George Mason University

CONCEPTUAL 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, and Nguyen Nguyen **,** Suboh A. Suboh (GWU)

February 21, 2001

OUTLINE

EXECUTIVE SUMMARY

This report is part of our effort to create a distributed computing software system for the effective utilization of networked reconfigurable computing resources. The objective is to construct a system that can leverage under-utilized resources at a given time to serve other users who currently have the needs, in a grid computing like style. In order to take advantage of previous related works, our effort started with a conceptual study that aimed at the comparative evaluation of currently available job management systems (JMS). Twelve systems were considered in this study and evaluated with respect to their suitability for being extended to handle FPGA reconfigurable computing resources and tasks. The compared systems are LSF, PBS, Condor, RES, Compaq DCE, Sun Grid Engin/CODINE, Globus, Legion, NetSolve, MOSIX, AppLES, and NWS.

In order to facilitate the comparison, twenty-five major requirements were identified, and a scoring system was developed to gauge the compliance of these systems to the requirements, and rank them accordingly. As many objective facts about these systems are hard to communicate through simple numbers, the scoring is only considered a guideline. Therefore, pros and cons of each of the systems as well as summary information were compiled into comparative tables. In addition, the 12 systems were also clustered into several categories in order to easily appreciate the functional and operational differences among them and help the decision-making.

Based on the information compiled, the top five systems, and therefore candidates for further consideration, were found to be LSF, CODINE, PBS, Condor, and Globus. The first four of these systems belong to the same category, Centralized Job Management Systems, in which scheduling is done centrally. Any of these systems can be potentially extended to meet the objectives of our project. Globus, however, belongs to the category of Job Management Systems without a Central Scheduler. Globus is selected because of its superior flexibility and interoperability with other JMS systems and modules. Our strategic goal is to exploit the open architecture of Globus and investigate the possibility of using it to extend the capabilities and scale the size of clusters used for reconfigurable computing, beyond what the adopted centralized JMS can do.

1. INTRODUCTION

This study is aimed at guiding the development of a job scheduling and management system that can exploit networked reconfigurable computing resources, by allowing users to discover and deploy jobs onto under-utilized remote reconfigurable computers. As many currently available systems are likely to embody significant related functionality, a study of the relative suitability of such systems for being extended and used for this purpose is a logical first step. This report presents the findings of such study that considered the twelve most relevant systems to our purpose. These are LSF, Sun Grid Engine, PBS, Condor, RES, MOSIX, GLOBUS, LEGION, NetSolve, AppLES, NWS, and Compaq DCE.

Conducting a comparative study addressing job management systems can be approached in many different ways and many options exist for producing and presenting the findings. However, in order to maximize the benefit, we have selected to follow, in spirit, some of the previous related works. This section describes the previous studies that were somewhat leveraged, and how this report is structured.

1.1 Related Comparative Studies

Two main previous studies have been considered in the light of the objectives of our project. These are the "cluster computing survey" by Baker et. al. [BFY95] and the Job Queuing/Scheduling Software comparative studies conducted by by Jones and Brickell at NAS, NASA Ames Research Center[NAS] and [NAS96]. In addition, the example requirements list compiled by NSA and discussed in the December 2000 project meeting, were also considered.

Many of the criteria, definitions, and weighting (scoring) are based on these aforementioned studies. Two important distinctions, however, distinguish the work in this report. These are the different purpose of our work and the fact that many new relevant systems have been recently introduced, mainly in the grid computing context.

1.2 Organization of this Report

This report follows a top down approach, which conveys the study findings in a hierarchical manner starting with an executive summary, then a summary of findings section, followed by detailed system descriptions and evaluations.

Section 2, fully specifies how the detailed study and evaluation of each one of the systems considered is carried out. To this end, section 2 can be considered as the evaluation template used for analyzing each of the systems. Each of the detailed system evaluations provided in Appendix A has greatly adhered to this evaluation template. In addition to being the evaluation template, section 2 provides a precise definition for each of the system requirements and attributes considered.

Section 3, Summary of Findings, provides the next level of details of this study after the Executive Summary. Due to the wide range of systems considered, section 3 starts with a classification of these systems based on their main functional differences. Comparative tables are then presented and discussed to provide an overview of the main differences. They also state the pros and cons of each system, along with the computed score, which attempts to reflect the degree of conformance of each of these systems to the defined requirements. It should be noted, however, that many significant system issues might be hard to capture with a straightforward scoring system. Therefore, the section closes with the study recommendation, which takes into account the scoring, as well as the objective issues that are hard to translate into numbers.

Appendix A provides for each of the considered twelve systems a detailed description and evaluation based on the template requirements and definitions of section 2.

2. JOB MANAGEMENT SYSTEM REQUIREMENT DEFINITIONS

2.1 Introduction

The system requirements in this section were partially derived from previous studies, augmented with additional considerations, and organized in accordance with the objectives of our project. The definitions and the organization in the rest of this section provide an evaluation template that allows the examination of the considered systems in a consistent manner that simplifies the cross comparison and simplify the differentiation among the systems. Section 2.2 requires standard background information about each of the systems. Section 2.3 establishes common means to describe the functional and architectural attributes of each system. In section 2.4, twenty five major requirements are selected and defined. These requirements constitute the most relevant system characteristics needed in the target distributed system for reconfigurable computing. Each of these requirements is assigned a weight of 4 points, with a total score of 100 per system. The scoring system was also greatly inspired by the work at NAS, NASA Ames Research Center, which helps the ranking of the systems considered. Requirements in this section that directly relate to those used in the NAS studies will be provided along with reference to the specific source of information including the requirement number in the NAS reports. As the degree of conformance of each of these systems may not be occasionally well documented in the literature, we point that the scores should not be used alone. Instead, they must be considered in conjunction with the tabulated comparative information and the clustering of the studied systems into functional groups.

