Effective Utilization and Reconfiguration of Distributed Hardware Resources Using Job Management Systems

Kris Gaj[1], Tarek El-Ghazawi[2], Nikitas Alexandridis[2],
Jacek R. Radzikowski[1], Mohamed Taher[2], and Frederic Vroman[2]

[1] George Mason University,
[2] The George Washington University
kgaj@gmu.edu, tarek@seas.gwu.edu, alexan@seas.gwu.edu,
jradziko@gmu.edu, mtaher@gwu.edu, f.vroman@ifrance.com

Abstract

Reconfigurable hardware resources are very expensive, and yet can be underutilized. This paper describes a middleware capable of discovering underutilized computing nodes with FPGA-based accelerator boards in a networked environment. Using an extended Job management system (JMS), this middleware permits sharing reconfigurable resources at least among the members of the same organization. Traditional resources, such as CPU time of loosely coupled workstations can be shared using a variety of existing Job Management Systems (JMSs). We analyzed four of these systems, LSF, Sun Grid Engine / CODINE, PBS Pro, and Condor from the point of view of their functional characteristics and ease of extension to support reconfigurable hardware. LSF was shown to efficiently address the majority of identified requirements. The general architecture of the extended system was developed, and the exact techniques of extending LSF, CODINE, and PBS Pro to manage FPGA-based accelerator boards were identified. The system architecture was verified experimentally for the specific case of LSF and three types of FPGA accelerator boards. The utilization of FPGA boards was demonstrated to reach up to 86% in our experimental setting consisting of Linux and Windows NT workstations[1].

1. Introduction

This paper reports on a research effort to create a distributed computing system interface for the effective utilization of networked reconfigurable computing resources. The objective is to construct a system that can leverage under-utilized resources at any given time to serve users who currently have the needs, in a grid computing like style. The targeted type of resources are workstations and clusters that are equipped with Field Programmable Arrays (FPGA) boards serving as reconfigurable coprocessors, as one can find in academic and government research labs. In order to take advantage of previous related works, our strategy is to extend an efficient Commercial Off the Shelf (COTS) Job Management System (JMS) [1-5]. Such extensions should provide the ability to recognize reconfigurable resources, monitor and understand their current loading, and effectively schedule them for the incoming remote user requests, with little impact on local users. It also includes providing local users with proper tools to control the degree to which they wish to share their own resources and how others may use such resources.

Our effort started with a study that aimed at the comparative evaluation of currently available job management systems and a conceptual design of how to architect such a system for managing networked reconfigurable resources [5-7]. After selecting one system most suitable for the extension, the detailed architecture of the extended system was developed and experimentally tested.

This paper is organized as follows. In Section 2, we present a general architecture of a JMS and compare functional characteristics of four popular JMSs selected for our study. In Section 3, we determine which JMS features are most important from the point of view of extension to reconfigurable hardware, and present general architecture of the extended system. In Section 4, we describe in detail an extended system based on LSF and three types of the FPGA accelerator boards. We also present an experimental setup used to verify the correct behavior and to measure efficiency of the proposed system. We follow in Section 5 with the description of the system behavior and the analysis of the obtained results.

[1] This work has been partially supported by the Department of Defence under the LUCITE contract no. MDA904-98-CA0810000.

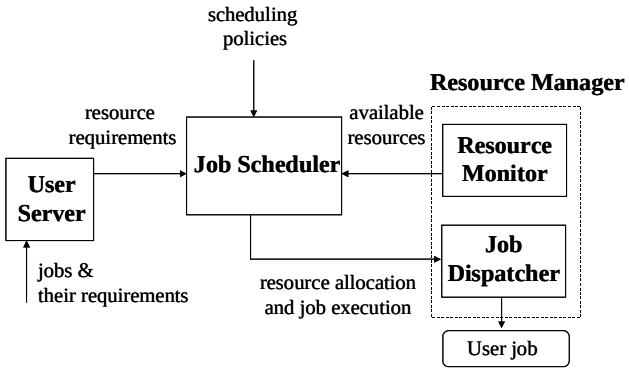

Figure 1. Major functional units of a Job Management System

2. Job Management Systems

2.1. General architecture of a JMS

The objective of a JMS is to let users execute jobs in 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, 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*, including
   - *Resource monitor* – which collects information about all available resources (tasks a and e), and
   - *Job dispatcher* – which allocates resources and initiates execution of jobs submitted to JMS (task d).

