_1024.in_sun`	`mat_1024.out_pc` `mat_1024.out_sun`
5	UPC	MM	2048	`matrix.3.io.sh`	0:40	18.4	**23**	`mat_2048.in_pc` `mat_2048.in_sun`	`mat_2048.out_pc` `mat_2048.out_sun`
6	NPB	LU	W	`lu.W.io.sh`	8:09	6.8	**9**	-	LU_W.out
Average					**1:36**				

Table A1-13 Benchmark Set No. 4: Input/Output Job List.

Appendix 2

Definitions of Selected Performance Measures

Notation:

N - number of jobs

M - number of execution hosts

t_s^i – *submission time of job* i, i=*1..N*

t_b^i – begin of execution time for job *i*, *i*=1..N
time when the job begins execution on one of the execution hosts

t_b^i – end of execution time for job ***i*****,** *i*=1..N
time when the job ends execution on one of the execution hosts

t_d^i – delivery time of job ***i*****,** *i*=1..N
if output is to be returned to the submission host, the time when the entire output is transferred to this host, otherwise, time when the job ends execution on the execution host

T_R^i – response time of job *i*, *i*=1..N

T_{Rrel}^i – relative response time of job *i*, *i*=1..N

T_{EXE}^i – execution time of job *i*, *i*=1..N

T_{TA}^i – turn around time of job *i*, *i*=1..N

T_{TArel}^i – relative turn around time of job *i*, *i*=1..N

T_D^i – delivery period for job *i*, *i*=1..N

T_R^{avr} – average response time

T_{EXE}^{avr} – average execution time

T_{TA}^{avr} – average turn around time

$T_{EXE}^{1..N}$ - time of execution of N jobs

L – number of samples of the CPU utilization during the execution time of N jobs

U_j^k - CPU utilization of machine *j* during the period *k*, *j*=1..M, *k*=1..L

U_j^{avr} – average CPU utilization of machine *j*, *j*=1..M

U – utilization of all machines

LI – load imbalance of all machines

O_{EXE} – scheduling overhead of all execution hosts

O_M – scheduling overhead of the master node

O_j^k - scheduling overhead for machine *j* during the period *k*, *k*=1..L

O_j^{avr} – average scheduling overhead for the machine *j*

Definitions

Execution time

Execution time = End of execution time - Begin of execution time

Execution time of job *i*

$$T_{EXE}^i = t_e^i - t_b^i$$

Average response time *is the average amount of time between submitting the job on one of the submission hosts and starting the job on the execution host assigned by JMS. Absolute response time is measured in time units, and the relative response time is measured as a ratio to the execution time of the job.*

Response time

Response time = Begin of execution time – Submission time

Response time for job *i*, *i* = 1..N

$$T_R^i = t_b^i - t_s^i$$

Relative response time

Relative response time = Response time / Execution time

Relative response time for job *i*, *i* = 1..N

$$T_{Rrel}^i = \frac{T_R^i}{T_{EXE}^i}$$

Average response time

$$T_R^{avr} = \frac{1}{N}\sum_{i=1}^{N} T_R^i$$

Average relative response time

$$T_{Rrel}^{avr} = \frac{1}{N}\sum_{i=1}^{N} T_{Rrel}^{i}$$

Average turn around time *is the average time from submitting a job till completing it. If the job generates an output and this output is to be transferred back to the submission host, then the job is considered completed only when the entire output has been transferred to the submission host.*

Turn around time

Turn around time = Delivery time – Submission time

Turn around time of job *i*, *i* = 1..N

$$T_{TA}^{i} = t_{d}^{i} - t_{s}^{i}$$

Relative turn around time

Relative turn around time = Turn around time / Execution time

Relative turn around time of job *i*, *i* = 1..N

$$T_{TArel}^{i} = \frac{T_{TA}^{i}}{T_{EXE}^{i}}$$

Average turn around time

$$T_{TA}^{avr} = \frac{1}{N}\sum_{i=1}^{N} T_{TA}^{i}$$

Average relative turn around time

$$T_{TArel}^{avr} = \frac{1}{N}\sum_{i=1}^{N} T_{TArel}^{i}$$

Throughput- *overall amount of work performed per time unit. In our experiments, throughput was measured using the number of jobs completed (including the transfer of the output to the submission host) in a unit of time.*

