# A Performance Study of Job Management Systems*

Tarek El-Ghazawi[1] , Kris Gaj[2], Nikitas Alexandridis[1], Frederic Vroman[1], Nguyen Nguyen[2],
Jacek R. Radzikowski[2], Preeyapong Samipagdi[1], and Suboh A. Suboh[1]
*[1] The George Washington University, [2] George Mason University,*
*tarek@seas.gwu.edu*

## Abstract

*Job Management Systems (JMSs) efficiently schedule and monitor jobs in parallel and distributed computing environments. Therefore, they are critical for improving the utilization of expensive resources in high-performance computing systems and centers, and an important component of grid software infrastructure. With many JMSs available commercially and in the public domain, it is difficult to choose an optimum JMS for a given computing environment.. In this paper, we present the results of the first empirical study of JMSs reported in the literature. Four commonly used systems, LSF, PBS Pro, Sun Grid Engine / CODINE, and Condor were considered. The study has revealed important strengths and weaknesses of these JMSs under different operational conditions. For example, LSF was shown to exhibit excellent throughput for a wide range of job types and submission rates. On the other hand, CODINE appeared to outperform other systems in terms of the average turn-around time for small jobs, and PBS appeared to excel in terms of turn-around time for relatively larger jobs.*

## 1. Introduction

A lot of work has been done in grid software infrastructure. One of the major tasks of this infrastructure is *job management*, also known as workload management, load sharing, or load management. Software systems capable of performing this task are referred to as *Job Management Systems* (*JMSs*).

Job Management Systems can leverage under-utilized computing resources in a grid computing like style. Most JMSs can operate in multiple environments, including heterogeneous clusters of workstations, supercomputers, and massively parallel systems. The focus of our study is performance of JMSs in a loosely coupled cluster of heterogeneous workstations.

Taking into account the large number of JMSs available commercially and in public domain, choosing the best JMS for particular type of distributed computing environment is not an easy task. All previous comparisons of JMSs reported in literature had only a conceptual character. In [1], selected JMSs were compared and contrasted according to a set of well defined criteria. In [2, 3,4], the job management requirements for the Numerical Aerodynamic Simulation (NAS) parallel systems and clusters at NASA Ames Research Center were analyzed and several commonly used JMSs evaluated according to these criteria. In [5,6,7], three widely used JMSs were analyzed from the point of view of their use with Sun HPC Cluster Tools. Finally, our earlier conceptual study, reported in [7,8,9] , gave a comparative overview and ranking of twelve popular systems for distributed computing, including several JMSs.

In this paper, we extend the conceptual comparison with the empirical study based on a set of well defined experiments performed in a uniform fashion in a controlled computing environment. To our best knowledge, this is a first reported experimental study quantifying the relative performance of several Job Management Systems.

Our paper is organized as follows. In Section 2, we give an introduction to Job Management Systems, and summarize conceptual functional differences among them. In Section 3, we define metrics used for comparison, present our experimental setup, and discuss parameters and role of all experiments. In Section 4, we describe our methodology and tools used for the measurement collection. Finally, in Sections 5 and 6, we present experimental results, their analysis, and we draw conclusions regarding the relative strengths and weaknesses of investigated JMSs.

## 2. Job Management Systems

### 2.1. General architecture of a JMS

The objective of a JMS, for an environment investigated in this paper, is to let users execute jobs on a non-dedicated cluster of workstations with a minimum impact on owners of these workstations by using computational resources that can be spared by the owners. The system should be able to perform at least the following tasks:

a. monitor all available resources,
b. accept jobs submitted by users together with resource requirements for each job,

*This work has been funded by the DoD under the LUCITE contract #MDA904-98-C-A081 and by the NSF INT-0109038

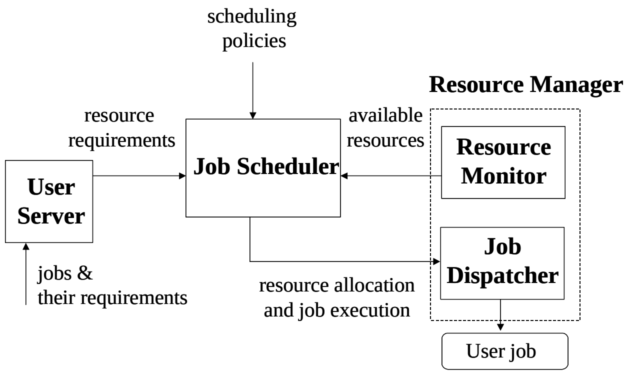


**Figure 1. Major functional blocks of a Job Management System**

c. perform centralized job scheduling that matches all available resources with all submitted jobs according to the predefined policies [10],

d. allocate resources and initiate job execution,

e. monitor all jobs and collect accounting information.