2.2 Background Information

A. Authors
Company/organization that designed the first version of the product.

B. Support
Company/organization currently supporting the development.

C. Distribution
Commercial vs. public domain.

D. License
Type of license needed (commercial license, general public license, etc.)

2.3 System Description

E. Type of the system

We have identified the several major types of systems suitable for grouping the compared systems into meaningful clusters. Please note that this is not intended to be an exhaustive or generalized classification, but rather one that can capture the major differences in our twelve compared systems. The identified categories are listed below:

a. Centralized Job Management Systems

A system in this category includes at least three major modules: user server, job scheduler, and resource manager (as defined in point F). Control, in this case, is provided by the central job scheduler. Examples: LSF, CODINE, PBS, Condor, RES.

b. Distributed Job Management Systems without a central job scheduler

A system in this category includes at least the user server and resource manager modules. A job scheduler is not present or is local for each user.
Examples: Legion, Globus, Netsolve.

c. Distributed Operating Systems

A system in this class operates on processes and not jobs. Such systems are used primarily for tightly coupled clusters of homogeneous workstations. Example: MOSIX

d. Parameter Study Schedulers

This category includes systems that are geared towards one specific application, parameter study applications in this case. Example: AppLES

e. Distributed Computing Interfaces

These are systems that mainly provide interoperability support for distributed applications. Example: DCE

f. Resource Monitors and Forecasters

These are systems that have only one or two relevant functional modules, in this case only resource monitors and forecasters. Example: NWS

F. Major Modules of a Job Management System

A job management system should include at least the following three modules:

a. User server (known also as *job server*) - lets the user submit jobs to one or more queues, specify resource requirements for each job, delete a job from the queue, inquire about the status of a job or a queue.

b. Job scheduler - performs job scheduling and queuing according to job types, resource requirements, resource availability, and scheduling policies.

c. Resource manager (known also as *resource allocation manager*) - allocates and monitors resources, enforces scheduling policies, and collects accounting information.
This module consists typically of the two functional units:
- **Resource monitor**, and
- **Job dispatcher** (known also as *job executor*).

Additionally, a job management system may include the following modules:

d. Checkpointing module

This is a module that supports storing and retrieving the checkpointing information.

e. Load balancing module

This is a module that supports dynamic load balancing.

f. Security module
This is a module that allows the user to authenticate to JMS, provides authorization, i.e., verification and enforcement of access rights; and encryption of communication links among JMS modules.

g. Fault-detection module – A module responsible for detecting failures of any hardware or software component of a system.

G. Functional flow

Functional flow defines the interaction among all modules that form the system.

H. Distribution of control

Central vs. distributed control. Module(s) responsible for control.

I. Distribution of information services

Information about the system resources can be stored in one central location, or distributed among multiple nodes. If the information is distributed, an attempt may be made to store the same information in all nodes, or store only partial information in every node and download the remaining information only when needed.

J. Communication

Communication among the JMS modules can be supported using either of the following methods.

a. Protocols
Upper level (above TCP/IP) protocols used for data transfer among JMS modules, among JMS and JMS jobs, and among processes of the same job running on multiple machines (e.g., rpc, ftp, http, shttp, rcp).

b. Shared file systems
Interaction among the JMS modules, among jobs, as well as between JMS modules and jobs can be supported through a shared file system, such as NFS or AFS.

K. Interoperability

A JMS can interoperate with another system in one or more ways as follows
a. JMS may be able to run as a part of another system
b. JMS can use another system in place of one or more of its modules
c. JMS can run concurrently with another system

2.4 Major Requirements

1. **Availability and quality of:**
   a. binary code
   b. source code
   c. documentation
   d. roadmap, or plans for future development of the system.
   e. training
   f. customer support

[NAS 3.1.7]

2. **Operating systems/platforms**
   a. submission
   b. execution
   c. control

   Required operating systems: Linux, Windows NT, Solaris, True64UNIX, others.

[NAS 3.2.1, 4.1.17]

3. **Batch job support**

   Batch jobs usually run overnight, so that results can be retrieved the next day. Their output is sent to a file rather than the terminal. Batch jobs require a larger resource allocation.

[NAS 3.4.1]

4. **Interactive job support**

   Interactive jobs are those that require fast turn-around (e.g. for debugging) with their input, output and error file streams connected to a terminal. Interactive jobs would usually have small resource requirements, so they can run immediately or in a short period of time.

[NAS 3.2.8, 3.4.1, 4.1.11]

5. **Parallel job support**

   A job management system must be "parallel aware," i.e. understand the concept of a parallel job and maintain complete control over that job. This capability requires:

   - *Tracking all processes (and sub-processes) of the job.*
   - *Ability to kill any job completely, including sub-processes, without leaving orphaned processes. This implies that the JMS must be aware of distributed processes and capable of forwarding signals.*
   - *Ability to "clean up" after jobs, i.e. provide node condition equivalent to the state before the given job existed.*
   - *Collecting complete job accounting information for all processes of a job, which must be combined to provide an aggregate job accounting record, in addition to per-node totals. The job accounting record must indicate total usage of all resources allocated to the job, and which limits, if any, were exceeded.*

- *Providing a mechanism (i.e. a programming interface), which allows a parallel program to communicate with the JMS to coordinate resource usage and to start processes.*

[NAS 3.2.1]

6. Resource requests by users

During job submission, a user must be able to request resource allocations specifying:

- *Number of CPUs per job*
- *Number of nodes per job*
- *Type of nodes per job*
- *Wall clock time*
- *CPU time (per node and per application)*
- *System time*
- *Memory utilization*
- *Disk usage*
- *Swap space*
- *Dedicated access*
- *Shared access*
- *Network adapter access*
- *Network utilization*
- *Disk access*
- *Type of network connection*

[NAS 3.1.5]

7. Limits on resources by administrators

a. general

b. specific users or groups

Access control filters should include restrictions for:

- *Number of CPUs per job*
- *Number of nodes per job*
- *Type of nodes per job*
- *Wall clock time*
- *CPU time (per node and per application)*
- *System time*
- *Memory utilization*
- *Disk usage*
- *Swap space*
- *Dedicated access*
- *Shared access*
- *Network adapter access*
- *Network utilization*
- *Disk access*

[NAS 3.1.5]

8. File stage-in and stage-out

File "stage-in" and "stage-out" capability allows the user to identify files that should be transferred to and from appropriate locations on the computer system on which his/her job will be run. Stage-in needs to occur before the actual job starts, but after disk resources have been allocated for that job. Stage-out should follow termination of the job, but before the disk resources are released for re-allocation.