Throughput

$$Th = \frac{N}{T_{EXE}^{1..N}}$$

$$T_{EXE}^{1..N} = \max_{i=1..N}(t_d^i) - \min_{i=1..N}(t_s^i)$$

Utilization - *busy time span to available time span. Time span is measured in time*resource units. In our experiment, we will measure the utilization by measuring the average percentage of the CPU time used by JMS jobs on each execution host. These average machine utilizations will be then averaged for all execution hosts.*
An average CPU utilization by a given job in a specified period of time is obtained using the system utility "`top`".

Utilization of all machines

$$U = \frac{1}{M}\sum_{j=1}^{M} U_j^{avr}$$

Average CPU utilization of machine j

$$U_j^{avr} = \frac{1}{L}\sum_{k=1}^{L} U_j^k$$

Load imbalance- *is measured as the variability in the busy fraction of the time of different resources. In our experiment, we measured load imbalance as a standard deviation of the average CPU utilization of all execution hosts.*

Load imbalance

$$LI = \frac{1}{M}\sqrt{\sum_{j=1}^{M}(U_j^{avr} - U)^2}$$

Scheduling overhead of execution hosts

Average CPU utilization by JMS daemons averaged for all execution hosts

$$O_{EXE} = \frac{1}{M}\sum_{j=1}^{M} O_j^{avr}$$

Scheduling overhead of the master node

$O_M = O_j^{avr}$ for j = master node

Average scheduling overhead for machine j

$$O_j^{avr} = \frac{1}{L}\sum_{k=1}^{L} O_j^k$$

Appendix 3

Description of Timing Functions and Utilities

gettimeofday()

The function `gettimeofday()` gets the current time from the operating system. The time is expressed in seconds and microseconds elapsed since Jan 1, 1970 00:00 GMT. The actual resolution of the returned time depends on the accuracy of the system clock, which is hardware dependent.

Procedure takes as a parameter a pointer to `timeval` structure (defined in `sys/time.h`) and sets the proper values to its fields.

```
struct timeval
{
  long tv_sec;        /* seconds */
  long tv_usec;       /* microseconds */
};
```

Function returns 0 as an indicator of successful execution, or –1 when it fails. In the latter case `errno` variable holds the appropriate error code.

NTP

The Network Time Protocol (NTP) was developed to synchronize system clocks of computers connected using the TCP/IP network. NTP can work in the client-server and symmetric modes of operation. In the client-server mode a client synchronizes to a stateless server. In the symmetric mode peer computers synchronize their clocks to each other.

The protocol provides accuracy ranging typically from milliseconds (on LANs) to tenths of milliseconds (on WANs). To achieve such accuracy a node must be able to compensate a latency introduced by a network connection. Since it is impossible to measure one-way delay, unless clock offset between both nodes is known, the protocol measures roundtrip delay and assumes, that the delay is equal to a halt of this time. Typically a computer should use multiple NTP servers and diverse network connections to achieve high accuracy and reliability.

Top

`Top` is a program that continuously reports the state of the system, including a list of most CPU-hungry processes. It was originally developed as a UNIX equivalent and extension of a VMS command, which listed the top five busiest processes along with a bar graph of resource utilization. While the VMS command showed nothing besides the process name and the CPU percentage, `top` displays the generic information about the system and the processes that are currently running. By default, the processes are sorted by CPU utilization. The display is updated at the regular time intervals to reflect the current state of the system.

Top few lines shown on the screen contain the information about the state of the system, such as average load for the last 1, 5 and 15 minutes, the number of processes in each state (running, sleeping, starting, stopped and zombies) and the information about memory usage (both, physical and swap allocation). The remaining part of the screen

shows the information about individual processes, including name, process identifier, name of a user who owns it, state of the process, amount of allocated memory, total CPU time used, and the CPU utilization expressed as percentage of time assigned to the process since last update.