To perform these basic tasks, a JMS must include at least the following major functional units shown in Fig. 1:

1. *User server* – which lets user submit jobs and their requirements to a JMS (task b), and additionally may allow the user to inquire about the status and change the status of a job (e.g., to suspend or terminate it).
2. *Job scheduler* – which performs job scheduling and queuing based on the resource requirements, resource availability, and scheduling policies (task c).
3. *Resource manager* – used to monitor resources and dispatch jobs on a given execution host (tasks a, d, e).

## 2.2. Choice of Job Management Systems

More than twenty JMS packages, both commercial and public domain, are currently in use [1, 7]. For interest of time we selected four representative and commonly used JMSs

- LSF – Load Sharing Facility
- PBS – Portable Batch System
- Sun Grid Engine / CODINE, and
- Condor

The common feature of these JMSs is that all of them are based on a central Job Scheduler running on a single node.

LSF (Load Sharing Facility) is a commercial JMS from Platform Computing Corp. [11,12]. It evolved from Utopia system developed at the University of Toronto [13], and is currently probably the most widely used JMS.

PBS (Portable Batch System) has both a public domain and a commercial version [14]. The commercial version called PBS Pro is supported by Veridian Systems. This version was used in our experiments. PBS was originally developed to manage aerospace computing resources at NASA Ames Research Center.

Sun Grid Engine/CODINE is an open source package supported by Sun Inc. It evolved from DQS (Distributed Queuing System) developed by Florida State University. Its commercial version called CODINE was offered by GENIAS Gmbh in Germany and became widely deployed in Europe.

Condor is a public domain software package that was started at University of Wisconsin. It was one of the first systems that utilized idle workstation cycles and supported checkpointing and process migration.

## 2.3. Functional similarities and differences among selected Job Management Systems

The most important functional characteristics of selected four JMSs are presented and contrasted in Table 1. From this table, it can be seen that LSF supports all operating systems, job types, and features included in the table. CODINE lacks support for Windows NT, stage-in and stage-out, and checkpointing. PBS and Condor trail LSF and CODINE in terms of support for parallel jobs, dynamic load balancing and master daemon fault recovery. They also support a smaller number of operating systems compared to LSF.

## 3. Experimental Setup

### 3.1. Metrics

The following performance measures were investigated in our study:

**1. Throughput** is defined in general as a number of jobs completed in a unit of time. Since this number depends strongly on how many jobs are taken into account, we consider *throughput* to be a function of the number of jobs, *k*, and define it as *k* divided by the amount of time necessary to complete *k* JMS jobs (see Fig. 2a). We also define *total throughput* as a special case of throughput for parameter *k* equal to the total number of jobs submitted to a JMS during the experiment, *N* (see Fig. 2b).

In Fig. 3, we show the typical dependence of throughput on the number of jobs taken into account, *k*. It can be seen that throughput increases sharply as a function of *k* until the moment when either all system CPUs become busy, or the number of jobs submitted and completed in a unit of time become equal. When the

**Table 1. Conceptual functional comparison of selected Job Management Systems**

| | LSF | CODINE | PBS | Condor |
|---|---|---|---|---|
| **Distribution** | commercial | public domain | commercial and public domain | public domain |
| **Operating System Support** | | | | |
| Linux, Solaris | yes | yes | yes | yes |
| Tru64 | yes | yes | yes | no |
| Windows NT | yes | no | no | partial |
| **Types of Jobs** | | | | |
| Interactive jobs | yes | yes | yes | no |
| Parallel jobs | yes | yes | partial | limited to PVM |
| **Features Supporting Efficiency, Utilization, and Fault Tolerance** | | | | |
| Stage-in and stage-out | yes | no | yes | yes |
| Process migration | yes | yes | no | yes |
| Dynamic load balancing | yes | yes | no | no |
| Checkpointing | yes | using external libraries | only kernel-level | yes |
| Daemon fault recovery | master and execution hosts | master and execution hosts | only for execution hosts | only for execution hosts |