[NAS 3.2.3]

9. Flexible scheduling

Because user's resource requirements are complex, a JMS should have a flexible scheduler interface to implement complex scheduling rules.

JMS should support:

a. Several scheduling policies

Simple, out-of-the-box scheduling policies, including:

- *FIFO*
- *Shortest job first*
- *Favor large memory (or CPU) jobs, or small jobs*
- *Favor long running jobs, or short jobs*
- *User or group priority*
- *Fair sharing (past usage consumption)*

b. Policies changing in time

- *Dynamic policy changes (from any computable scheduling policy to any other without restarting the batch system).*
- *Allows different scheduling policies at different times of the day and distinction between prime and non-prime time.*

c. Scheduling multiple resources simultaneously

Including at least the following:

- *number of nodes*
- *type of nodes*
- *number of processors per node*
- *memory per node*
- *network connections*
- *disk*
- *OS version*
- *system specific resources (e.g. switch adapter mode on SP2)*

d. highly configurable scheduling

- *Dynamic and preemptive resource allocation (reshuffling queue, tiling etc.)*
- *Awareness/distinction between batch, interactive and foreign jobs.*
- *Custom scheduling policies*

[NAS 3.3.1, 3.3.2, 3.3.3, 4.1.1]

10. Job priorities

How many priority levels are there (both inter-queue and intra-queue)?

- **User assigned**

A user should be able to order his jobs using a priority scheme.

- **System assigned**

The system should have the ability to assign different priorities to different users or user groups.

[NAS 3.4.2]

11. Timesharing

In the "dedicated" or "space sharing models" **[HWA98]***, only one user process is allocated to a node. However, the system processes or daemons are still running on the same node. In the* ***timesharing*** *mode, multiple user processes are assigned to run concurrently on the same node.*

a. processes

Multiple processes per node.

b. jobs

Multiple JMS jobs per node.

[NAS 3.3.7, 3.3.8]

12. Impact on owners of computational nodes

Local users should not be influenced by the presence of their resources in the JMS resource pool. They should have priority when accessing local resources.
Local administrators/privileged users should be able to enforce limits on local resources like:

- *Wall clock time*
- *CPU time (per node and per application)*
- *System time*
- *Memory utilization*
- *Disk usage*
- *Swap space*
- *Dedicated access*
- *Shared access*
- *Network utilization*
- *Disk access*

Each supported resource should have a corresponding hard limit. Jobs exceeding a hard limit should typically be killed, suspended, held, or rejected, but this should be a function of the job manager, and site configurable. Each supported resource should also have a corresponding soft limit. Jobs exceeding a soft limit should be notified and allowed to continue until the hard limit is reached.

[NAS 3.1.5, 4.1.6]

13. Checkpointing

Checkpointing is the process of periodically saving the state of an executing program (job) to a stable storage, from which the system can recover after a failure. Each program state saved is called a **checkpoint**. The disk file that contains the saved state is called the **checkpoint file**. Checkpointing techniques are useful not only for availability (restarting program for fault recovery), but also for program debugging, process migration, and load balancing. Checkpointing can be implemented at different levels as given below:

a. user-level

- *application periodically checkpoints its state without system support.*
- *JMS should have a well-defined interface (API) to facilitate checkpointing and restarting.*

b. run-time library level

A less transparent approach links the user code with a checkpointing library in the user space. Checkpointing and restarting are done by this run-time support. User applications do not have to be modified, but they would have to be relinked with the checkpointing library.

c. OS (kernel) level

*Checkpointing can be realized by the operating system at the **kernel level,** where the OS transparently checkpoints and restarts processes. This is ideal for users. Few systems support checkpointing and fewer support parallel programs checkpointing. Checkpointing is:*

- *Needed for true fault tolerance*
- *Needed for true dynamic resource allocation*

[NAS 3.2.4, 5.1.4]

14. Suspending/resuming/killing jobs

a. user
b. administrator
c. automatic

[NAS 3.4.7, 3.4.10]

15. Process migration

Process migration is the ability to suspend a job or part of a job, and move its full computing environment (binary, local files, etc.) to a different node, or a set of nodes, of the same architecture. Information about the best migration point could be given by the user to simplify the migration process.

a. user
b. administrator
c. automatic

[NAS 5.1.3]

16. Static load balancing

Static load balancing uses algorithms that find adequate placement of all processes before they start execution (at scheduling time) such that resource utilization is leveled.

17. Dynamic load balancing

Dynamic load balancing attempts to equalize the system load dynamically as jobs are created and resumed. Load balancing algorithms allow processes to migrate to remote computers once they have begun execution.

The JMS should provide dynamic load balancing, including the ability to:

- *change resource allocation dynamically*
- *migrate a running application to other nodes*
- *reduce/increase number of nodes allocated to a job as availability/priority changes*
- *reliably migrate jobs*

[NAS 5.1.2]

18. Fault tolerance

Fault tolerance should be provided for both the computational nodes as well as for the scheduler itself as follows:

a. **computational nodes**

JMS must be able to sustain hardware or system failure i.e. no jobs get lost. It should be able to rerun interrupted jobs or use the checkpointing mechanism to rollback these jobs.

b. **scheduler**

Ability to recover from hardware/software failure without losing any important information.