It can work in two modes: interactive and batch. In the interactive mode a user can issue a command (send a signal, change priority of a process or change the amount of information being displayed), while in the batch mode the behavior of the program is determined only by the command line options. In the latter mode, `top` prints the information to its standard output, so it can be captured and redirected to a file. For capturing statistics during experiments we are using the batch mode, so the results can be post-processed to retrieve required information.

Appendix 4

Comparative Experimental Measurements

The results of five experiments for LSF, PBS, and CODINE are collected in Tables A4-1, A4-2, A4-3, A4-4, and A4-5.

Two numbers in the scheduler overhead row represent respectively, the master daemons overhead and the execution host daemons overhead.

TABLE A4-1A: EXPERIMENT 1: MEDIUM JOB LIST – NO TIMESHARING

Average interval between job submissions = 30 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	3.2	9.7	13.3	**8.7**	2.9	2.8	2.8	**2.8**	29.4	34.2	28.5	**30.7**
Execution Time [s]	463.1	474.7	472.8	**470.2**	471.0	455.9	450.2	**459.0**	572.2	583.2	573.7	**576.4**
Turnaround time [s]	466.3	484.3	486.1	**478.9**	473.9	458.7	453.0	**461.9**	601.6	617.4	602.3	**607.1**
Throughput [jobs/hour]	91.8	81.2	68.8	**80.4**	75.8	68.3	66.8	**70.3**	N/A	65.3	69.9	**67.6**
Utilization [%]	N/A	64.1	47.2	**55.7**	47.3	38.0	38.8	**41.4**	N/A	57.7	82.2	**69.9**
Scheduler overhead (E/M) [%]	N/A	0.13 0.10	0.13 0.15	**0.1** **0.1**	0.005 0.515	0.005 0.34	0.005 0.32	**0.005** **0.4**	N/A	0.35 6.2	0.81 6.0	**0.6** **6.1**
Load Imbalance [%]	N/A	2.3	2.3	**2.3**	4.5	4.0	4.3	**4.3**	N/A	4.5	4.4	**4.45**
Relative response time	0.008	0.025	0.034	**0.022**	10.007	0.007	0.007	**3.340**	0.071	0.080	0.066	**0.072**
Relative turn-around time	1.008	1.025	1.034	**1.022**	1.007	1.007	1.007	**1.007**	1.071	1.080	1.066	**1.072**

TABLE A4-1B: EXPERIMENT 1: MEDIUM JOB LIST – NO TIMESHARING

Average interval between job submissions = 15 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	598.9	628.2	640.3	**622.5**	415.6	480.5	459.3	**451.8**	719.5	754.5	727.7	**733.9**
Execution Time [s]	500.6	507.2	496.4	**501.4**	476.2	489.5	509.5	**491.7**	572.6	565.3	540.3	**559.4**
Turnaround time [s]	1099.5	1135.5	1136.8	**1123.9**	891.8	970.1	968.9	**943.6**	1292.1	1319.9	1268.0	**1293.3**
Throughput [jobs/hour]	92.5	92.6	90.6	**91.9**	92.8	92.8	87.9	**91.2**	72.638	85.263	87.383	**81.8**
Utilization [%]	58.4	35.8	65.0	**53.0**	60.6	60.5	50.4	**57.2**	63.190	78.88	70.02	**70.7**
Scheduler overhead (E/M) [%]	0.23 0.17	0.13 0.16	0.13 0.30	**0.2** **0.2**	0.008 0.64	0.007 0.67	0.008 0.46	**0.01** **0.6**	0.34 4.63	0.96 6.45	0.36 7.31	**0.6** **6.1**
Load Imbalance [%]	0.9	7.9	3.3	**4.0**	2.0	2.0	2.7	**2.25**	4.8	2.9	3.65	**3.8**
Relative response time	1.520	1.596	1.591	**1.569**	1.077	1.242	1.168	**1.162**	1.672	1.757	1.677	**1.702**
Relative turn-around time	2.520	2.596	2.591	**2.569**	2.077	2.242	2.168	**2.162**	2.672	2.757	2.677	**2.702**