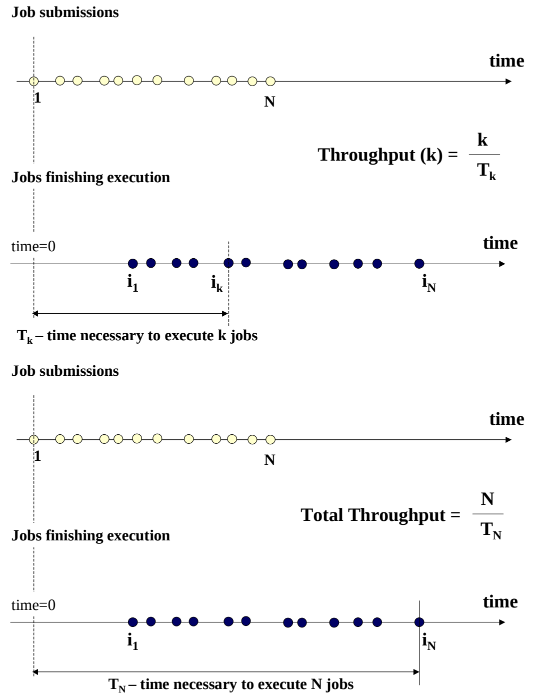


**Figure 2. Definition of (a) throughput and (b) total throughput**

number of jobs taken into account, *k*, gets close to the total number of jobs submitted during the experiment, the throughput drops sharply and unpredictably. This drop is the result of a boundary effect and is not likely to appear during the regular operation of a JMS, when the flow of jobs submitted to the JMS continues uninterrupted for a long period of time. Therefore, we decided to use for comparison

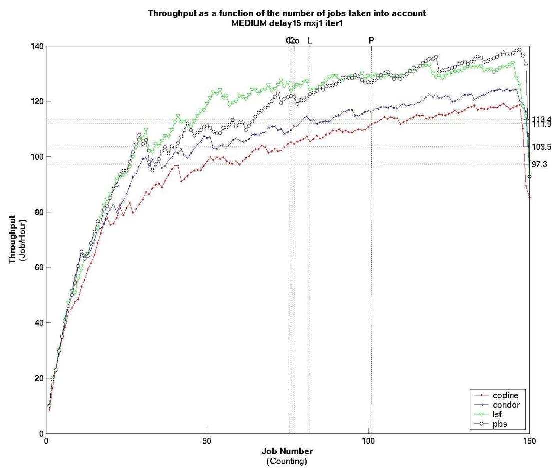


**Figure 3. Throughput as a function of number of jobs taken into account**

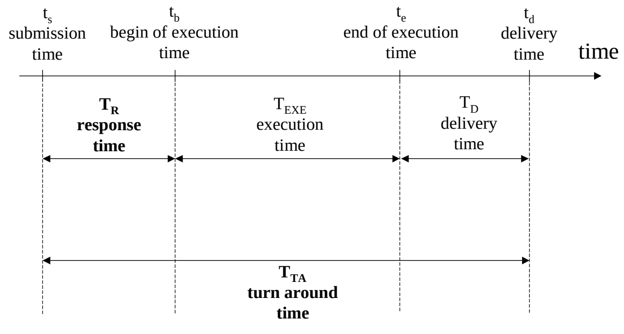


**Figure 4. Definition of timing parameters**

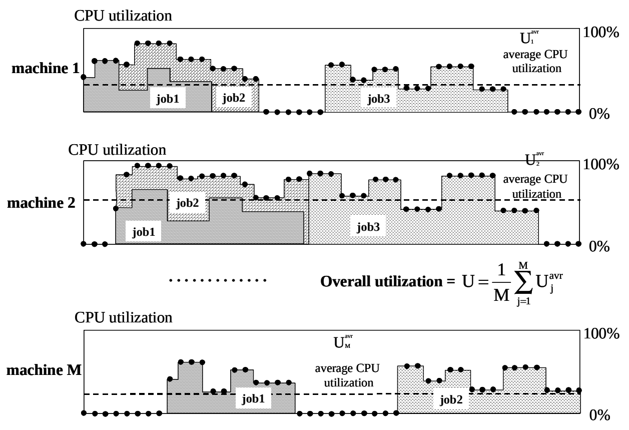


**Figure 5. Definition of the system utilization and its measurement using `top`**

*average throughput*, defined as *throughput* averaged over all possible values of the job number, *k*.

**2. Average turn-around time** is the time from submitting a job till completing it, averaged over all jobs submitted to a JMS (see Fig. 4).

**3. Average response time** is the average amount of time between submitting a job to a JMS and starting the job on one of the execution hosts (see Fig. 4).

**4. Utilization** is the ratio of a busy time span to the available time span. In our experiments, 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. 5).

## 3.2. Our Micro-grid testbed

A Micro-grid testbed used in our experiments is shown in Fig. 6. The testbed consists of 9 PCs running Linux OS, and 4 workstations Ultra 5, running Solaris 8.