[NAS 3.4.4]

19. Job monitoring

- **real-time monitoring tools including command line interface (CLI) and GUI**
- **history log of all jobs, which includes:**
  - *Time job entered batch system*
  - *Time job entered (each) queue*
  - *Time job started execution*
  - *Time job suspended execution*
  - *Time job restarted execution*
  - *Time job terminated*
  - *Exit status of job*
  - *Total usage and identification of each resource allocated to job*

[NAS 3.2.5]

20. User interface

JMS should provide at least a CLI to all of its modules. A GUI is preferable.

[NAS 3.1.3, 4.1.2]

21. Published APIs

JMS should provide a well-documented API to all of its modules.
[NAS 3.1.4]

22. Dynamic system reconfiguration

Dynamic system reconfiguration refers to the ability to change a configuration of a system during runtime, including ***adding, deleting and modifying resources****, with minimal impact on running jobs. Any jobs, which were running on the removed nodes must, at least, be suspended and started up again once those nodes are turned back over for general use. Ideally, the jobs, which were running on the affected nodes would be checkpointed, moved and continue running on available nodes not affected by reconfiguration.*

[NAS 3.4.6, 3.4.8]

23. Security

This refers to integration with authentication/security subsystem in order to provide:

- *Well documented interface with the security/authentication system*
- *Site configurable authentication mechanism*
- *Necessary hooks for site to interface JMS with local environment. This requires Out-of-the-box support for common and standard authentication systems.*

a. Authentication

Authentication is the process of determining whether someone or something is, in fact, who or what it is declared to be. In private and in public computer networks (including the Internet), authentication is commonly done through the use of logon passwords. The other way of proving user's identity employs asymmetric encryption techniques and digital certificates. Small chunk of information, sent by the other side, is encrypted by the user with the private key (known only to the user) and sent over the network together with public key and its certificate. The other side decrypts encrypted information and compares it with unencrypted part. If both parts match, user is considered as authenticated. Authentication can be performed between any of the following entities: user, scheduler, and execution host.

b. authorization

Authorization is the process of giving someone permission to do or to have something. In multi-user computer systems, a system administrator defines for the system which users are allowed access to the system resources and what privileges they have. Assuming that someone has logged on to a computer operating system or application program, the system or application may want to identify what resources the user can be given during this session. Thus, authorization is sometimes seen as both the preliminary setting up of permissions by a system administrator and the actual checking of the permission values that have been set up when a user is getting access. The most flexible and commonly used means of enforcing access permissions are Access Control Lists (ACL). These are the lists of users and groups who are permitted or denied access to a resource, such as a queue or a certain host. Users and groups may belong to multiple access lists and the same access lists can be used

in various contexts. Authorization used together with a strong authentication prevents against unauthorized use of computational resources.

c. encryption

Encryption is the translation of data into a secret code. Encryption is the most effective way to achieve data security. To read an encrypted file, you must have access to a secret key or password that enables you to decrypt it. Unencrypted data is called plain text; encrypted data is referred to as cipher text. There are two main types of encryption: asymmetric encryption (also called public-key encryption) and symmetric encryption.

A public key cryptosystem uses two keys – a public key known to everyone and a private or secret key known only to the recipient of the message. An important element to the public key system is that the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them. Moreover, it is virtually impossible to deduce the private key if you know the public key.

In symmetric encryption schemes the same key is used to encrypt and decrypt the message. This differs from the asymmetric (or public-key) encryption, which uses one key to encrypt a message and other to decrypt the message. Entities that are most likely to be subjected to the encryption in the JMS are user's data and control information exchanged between all modules.

[NAS 3.1.7]

24. Accounting

Recording of resource usage by users for billing purposes.

ASCII is the preferred format of the files containing accounting information.

GUI interface to control data collection.

[NAS 4.1.9]

25. Scalability

Managing very large clusters (>500 nodes)

Allowing very large parallel jobs (>200 nodes)

Type and scalability of the shared file system if there is one.

[NAS 3.1.10]

MAXIMUM TOTAL SCORE = 100 POINTS

3. SUMMARY OF FINDINGS

3.1 Classifying the Compared Systems

The twelve compared systems can be classified into several major categories, based on their primary objective, major modules, and distribution of control. These groups are defined below, together with the discussion of their relevance to the project goals.

A. Centralized Job Management Systems

The objective of a *centralized JMS* 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,
c. perform centralized job scheduling that matches all available resources with all submitted jobs according to the predefined policies,
d. allocate resources and initiate job execution,
e. monitor all jobs and collect accounting information.

To perform these basic tasks, the centralized JMS must include at least the following major functional units:

1. *User server* – which lets user submit jobs and their requirements to 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. *Central job scheduler* – which performs job scheduling and queuing based on the resource requirements, resource availability, and scheduling policies (task c).
3. *Resource manager*, including
    - *Resource monitor* (tasks a and e), and
    - *Job dispatcher* (task d).

We have classified the following five systems as centralized Job Management Systems:

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

Systems belonging to this group are of primary interest to us from viewpoint of the project objectives. All these systems can be potentially extended to support distributed computations using FPGA accelerator boards.

B. Distributed Job Management Systems without a Central Scheduler

The primary difference compared to the first group is the lack of a *central job scheduler*. The system is able to collect information about all available resources and accept job

submissions, but the decision about matching jobs with available resources is performed locally by a client module or a local job scheduler.

A system of this kind often works as a *high-level job management* system, which can submit jobs to several lower-level centralized job management systems, based on the information about their currently available resources.

This group of systems includes:

- Globus
- Legion
- NetSolve.

Systems belonging to this group are of secondary interest to us from the viewpoint of the direct project goals. However, after extending the capabilities of a centralized JMS, such JMS could be used as a part of a high-level Distributed Job Management System in order to achieve additional scalability.