TABLE A4-1C: EXPERIMENT 1: MEDIUM JOB LIST – NO TIMESHARING

Average interval between job submissions = 5 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	1236.6	1293.7	1291.0	**1273.8**	972.0	998.4	980.6	**983.7**	1402.5	1376.3	1376.8	**1385.2**
Execution Time [s]	487.3	491.6	494.9	**491.3**	477.6	485.6	484.1	**482.4**	563.3	551.4	577.6	**564.1**
Turnaround time [s]	1724.0	1785.3	1785.9	**1765.1**	1449.7	1484.1	1464.8	**1466.2**	1965.8	1927.7	1954.5	**1949.3**
Throughput [jobs/hour]	91.4	114.2	115.4	**107.0**	92.1	100.7	113.0	**102.0**	92.9	90.8	74.7	**86.1**
Utilization [%]	63.3	77.0	78.4	**72.9**	63.5	66.2	72.5	**67.4**	74.6	70.6	89.6	**78.3**
Scheduler overhead (E/M) [%]	0.16 0.30	0.13 0.26	0.13 0.27	**0.1** **0.3**	0.008 0.86	0.009 0.97	0.010 0.72	**0.01** **0.9**	0.35 5.9	0.38 9.2	0.38 9.1	**0.4** **8.1**
Load Imbalance [%]	2.9	2.5	2.4	**2.6**	3.701	1.817	1.602	**2.4**	3.325	3.404	1.817	**2.8**
Relative response time	3.229	3.277	3.252	**3.253**	2.449	2.581	2.557	**2.529**	3.227	3.277	3.182	**3.229**
Relative turn-around time	4.229	4.277	4.252	**4.253**	3.449	3.581	3.557	**3.529**	4.227	4.277	4.182	**4.229**

TABLE A4-2: EXPERIMENT 2: MEDIUM JOB LIST – TWO JOBS PER CPU

Average interval between job submissions = 15 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	399.9	379.7	380.8	**386.8**	250.2	334.0	269.5	**284.6**	391.1	380.1		**385.6**
Execution Time [s]	918.6	956.6	855.0	**910.1**	980.4	1,021.9	963.8	**988.7**	1,100.2	1,093.0		**1,096.6**
Turnaround time [s]	1318.5	1336.3	1235.8	**1,296.9**	1,230.4	1,355.9	1,233.3	**1,273.2**	1,491.3	1,473.2		**1,482.3**
Throughput [jobs/hour]	83.7	67.1	120.2	**90.3**	80.3	80.5	80.1	**80.3**	69.9	64.6		**67.3**
Utilization [%]	58.5	54.8	76.5	**63.3**	56.9	58.4	N/A	**57.7**	63.0	N/A		**63.0**
Scheduler overhead (E/M) [%]	0.008 0.61	0.13 0.20	0.13 0.18	**0.09** **0.33**	0.007 0.55	0.008 0.57	N/A	**0.01** **0.56**	0.33 4.39	N/A		**0.33** **4.39**
Load Imbalance [%]	3.4	5.3	1.7	**3.47**	3.8	3.6	N/A	**3.70**	5.7	N/A		**5.70**
Relative response time	0.579	0.521	0.528	**0.543**	0.343	0.448	0.377	**0.389**	0.491	0.488		**0.490**
Relative turn-around time	1.579	1.521	1.528	**1.543**	1.343	1.448	1.377	**1.389**	1.491	0.488		**0.990**