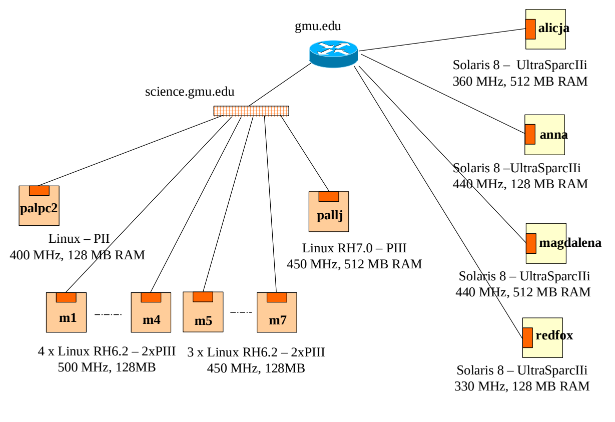


**Figure 6. A Micro-grid testbed used in the experimental study**

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.

## 3.3. Application 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. Each benchmark has been characterized in terms of the CPU time, wall time, and memory requirements using one of the Linux machines.

All benchmarks have been divided into the following three 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.

## 3.4. Experiments

Each experiment consists of running N jobs chosen pseudo-randomly from the given set of benchmarks, and submitted one at a time to a given JMS in the pseudo-random 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 only job requirement 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.

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

**Table 2. Characteristic features of experiments performed during our study**

| **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 3, Long job list | 16 min 51 s | 2 min, 30 s | 75 | one job / CPU |
| 3 | Set 1, Short job list | 22 s | 15 s, 10 s, 5 s, 2 s, 1 s | 150 | one job / CPU |
| 4 | Set 2, Medium job list | 7 min 22 s | 15 s | 150 | two jobs / CPU |
| 5 | Set 1, Short job list | 22 s | 5 s | 150 | one job / CPU; emulation of daemon faults |

Experiment 2, 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 four JMSes, under exactly the same initial conditions, including the same initial seeds of the pseudo-random 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 job submission rates. These rates have been chosen experimentally in such a way that they correspond to qualitatively different JMS loads. For the smallest submission rate, 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 highest submission rate, 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.

The characteristic features of five experiments performed during our experimental study are summarized in Table 2. Experiments 1-4 were designed to measure the performance of each JMS for different job submission rates.

Experiment 5 was aimed at quantifying fault tolerance of each JMS by determining its resistance against the master and execution daemon failures. Five minutes after the beginning of this experiment, master daemons on a master host or execution host daemons on a single execution host were killed. In one version of the experiment, the killed daemons were restarted one minute later, in the other version, no further action was taken. In all cases, the total number of jobs that completed execution was recorded and compared with the total number of jobs submitted to the system for execution.

## 3.5 Common settings of Job Management Systems

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

*A. Maximum 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 4, where two jobs were allowed to share the same CPU.

*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 for LSF and CODINE were set based on the relative performance of benchmarks representing a typical load. For each list of benchmarks, two representative benchmarks were selected, and run on all machines of distinctly different types. 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 and long job lists. The CPU factors of Condor were computed automatically by this JMS based on the Condor-specific benchmarks running on the execution hosts in the spare time. The CPU factors in LSF, CODINE, and Condor affect the operation of the scheduler. In PBS, the equivalent parameter has no effect on scheduling, and affects only accounting and time limit enforcement.

*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, CODINE, and Condor on the other side use a different definition of this parameter. In all systems, this parameter describes the maximum time in seconds between subsequent attempts to schedule jobs. However, in PBS, CODINE, and Condor the attempts to schedule a job 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.

*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 JMSs.

## 4. Methodology and measurement collection

Each experiment was aimed at determining values of all performance measures defined in Section 3.1. All parameters were measured in the same way for all JMSs, 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 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.

The Unix Network Time Protocol (NTP) was used to synchronize clocks of all machines of our Micro-grid. The protocol provides accuracy ranging from milliseconds (on LANs) to tenths of milliseconds (on WANs).

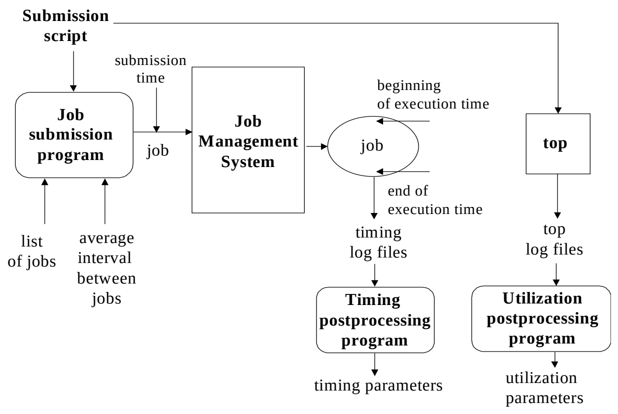


**Figure 7. Software used to collect performance measures**

In order to determine the JMS utilization, the Unix `top` utility was used. 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. 5). 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.