C. Distributed Operating System

Distributed Operating System differs from the Job Management System by operating on processes instead of jobs, and being oriented primarily towards *tightly coupled* clusters of *homogenous* workstations.

The only member of this category is

- MOSIX.

Since one of the primary requirements of the project is support for *loosely coupled* clusters of *heterogeneous* workstations, MOSIX and other systems of this type should not be considered as potential candidates for extension.

D. Parameter Study Schedulers

Parameter Study Scheduler is a system geared specifically towards the distribution of parameter study applications over networked resources. It is used to schedule jobs belonging to a single application on multiple workstations, and to assign data to these jobs, depending on the current use of resources of these workstations. The application in this case should involve solving a given task for a large number of parameter sets. Each computational node is used to work only on a small subset of a large parameter space. The size of this subset depends on the current computational capabilities of the given node. The instances of the program running on the different workstations do not communicate among each other, but rather return partial results to the central node.

The only representative of this group is

- AppLES

This type of a system is not really a JMS and does not fulfill many major requirements formulated in our project.

E. Resource Monitors and Forecasters

A Resource monitor and forecaster is similar to the resource monitor which is a standard part of each JMS system, as defined in point A. The difference is that the system of this type does not only monitor computational resources, but also predicts their availability and performance in the future. This system uses numerical models to generate forecasts regarding availability of resources in a given time frame.

The only representative of this group is

- Network Weather Service.

The system of this type may be combined with a central JMS to extend its capabilities.

F. Distributed Computing Interface

Distributed Computing Interface is a set of interoperable libraries that can be linked with application programs or operating systems to support client-server applications. Distributed Computing Interface provides

- secure communication between client and server, based on public key certificates;
- directory services, including the means of creating, updating, and accessing directories of currently active servers, public key certificates, etc.,
- distributed file services (e.g., a client access to remote disks), and
- time services (e.g., synchronization of local clocks on several distributed workstations).

3.2 Comparative Overview

The main features of the analyzed systems are presented in Tables I-IV. These features are based on requirements formulated in section II, and are divided into four major categories:

I – System, Flexibility, and Interface
II – Scheduling and Resource Management
III – Efficiency and Utilization
IV – Fault Tolerance and Security.

Analysis of the first set of requirements, “System, Flexibility, and Interface,” leads to the following conclusions (see Table I):

- Majority of systems are available in public domain. The only system that does not have a public domain version is LSF. Codine, PBS, and DCE have both commercial and public domain versions.

- Documentation is excellent for LSF, and good for the majority of the other systems, except RES, AppLES, and NWS. Majority of systems are constantly updated and extended with new capabilities.
- Majority of systems operate under several versions of UNIX, including Linux. The only system that does not directly support Linux is RES. The only system that does not directly support Solaris is MOSIX. Windows NT is supported by only a few systems, including LSF, Legion, NetSolve, DCE, AppLES, and NWS. Windows NT versions are under development for CODINE, Condor, and Globus.
- Impact on the owners of computational nodes is typically small and configurable by the users themselves.
- Majority of systems have both graphical and command-line interface. The only systems without a graphical interface are RES, Legion, and AppLES.

Analysis of the second set of requirements, "Scheduling and Resource Management," leads to the following conclusions (see Table II):

- Batch jobs are supported by all job management systems (both centralized and distributed), except NetSolve. Interactive jobs are not supported by Condor and RES.
- Parellel jobs are fully supported only by LSF, Codine and MOSIX. Limited support is provided by PBS, Globus, and Legion.
- The resource requests by users are supported by majority of job management systems, except Legion and NetSolve. In Globus, a special specification language, RSL, is used to specify the job requirements.
- The most flexible scheduling is provided by LSF and Codine. In PBS and Condor, one of several predefined scheduling policies can be selected by a global administrator.
- All centralized job management systems support job priorities assigned by users.
- All centralized job management systems and Globus support job monitoring.
- Accounting is available in all centralized job management systems, except for RES.

Analysis of the third set of requirements, "Efficiency and Utilization," leads to the following conclusions (see Table III):

- Stage-in and stage-out is supported by LSF, PBS, Globus, and Legion, and to a limited extent by Condor.
- Timesharing of jobs is supported by LSF, Codine, Globus, Legion, and MOSIX. Timesharing of processes is available in PBS and Condor, but is not available in RES.
- Process migration is supported by MOSIX and all centralized job management systems, except RES.
- Dynamic load balancing is supported only by LSF and Codine.
- Scalability is high for LSF, all distributed job management systems, and NWS. Scalability is limited by a central scheduler in CODINE, PBS, Condor, and RES. MOSIX supports only tightly coupled clusters of workstations, and is not scalable to larger clusters connected through internet.

Analysis of the fourth set of requirements, "Fault Tolerance and Security," leads to the following conclusions (see Table IV):

- System level checkpointing is supported only by LSF and CODINE, and for only a small subset of the operating systems. Run-time library checkpointing is supported by LSF, CODINE, PBS, and Condor. User level checkpointing is supported by LSF, CODINE, and Condor.
- Fault tolerance is the best in LSF, CODINE, Condor, NetSolve, MOSIX, and NWS.
- Authentication based on Kerberos is provided in LSF, PBS, Globus, and Legion. Strong authentication based on SSL and X.509 certificates is provided in Condor, Globus, and DCE. Additionally, Globus supports hardware authentication tokens.
- Strong authorization is available in LSF, CODINE, PBS, Condor, Legion, DCE, and Mosix.
- Encryption is clearly documented in Globus, Legion, and DCE, but may also be present in several other systems using Kerberos and SSL.

3.3 Recommendations

The major findings of our study are summarized in Table V. In this table, for each system, we provide its total score, rank among its peers, and major cons and pros.