TABLE A4-3A: EXPERIMENT 3: LONG JOB LIST – NO TIMESHARING

Average interval between job submissions = 2 min

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	12.3	13.1	13.1	**12.8**	2.9	2.9		**2.9**	2.4	3.8		**3.1**
Execution Time [s]	1,146.7	1,251.1	1,134.5	**1,177.4**	1076.3	1075.5		**1,075.9**	1,883.7	1,916.5		**1,900.1**
Turnaround time [s]	1,159.1	1,138.3	1,147.6	**1,148.3**	1079.3	1078.4		**1,078.9**	1,886.2	1,920.3		**1,903.3**
Throughput [jobs/hour]	21.5	26.3	26.0	**24.6**	25.6	25.6		**25.6**	20.3	15.0		**17.7**
Utilization [%]	43.0	46.2	40.3	**43.2**	45.2	46.2		**45.7**	60.9	42.5		**51.7**
Scheduler overhead (E/M) [%]	0.12 0.13	0.13 0.13	0.13 0.12	**0.13** **0.13**	0.004 0.42	0.004 0.25		**0.004** **0.34**	0.33 4.3	0.35 6.4		**0.34** **5.4**
Load Imbalance [%]	4.5	5.5	5.6	**5.2**	10.0	10.0		**10.0**	2.9	4.2		**3.6**
Relative response time	0.013	0.014	0.014	**0.014**	0.003	0.003		**0.003**	0.001	0.002		**0.002**
Relative turn-around time	1.013	1.014	1.014	**1.014**	1.003	1.003		**1.003**	1.001	1.002		**1.002**

TABLE A4-3B: EXPERIMENT 3: LONG JOB LIST – NO TIMESHARING

Average interval between job submissions = 30 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	871.9	851.0	857.2	**860.0**	801.2	796.0		**798.6**	1,456.6	1498.9		**1477.8**
Execution Time [s]	1,288.0	1350.6	1353.7	**1330.8**	1365.8	1363.1		**1364.5**	1,928.7	1918.1		**1923.4**
Turnaround time [s]	2,159.9	2201.7	2210.9	**2190.8**	2167.0	2159.2		**2163.1**	3,385.3	3417.0		**3401.2**
Throughput [jobs/hour]	36.2	23.5	23.1	**27.6**	29.9	29.9		**29.9**	N/A	22.7		**22.7**
Utilization [%]	60.5	53.9	53.4	**55.93**	57.4	58.2		**57.8**	N/A	63.8		**63.8**
Scheduler overhead (E/M) [%]	0.14 0.009	0.13 0.14	0.12 0.21	**0.13** **0.12**	0.004 0.63	0.004 0.44		**0.004** **0.54**	N/A	0.37 8.4		**0.37** **8.4**
Load Imbalance [%]	1.7	5.6	5.7	**4.3**	3.4	3.3		**3.35**	N/A	4.3		**4.30**
Relative response time	0.901	0.840	0.844	**0.862**	0.779	0.781		**0.780**	1.022	1.061		**1.042**
Relative turn-around time	1.901	1.840	1.844	**1.862**	1.779	1.781		**1.780**	2.022	2.061		**2.042**

TABLE A4-4A: EXPERIMENT 4: SMALL JOB LIST – NO TIMESHARING

Average interval between job submissions = 15 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	2.1	16.4	18.0	**12.2**	1.5	2.3	2.3	**2.0**	1.3	1.5	1.5	**1.4**
Execution Time [s]	27.4	26.9	26.6	**27.0**	29.0	32.6	32.8	**31.5**	28.2	27.2	28.1	**27.8**
Turnaround time [s]	29.6	43.3	44.6	**39.2**	30.5	34.9	35.1	**33.5**	29.5	28.8	29.6	**29.3**
Throughput [jobs/hour]	239.6	239.5	239.6	**239.6**	232.5	224.5	224.4	**227.1**	234.1	233.4	233.0	**233.5**
Utilization [%]	N/A	7.2	6.5	**6.9**	9.6	22.6	22.6	**18.3**	5.8	6.5	6.7	**6.3**
Scheduler overhead (E/M) [%]	N/A	0.12 0.12	0.12 0.20	**0.12** **0.16**	0.035 0.27	0.047 0.36	0.044 0.36	**0.04** **0.33**	0.32 1.0	0.32 1.1	0.32 1.1	**0.32** **1.1**
Load Imbalance [%]	N/A	1.4	1.7	**1.55**	1.6	8.2	8.2	**6.00**	0.9	0.8	0.7	**0.8**
Relative response time	0.201	1.657	1.725	**1.194**	0.139	0.214	0.212	**0.188**	0.129	0.183	0.160	**0.157**
Relative turn-around time	1.201	2.657	2.725	**2.194**	1.139	1.214	1.212	**1.188**	1.129	1.183	1.160	**1.157**