Three programs were developed to support the experiments and were used in a way shown in Fig. 7. A C++ Job Submission 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 post-processing Perl scripts, Timing and Utilization post-processing utilities, 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 Micro-Grid testbed.

## 5. Experimental Results

Two most important parameters determining the performance of a Job Management System are turn-around time and throughput. *Throughput* is particularly important when a user submits a large batch of jobs to a JMS and does not do any further processing till all jobs complete execution. *Turn-around time* is particularly important when a user

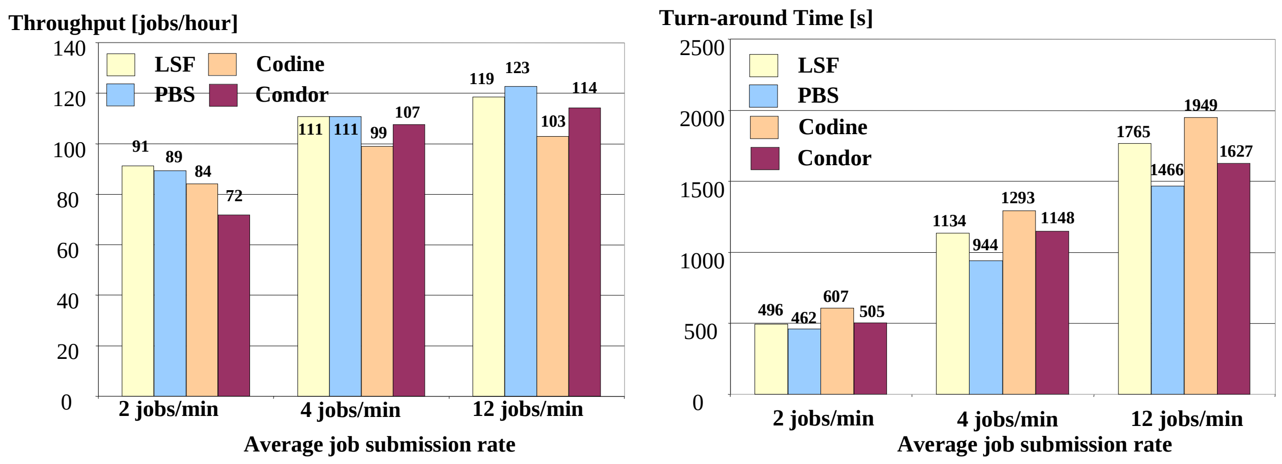


**Figure 8. Average throughput and average turn-around-time for the medium job list**

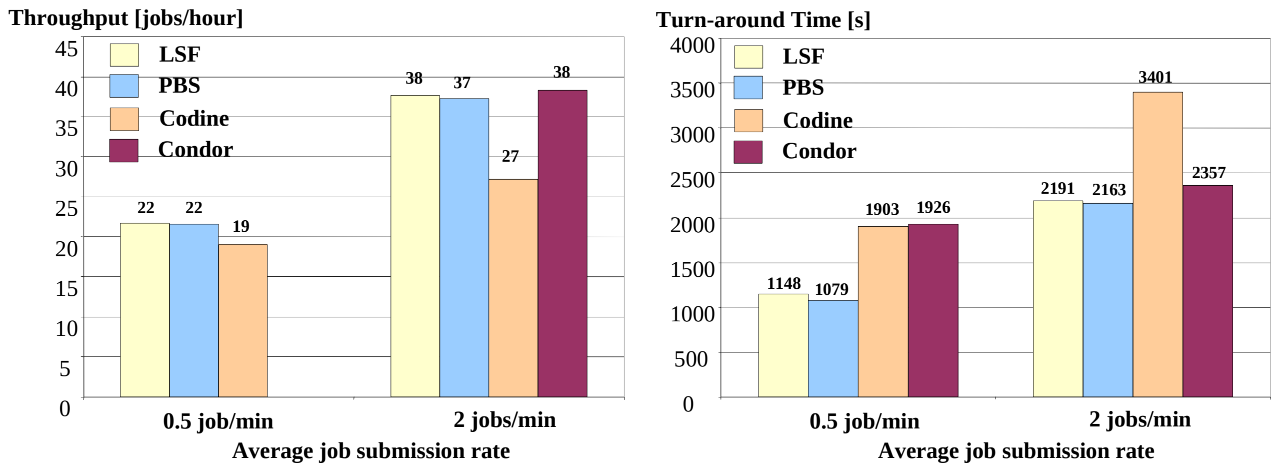


**Figure 9. Average throughput and average turn-around-time for the long job list**

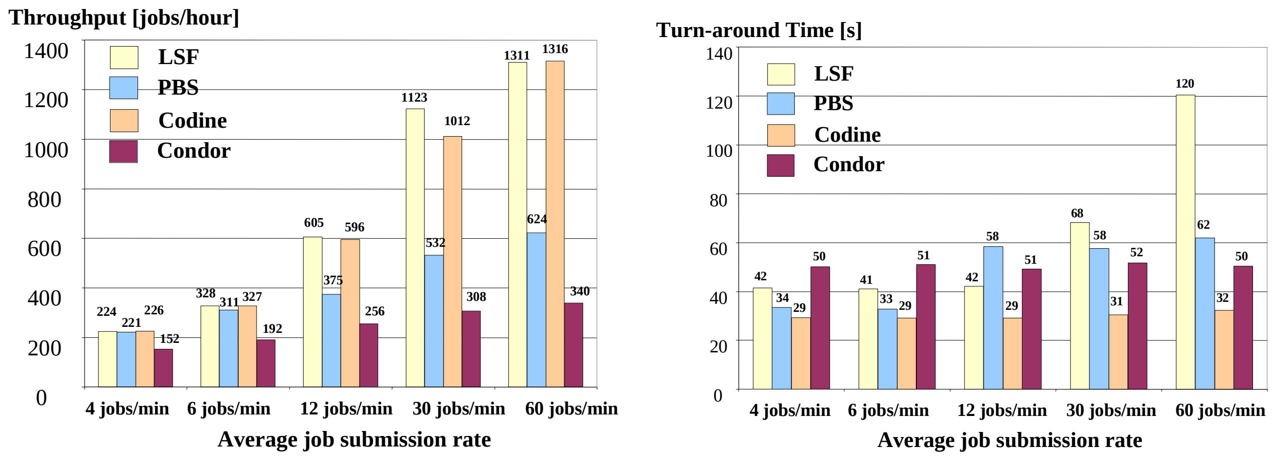
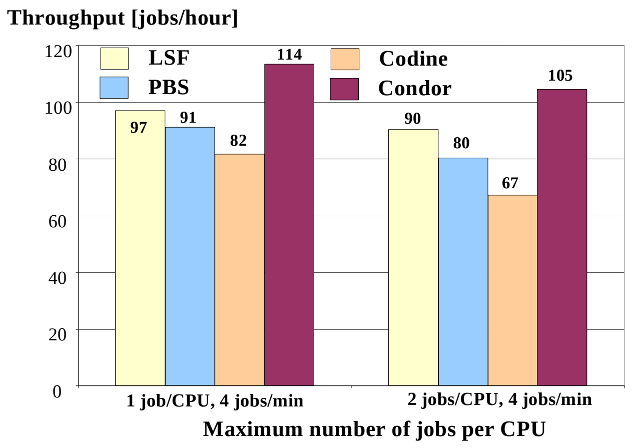