Among the five centralized Job Management Systems, LSF is the only one that fulfills almost all requirements, and scores the highest in our ranking, with 97% of the maximum number of points. Apart from its unique features that are not present in other systems, LSF also has the largest user base, excellent documentation, customer support, and development plan. The only drawback is the limited access to the source code.

Sun Grid Engine/CODINE is the close second in our ranking. The primary drawback of this system is that it is relatively new and only Unix oriented. At the time of writing, the source code was not yet publicly available, and there was no support for Windows NT. Additionally, only user-level checkpointing is available and based on externally supported libraries. Stage-in and stage-out are not implemented, and instead, all input and output files are accessed through a shared file system, which increases the system overhead.

PBS and Condor constitute the second pair of systems that are worth further consideration. Their ranking and features are very similar. The primary common drawbacks compared to LSF and CODINE are the limited parallel job support, the lack of timesharing support of jobs, the weak checkpointing, and the lack of dynamic load balancing. PBS does not support Windows NT, and the Windows NT version of Condor is incomplete. Additionally, Condor does not support interactive jobs.

These drawbacks do not necessarily eliminate these systems from further consideration. Dynamic load balancing and parallel job supports are relatively difficult to extend to jobs running on FPGA boards. On the other hand, timesharing of jobs might be very useful to

share the basic computational resources between an FPGA-accelerated job and a regular JMS job that wishes to run on the front-end of an FPGA-accelerated node. Checkpointing is very important from the point of view of fault tolerance.

RES shares all major weaknesses with PBS and Condor. Additionally, it has no process migration, very limited security mechanisms, and no accounting capabilities. It also does not support interactive jobs, and its current version runs on neither Windows NT nor Linux.

The primary drawback of systems in the second group, including Globus, Legion, and NetSolve, is the lack of a central scheduler. The absence of a central scheduler is likely to adversely affect the overall computational efficiency of the system. Additionally, the common drawback of these systems is the lack of checkpointing and process migration, as well as accounting capabilities.

MOSIX is a distributed operating system, not a JMS. It supports only tightly coupled clusters of homogenous workstations, which makes it less flexible and not useful for this project.

AppLES is used primarily to assign and deploy data to multiple processes belonging to the same application. It is not a stand-alone job management system, and its use is limited to one specific class of applications, parameter study.

DCE is used to provide services, such as secure communications, directory services, distributed file services, and time services to client-server applications. It is not a job management system.

NWS is devoted to one particular task of a JMS, and can be only used as a part of a larger system.

We recommend including only five systems in the second round of analysis, namely the experimental performance study. These systems are:
LSF, CODINE, PBS, Condor, and Globus.

The first four of these systems - LSF, CODINE, PBS, and Condor - are the centralized Job Management Systems. In the second round of analysis, we are planning to choose one of these four systems to extend to become capable of supporting reconfigurable computing resources. Globus is a distributed job management system. It is selected based on the highest ranking in the category of distributed Job Management Systems, and because of its superior flexibility and interoperability with other JMS systems. Our strategic goal is to investigate the possibility of using Globus to extend the capabilities and size of clusters used for reconfigurable computing.

TABLE I Summary of features of compared systems related to the operating system, flexibility, and user interface.

System, Flexibility, and Interface												
	LSF	**CODINE**	**PBS**	**Condor**	**RES**	**Globus**	**Legion**	**NetSolve**	**MOSIX**	**DCE**	**AppLES**	**NWS**
Version	4.1	5.1	OpenPBS 2.3	6.1.17		1.1.3	1.7	1.3	0.97.10	1.2.2	0.0beta-4	1.1.1.1
Distribution (COM – Commercial, PUB – public domain, RES - restricted)	COM	COM/ PUB	COM/ PUB	PUB	RES	PUB	PUB	PUB	PUB	COM/ PUB	PUB	PUB
Source code	No	No	Yes	Upon request	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Documentation	Excellent	Good	Good	Good	Sufficient	Good	Good	Good	Good	Good	Average	Sufficient
Roadmap	Yes	Yes	Yes	Yes	Limited	Yes	?	Yes	Yes	Yes	Yes	Yes
Linux support	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	?	Yes	Yes
NT support	Yes	Planned	No	Not stable	No	Planned	Yes	Yes	No	Yes	No	No
Solaris support	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
Tru64Unix support	Yes	Yes	Yes	No	No	Yes	Yes	?	No	Yes	Yes	Yes
Other OS support	Unix-like	Unix-like	Unix-like	Unix-like	Unix-like	Unix-like	Unix-like	Unix-like	BSD	Unix-like	Unix-like	Unix-like
Impact on owners of computational nodes	Small	Small	?	Small	Very small	?	Small	Medium	Average	?	Function of the underlying JMS	Small
Dynamic system reconfiguration	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
User interface	CLI & GUI	CLI & GUI	CLI & GUI	CLI & GUI	CLI	CLI & GUI	CLI	CLI & GUI	CLI & GUI	Variable	Poor	CGI web
API	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	

TABLE II Summary of features of compared systems related to scheduling and resource management.

Scheduling and Resource Management												
	LSF	**CODINE**	**PBS**	**Condor**	**RES**	**Globus**	**Legion**	**NetSolve**	**MOSIX**	**DCE**	**AppLES**	**NWS**
Batch jobs	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Processes only	No	Depends on the underlying JMS	N/A
Interactive jobs	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	Depends on the underlying JMS	N/A
Parallel jobs	Yes	Yes	Limited	No	No	Limited	Limited	No	Yes	No	?	N/A
Resource requests	Yes	Yes	Limited	Yes	Yes	RSL	No	No	No	No	No	N/A
Limits on resources	Yes	Yes	Limited	Yes	Limited	Yes	No	No	Limited	No	No	N/A
Flexible scheduling	Yes	Yes	Defined using attributes only	Defined using attributes only	No	Depends on lower-level systems	Local scheduling	No	Fixed in time	N/A	No central scheduling	N/A
Job priorities	Users-assigned and automatic	Yes	Assign-ed by users and adminis-trator	Assign-ed by users and adminis-trator	Assign-ed by users	?	?	No	No	No	No	N/A
Job monitoring	Monitoring tools and logs	Yes	Moni-toring tools and logs	Moni-toring tools and logs	Moni-toring tools	Moni-toring tools and logs	?	No	Yes	Limited	?	N/A
Accounting	Yes	Yes	Interface to ACCT++	Yes	No	No	?	No	No	?	Done by the underlying JMS	N/A

TABLE III Summary of features of compared systems related to efficiency and utilization.