TABLE A4-4B: EXPERIMENT 4: SMALL JOB LIST – NO TIMESHARING

Average interval between job submissions = 10 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	2.4	11,5	10.5	**6.5**	1.5	2.5	2.5	**2.2**	1.2	1.3	1.3	**1.3**
Execution Time [s]	28.4	27.2	26.9	**27.5**	29.0	31.0	31.1	**30.4**	27.6	28.4	27.8	**27.9**
Turnaround time [s]	30.8	38.7	37.4	**35.6**	30.5	33.5	33.6	**32.5**	28.8	29.7	29.1	**29.2**
Throughput [jobs/hour]	356.5	354.2	356.3	**355.7**	232.5	319.2	319.2	**290.3**	341.7	335.1	335.3	**337.4**
Utilization [%]	10.1	9.4	10.3	**9.9**	N/A	25.3	25.3	**25.3**	9.3	9.1	8.1	**8.8**
Scheduler overhead (E/M) [%]	0.14 0.06	0.11 0.07	0.11 0.07	**0.12** **0.07**	N/A	0.06 0.45	0.05 0.49	**0.06** **0.47**	0.36 4.6	0.36 4.7	0.36 4.7	**0.36** **4.7**
Load Imbalance [%]	1.7	1.7	1.7	**1.7**	N/A	8.9	9.0	**8.95**	0.8	1.1	0.8	**0.9**
Relative response time	0.234	1.263	1.092	**0.863**	0.139	0.230	0.230	**0.200**	0.122	0.140	0.124	**0.129**
Relative turn-around time	1.234	2.263	2.092	**1.863**	1.139	1.230	1.230	**1.200**	1.122	1.140	1.124	**1.129**

TABLE A4-4C: EXPERIMENT 4: SMALL JOB LIST – NO TIMESHARING

Average interval between job submissions = 5 s

Timing parameter	LSF				PBS				Codine			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	2.8	9.9	10.8	**7.8**	2.5			**2.5**	1.2	1.4	1.4	**1.3**
Execution Time [s]	28.7	27.9	26.8	**27.8**	31.2			**31.2**	27.7	27.6	27.8	**27.7**
Turnaround time [s]	31.4	37.9	37.6	**35.6**	33.7			**33.7**	28.9	29.0	29.3	**29.1**
Throughput [jobs/hour]	N/A	665.3	655.9	**660.6**	531.9			**531.9**	615.4	605.9	600.3	**607.2**
Utilization [%]	N/A	16.6	15.7	**16.2**	30.8			**30.8**	17.0	15.9	15.5	**16.1**
Scheduler overhead (E/M) [%]	N/A	0.13 0.2	0.13 0.2	**0.13** **0.2**	0.07 0.8			**0.07** **0.8**	0.37 2.8	0.36 2.9	0.37 2.8	**0.37** **2.8**
Load Imbalance [%]	N/A	2.1	1.7	**1.9**	8.1			**8.1**	1.3	1.3	1.1	**1.2**
Relative response time	0.244	1.225	1.267	**0.912**	0.237			**0.237**	0.123	0.123	0.143	**0.130**
Relative turn-around time	1.244	2.225	2.267	**1.912**	1.237			**1.237**	1.123	1.123	1.143	**1.130**

TABLE A4-5: EXPERIMENT 5: I/O JOB LIST – STAGE-IN / STAGE-OUT

Average interval between job submissions = 30 s

Timing parameter	LSF				PBS			
Iteration	**I1**	**I2**	**I3**	**Avr**	**I1**	**I2**	**I3**	**Avr**
Response Time [s]	13.3	14.8	13.7	**13.9**				
Execution Time [s]	108.8	109.3	107.8	**108.6**				
Turnaround time [s]	122.2	124.2	121.5	**122.6**				
Throughput [jobs/hour]	217.5	219.5	218.3	**218.4**				
Utilization [%]	30.2	39.2	30.5	**33.3**				
Scheduler overhead (E/M) [%]	0.229 0.255	0.120 0.073	0.236 0.309	**0.20** **0.21**				
Load Imbalance [%]	5.2	2.0	5.4	**4.2**				
Relative response time	0.802	0.903	0.926	**0.877**				
Relative turn-around time	1.802	1.903	1.926	**1.877**				