**Figure 10. Average throughput and average turn-around-time for the short job list**



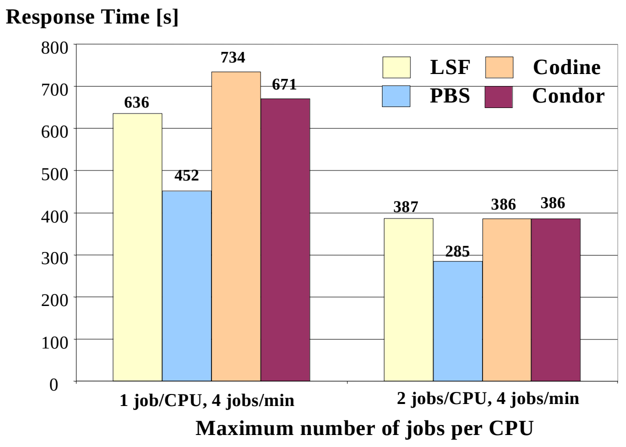


**Figure 11. Average partial throughput and response time for medium jobs with two jobs allowed to share a single CPU**

**Table 3. Results of Experiment 5 regarding daemon fault recovery**

| JMS | Daemons killed 5 minutes after the beginning of the experiment | | | |
|---|---|---|---|---|
| | Master daemons | | Execution host daemons | |
| | Not restarted | Restarted 1 min later | Not restarted | Restarted 1 min later |
| **LSF** | Master daemons automatically restarted, no jobs lost | Master daemons automatically restarted before 1 min, no jobs lost. | No jobs lost | No jobs lost |
| **PBS** | Daemons did not restart, all later jobs lost | 12 jobs submitted during the daemon down time lost | No jobs lost. | No jobs lost |
| **CODINE** | Master daemons automatically restarted, no jobs lost | Master daemons automatically restarted before 1 min, no jobs lost. | No jobs lost. | No jobs lost |
| **Condor** | Daemons did not restart, all later jobs lost | 153 out of 150 jobs finished execution | No jobs lost | No jobs lost |