Efficiency and Utilization												
	LSF	**CODINE**	**PBS**	**Condor**	**RES**	**Globus**	**Legion**	**NetSolve**	**MOSIX**	**DCE**	**AppLES**	**NWS**
Stage-in, stage-out	Yes	No	Yes	No transfer of subdirectories	No	Yes	Yes	No	No	No	No	N/A
Timesharing	Jobs	Jobs	Processes	Process	None	Jobs	Jobs	?	Yes	No	No	N/A
Process migration	Yes	Yes	Yes	Yes	No	?	No	No	Yes	N/A	Automatic only	N/A
Static load balancing	Yes	Yes	Yes	Yes	Yes	Yes	?	Yes	Yes	Yes	Yes	N/A
Dynamic load balancing	Yes	Yes	No	No	No	No	No	No	Yes	No	No	N/A
Scalability	High	Medium	Med-ium	Medium	Med-ium	High	High	High	Limited to tightly coupled clusters	High	Depends on the underlying JMS	High

TABLE IV Summary of features of compared systems related to fault-tolerance and security.

Fault-tolerance and Security												
	LSF	**CODINE**	**PBS**	**Condor**	**RES**	**Globus**	**Legion**	**NetSolve**	**MOSIX**	**DCE**	**AppLES**	**NWS**
Checkpointing	User-level, run-time, and limited OS	Using foreign libraries or OS mechanisms	Run-time library	User-level and run-time library	No	No	No	No	No	N/A	No	N/A
Suspending, resuming, killing jobs	Yes	Yes	Yes	Yes	Yes	User only	User only	Yes	Yes	N/A	No	No
Fault-tolerance	Yes	Yes	Limited	Yes	Limited	Limited	Limited	Yes	Yes	Yes	Depends on the underlying JMS	Yes
Authentication	Kerberos	User names and IDs	Kerberos	SSL	Yes	Kerberos + SSL + HW tokens	Kerberos	No	Yes	Based on public-key certificates	No	No
Authorization	Yes	Yes (ACL)	Yes (ACL)	Yes (Different access levels)	No	Yes (User map files)	Yes (ACL)	No	Yes	Yes (ACL)	No	No
Encryption	?	No	?	?	No	DES	Yes	No	No	Yes	No	No

TABLE V Ranking and major pros and cons of the analyzed systems

Centralized JMS				
Name	**Rank**	**Score**	**Cons**	**Pros**
LSF	1	**97**	• Limited access to source code	• Strong support for parallel jobs • Job migration and dynamic load balancing • Strong checkpointing • Minimal impact on owners of computational nodes • GUI to all modules • Good fault tolerance • Strong authentication and authorization
Sun Grid Engine/ CODINE	**2**	**89.5**	• Source code not available at the moment • No Windows NT support • No stage-in / stage-out • Externally supported checkpointing only	• Parallel job support • Job migration and dynamic load balancing • Flexible scheduling • Minimal impact on owners of computational nodes • GUI to all modules • Good authorization mechanism
PBS	**3-4**	**75**	• No Windows NT support • Limited parallel job support • No timesharing of jobs • No checkpointing • No dynamic load balancing	• Flexible scheduling • GUI to all modules • Good authorization mechanism
Condor	**3-4**	**75**	• No interactive job support • Limited parallel job support • No timesharing of jobs • Limited checkpointing • No dynamic load balancing	• Minimal impact on owners of computational nodes • GUI to all modules • Strong authentication and authorization
RES	**8**	**48**	• No Windows NT nor Linux support • No interactive job support • No parallel job support • No stage-in / stage-out • No timesharing of jobs • No checkpointing • No process migration • Limited fault tolerance • No authorization mechanisms • No accounting capabilities	• Minimal impact on owners of computational nodes

Distributed JMS without a Central Scheduler				
Globus	**5**	**58**	• No central scheduling • No Windows NT support • Limited parallel job support • No checkpointing • No process migration • Limited fault tolerance • No accounting capabilities	• High interoperability with other systems • Stage-in and stage-out • GUI to all modules • Strong authentication, authorization, and encryption
Legion	**7**	**51**	• No central scheduling • Limited parallel job support • No checkpointing • No process migration • Limited fault tolerance • No accounting capabilities	• Stage-in and stage-out • Strong authentication and authorization
NetSolve	**9**	**44**	• No central scheduling • No batch job support • No parallel job support • No stage-in / stage-out • No checkpointing • Limited process migration • Limited fault tolerance • No security mechanisms • No accounting capabilities • Limited set of tasks executed by computational nodes	• Job management transparent to users • Interfaces with Matlab and Java
Distributed Operating Systems				
MOSIX	**6**	**57**	• Not a JMS • Supports only Linux and BSD • Supports only tightly coupled cluster of homogenous workstations • Impact on owners of computational nodes • No checkpointing • No accounting capabilities	• Transparent process migration • Parallel job support • Integration with distributed file system • Strong authentication and authorization
Distributed Computing Interface				
DCE	**10**	**35**	• Not a JMS • Need to link with an application	• Strong authentication • Support for a distributed file system • Support for distributed directory services

				• Support for a distributed time service
Parameter Study Scheduler				
AppLES	**11**	**30**	• Not a JMS • Submission only by a single user • No Windows NT support • No security mechanisms • No accounting capabilities • Poor user interface	• Cooperation with other systems • User-defined data scheduling policies
Resource Monitor and Forecaster				
NWS	**12**	**25**	• Not a JMS • No Windows NT support	• Excellent resource usage forecasting • May be used as a part of a JMS

REFERENCES

General References

[BFY95] M. A. Baker, G. C. Fox, and H. W. Yau, "Cluster Computing Review," Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.