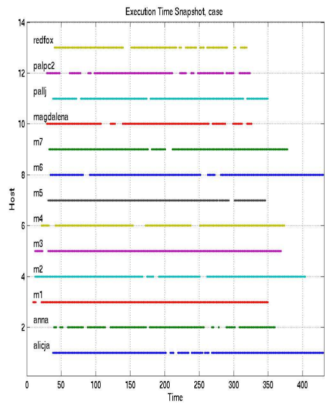
tends to work in a pseudo-interactive mode and awaits results of each subsequent experiment. Additionally, an *average response time* might be important in case of interactive jobs that require user input and thus a constant presence of the user.

In Experiment 1, with the medium job list, LSF and PBS were consistently the best in terms of the average throughput, while PBS was the best in terms of the average turn-around time. The overall differences among all four systems were smaller than 21% for average throughput, and 33% for average turn-around time.

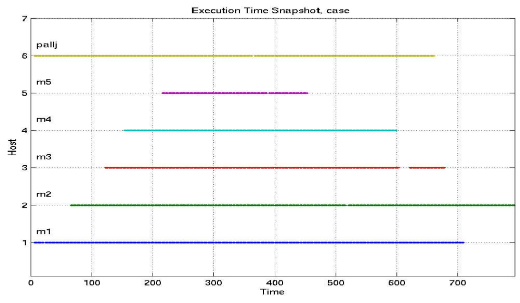
In Experiment 2, with the long job list, the throughputs of LSF, Condor, and PBS were almost identical, while CODINE was trailing by 29%. The turn-around time was the best for PBS and LSF for both investigated submission rates. For the higher submission rate, the performance of Condor, was approximately the same as performance of two best systems.

In Experiment 3, for short job list, the throughputs of LSF and Codine were the highest of all investigated systems. At the same time, the turn-around time was consistently the best for CODINE.

The analysis of the system utilization and job distribution revealed the following reasons for the different relative performance of each system in terms of throughput and turn-around time. LSF tends to dispatch jobs to all execution hosts, independently of their relative speed (see Fig. 12a). It also uses a complex algorithm for scheduling, which guarantees that jobs are executed tightly one after the other. Both factors contribute to high throughput. At the same time, distributing jobs to all machines, including slow ones, increases average



**(a)**



**(b)**

**Figure 12. Utilization of machines by a) LSF and b) PBS.**

execution time, and complex scheduling affects average response time. Both factors contribute to the increase in the average turn-around time. On the other hand, PBS distributes jobs only to a limited number of the fastest execution hosts (see Fig. 12b). As a result, the average execution time is smaller compared to LSF, which contributes to a better average turn-around time. At the same time, the limited utilization of the execution hosts contributes to only average throughput. Finally, Condor has a scheduling algorithm that seems to be more suitable for longer jobs. Although all execution hosts are utilized, nevertheless, there are significant gaps between the times when one job finishes execution and another job is dispatched to the same execution host.

In Experiment 4, the configuration parameters of each system were changed in order to allow two JMS jobs to execute on each execution host at the same time. Since all jobs used in our study are numerical, and have limited input/output, timesharing of jobs could not improve either average throughput or average turn-around time. In fact, these parameters deteriorated by a factor of 7 to 18%, depending on the JMS, because of the redundancy associated with context swapping. The only parameter that improved was the average response time that decreased by a factor ranging from 1.6 to 1.9 times.

In Experiment 5 (see Table 3), LSF was shown to be resistant against the master daemon failure. When the master daemons were killed, they were automatically restarted shortly after. No jobs were lost. PBS and Condor, do not have the capability to restart killed master daemons but they resume normal operation after these daemons are restarted manually. The interruption has a limited effect on the number of jobs that complete execution. All JMSs appeared to be resistant against the failure of execution host daemons. These daemons were not automatically restarted, but as a result of their failure, affected jobs were redirected to other execution hosts.

## 6. Summary and Conclusions

The summary of the performance of all investigated Job Management Systems as a function of the job size and the job submission rate is given in Tables 4 and 5.

**Table 4. JMS ranking in terms of the average throughput - summary (B - relatively best performance, W - relatively worst performance)**

| Job size | Submission rate | | |
|---|---|---|---|
| | Low | Medium | High |
| **Large** | B: LSF, PBS<br>W: CODINE (-14%) | B: Condor, LSF, PBS<br>W: CODINE (-29%) | |
| **Medium** | B: LSF, PBS<br>W: CODINE (-21%) | B: LSF, PBS<br>W: CODINE (-11%) | B: PBS, LSF<br>W: CODINE (-17%) |
| **Small** | B: LSF, CODINE<br>W: Condor (-57%) | B: LSF, CODINE<br>W: Condor (-69%) | B: LSF, CODINE<br>W: Condor (-71%) |

**Table 5. JMS ranking in terms of the average turn-around time - summary (B - relatively best performance, W - relatively worst performance)**

| Job size | Submission rate | | |
|---|---|---|---|
| | Low | Medium | High |
| **Large** | B: PBS, LSF<br>W: Condor, CODINE (+78%) | B: PBS, LSF<br>W: CODINE (+57%) | |
| **Medium** | B: PBS<br>W: CODINE (+31%) | B: PBS<br>W: CODINE (+37%) | B: PBS<br>W: CODINE (+33%) |
| **Small** | B: CODINE<br>W: Condor (+72%) | B: CODINE<br>W: PBS (+119%) | B: CODINE<br>W: LSF (+275%) |

Based on Tables 4 and 5, and Figures 8-11, we can draw the following conclusions. For large jobs with medium submission rate, Condor has compared favorably with the rest of the systems. In terms of the average system throughput, LSF appears to offer the best performance for all job sizes and submission rates. In terms of the average turn-around time, PBS is the best for large and medium jobs, but CODINE outperforms it for short jobs.

The relative performance of Job Management Systems was similar for medium and large jobs, and changed considerably for short jobs where the job execution times became comparable with the times required for resource monitoring and job scheduling. CODINE appeared to be particularly efficient for small jobs, while the relative performance of PBS and Condor improved with the increase in the job size and the job submission rate. LSF was the only system that performed uniformly well for all job sizes and submission rates with the exception of the turn-around time for small jobs and large submission rates.

Despite the limitations resulting from the relatively small size of our Micro-Grid testbed and a limited set of system settings exercised in our experiments, the practical value of our empirical knowledge comes, among the other, from the following factors:

- 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.
- 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, to our best knowledge, our study is the first empirical study of Job Management Systems reported in the literature. Our methodology and tools developed as a result of this project may be used by other groups to extend the understanding of similarities and differences among behavior and performance of existing Job Management Systems.

## References

[1] M. A. Baker, G. C. Fox, and H. W. Yau, “Cluster Computing Review,” NPAC Technical Report SCCS-748, Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.

[2] J. P. Jones, “NAS Requirements Checklist for Job Queuing/Scheduling Software,” NAS Technical Report NAS-96-003 April 1996.

[3] J. P. Jones, “Evaluation of Job Queuing/Scheduling Software: Phase 1 Report,” NAS Technical Report, NAS-96-009, September 1996.

[4] M. Papakhian, "Comparing Job-Management Systems:The User's Perspective," IEEE Computational Science & Engineering, vol.5, no.2, April-June 1998.

[5] C. Byun, C. Duncan, and S. Burks, "A Comparison of Job Management Systems in Supporting HPC Cluster Tools," Proc. SUPerG, Vancouver, Fall 2000.

[6] O. Hassaine, "Issues in Selecting a Job Management Systems (JMS)," Proc. SUPerG, Tokyo, April 2001.

[7] T. El-Ghazawi, Gaj, Alexandridis, Schott, Staicu, Radzikowski, Nguyen, and Suboh, *Conceptual Comparative Study of Job Management Systems*, Technical Report, February 2001.
http://ece.gmu.edu/lucite/reports.html

[8] T. El-Ghazawi, Gaj, Alexandridis, Schott, Staicu, Radzikowski, Nguyen, Vongsaard, Chauvin, Samipagdi, and Suboh, *Experimental Comparative Study of Job Management Systems*, Technical Report, July 2001.
http://ece.gmu.edu/lucite/reports.html

[9] A. V. Staicu, J. R. Radzikowski, K Gaj, N. Alexandridis, and T. El-Ghazawi, "Effective Use of Networked Reconfigurable Resources," Proc. 2001 MAPLD Int. Conf., Laurel, Maryland, Sep. 2001.

[10] Rajesh Raman, Miron Livny, and Marvin Solomon, "Resource Management through Multilateral Matchmaking", Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, PA., August 2000.

[11] I. Lumb, "Wide-area parallel computing: A production-quality solution with LSF," Supercomputing 2000, Dallas, Texas, Nov. 2000.

[12] LSF, http://www.platform.com/products/LSF/

[13] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: A load sharing facility for large heterogenous distributed computer systems," *Software Practice and Experience*, vol.23, no.12, Dec. 1993.

[14] Portal Batch System (PBS), http://pbspro.com