[HWA98] Kai Hwang, Zhiwei Xu, *Scalable Parallel Computing:Technology, Architecture, Programming*, McGraw-Hill 1998.

[GOS98] A. Goscinski, *Distributed Operating Systems. The Logical Design*, Addison-Wesley 1991.

[NAS] James Patton Jones, "NAS Requirements Checklist for Job Queuing/Scheduling Software," NAS Technical Report NAS-96-003 April 1996, available at http://www.nas.nasa.gov/Pubs/TechReports/NASreports/NAS-96_003/

[NAS96] James Patton Jones, "Evaluation of Job Queuing/Scheduling Software: Phase 1 Report," NAS Technical Report, NAS-96-009, September 1996 available at http://www.nas.nasa.gov/Research/Reports/Techreports/1996/nas-96-009-abstract.html

References Specific to A Particular System

LSF

http://www.platform.com/index.html
LSF Administrator 's Guide Version 4.1 December 2000
LSF JobScheduler Administrator's Guide
LSF JobScheduler User's Guide
LSF Parallel User 's Guide Version 4.1 December 2000
LSF Programmer 's Guide Version 4.1 December 2000
LSF Reference Guide Version 4.1 December 2000

Sun Grid Engine / Codine

Homepage: http://www.sun.com/software

The Sun Grid Engine manual
http://www.sun.com/software/gridware/docs/gridengine-manual.pdf

Sun Grid Engine Frequent Asked General Questions list
http://www.sun.com/software/gridware/faqs/faqs.html

Sun Grid Engine Frequent Asked technical Questions list
http://supportforum.Sun.COM/gridengine/

Sun Grid Engine Overview http://www.sun.com/software/gridware/ds-gridware/

Sun Grid Engine Detailed View
http://www.sun.com/software/gridware/details.html

PBS

http://pbs.mrj.com/
Portable Batch System OpenPBS Release 2.3 Administrator Guide

Condor

http://www.cs.wisc.edu/condor/
Condor Version 6.1.17 Manual

RES

William W. Carlson, "*RES: A simple system for distributed computing*", Technical Report SRC-TR-92-067, Supercomputing Research Center.

MOSIX

Homepage: http://www.mosix.org/

Barak A. and La'adan O., *"The MOSIX Multicomputer Operating System for High Performance Cluster Computing"*, Journal of Future Generation Computer Systems, Vol. 13, No. 4-5, pp. 361-372, March 1998.
http://www.mosix.cs.huji.ac.il/ftps/mosixhpcc.ps.gz

Barak A., La'adan O. and Shiloh A., "*Scalable Cluster Computing with MOSIX for LINUX*", Proceedings Linux Expo '99,pp. 95-100, Raleigh, N.C., May 1999.
http://www.mosix.cs.huji.ac.il/ftps/mosix4linux.ps.gz

Amar L., Barak A., Eizenberg A. and Shiloh A., "*The MOSIX Scalable Cluster File Systems for LINUX*", July 2000. http://www.mosix.cs.huji.ac.il/ftps/mfs.ps.gz

Slide presentation At Linux EXPO Paris/Linuxworld, Feb. 2, 2001.
http://www.mosix.cs.huji.ac.il/slides/Paris/paris.htm

Mosix reference manual http://www.mosix.cs.huji.ac.il/ftps/mosix.man.html

Globus

http://www.globus.org/

Globus Toolkit 1.1.3 System Administration Guide *December 2000*
Globus Tutorials

Legion

http://www.cs.virginia.edu/~legion/
Legion 1.7 Basic User Manual
Legion 1.7 Developer Manual
Legion 1.7 Reference Manual
Legion 1.7 System Administrator Manual

NetSolve

NetSolve Homepage:
http://www.cs.utk.edu/netsolve/

H. Casanova and J. Dongarra, "Providing Access to High Performance Computing Technologies," Springer-Verlag's Lecture Notes in Computer Science #1184, pp. 123-134.

H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, No. 3, pp. 212-223, Fall 1997.

D. C. Arnold, W. Lee, J. Dongarra, and M. Wheeler, "Providing Infrastructure and Interface to High Performance Applications in a Distributed Setting," *High Performance Computing 2000*.

H. Casanova, J. Plank, M. Beck, and J. Dongarra, "Deploying Fault-Tolerance and Task Migration with NetSolve," to appear in the *International Journal on Future Generation Computer Systems*.

AppLES

Homepage: http://apples.ucsd.edu/

Henri Casanova, Graziano Obertelli, Francine Berman and Rich Wolski: "*The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*", Proceedings of the Super Computing Conference (SC'2000), http://gcl.ucsd.edu/apst/publications/apst_sc00.ps

NWS

Homepage: http://nws.npaci.edu/NWS/

Rich Wolski, Neil T. Spring, and Jim Hayes, The Network Weather Service: "*A Distributed Resource Performance Forecasting Service for Metacomputing*",

Journal of Future Generation Computing Systems, 1999;
http://www.cs.ucsd.edu/users/rich/papers/nws-arch.ps.gz\

Compaq DCE

DCE Product Overview
http://www.transarc.ibm.com/Product/DCE/DCEOverview/dceoverview.html

Michael D. Millikin *"DCE: Building the Distributed Future"*, BYTE June 1994,
http://www.byte.com/art/9406/sec8/art1.htm

DCE Frequently Asked Questions, http://www.opengroup.org/dce/info/faq-mauney.html

Compaq DCE :
http://www.tru64unix.compaq.com/dce/index.html