2.2. Choice of a Job Management System

More than twenty JMS packages, both commercial and public domain, are currently in use [1, 3, 5]. For the 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 computational node.

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

PBS (Portable Batch System) has both a public domain and a commercial version. 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 comparison of selected Job Management Systems

The main features of selected JMSs are compared and contrasted in Table 1. These features are classified into the following categories:

I – Availability and Operating System Support,

II – Scheduling and Resource Management,

III – Efficiency and Utilization,

IV – Fault Tolerance and Security, and

V – Documentation and Technical Support.

In summary, LSF outperforms all other JMSs in terms of the operating system support, scalability, documentation, and technical support. It is also one of only two systems that fully support parallel jobs, checkpointing, and offer strong resistance against the master host failure.

CODINE performs extremely well in multiple categories such as parallel job support, job migration, load balancing, and resistance against the master host failure.

Table 1. Conceptual functional comparison of selected Job Management Systems

	LSF	CODINE	PBS	Condor
Availability and Operating System Support				
Distribution	commercial	public domain	commercial and public domain	public domain
Source code	no	yes	public domain version only	yes
Solaris, Linux	yes	yes	yes	yes
Tru64	yes	yes	yes	no
Windows NT	yes	no	no	partial
Scheduling and Resource Management				
Interactive jobs	yes	yes	yes	no
Parallel jobs	yes	yes	partial	limited to PVM
Efficiency and Utilization				
Stage-in and stage-out	yes	no	yes	yes
Process migration	yes	yes	no	yes
Dynamic load balancing	yes	yes	no	no
Fault Tolerance and Security				
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
Documentation and Technical Support				
Documentation	excellent	good	good	good
Technical Support	excellent	not tested	good	average

The major drawbacks of CODINE include the lack of support for Windows NT, no support for stage-in and stage-out, and only externally supported checkpointing. The primary weaknesses of PBS 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. The main weaknesses of Condor include no support for interactive jobs, limited support for parallel jobs, and average technical support.

3. Extending a JMS to support reconfigurable hardware

3.1. JMS features supporting extension

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;
- support for Windows NT and Linux, which are two primary operating systems running on PCs that can be extended with commercially available FPGA-based accelerator boards with the PCI interface.

An ease of defining new dynamic resources appears to be a minor factor in comparison. Three out of four systems, LSF, CODINE, and PBS Pro, seem to be easily extendable with new dynamic resources without the need for any changes in their source code. Condor can also be relatively easily extended, taken into account the full access to its source code. Stage-in and stage-out are supported by all systems except CODINE. LSF is the only JMS that fully supports Windows NT. In Condor, jobs submitted from Windows NT can only be executed on machines running Windows NT.

Taking into account the combined results of our study we consider LSF the best candidate for use with the FPGA-based accelerator boards.

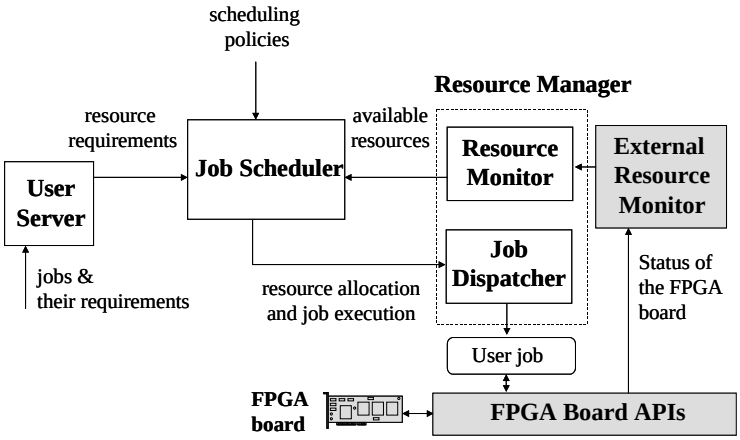

Figure 2. Extension of a JMS to recognize, monitor, and schedule reconfigurable resources

3.2. General architecture of the extended system

General architecture of the extended system is shown in Fig. 2. The primary component of this extension is an external resource monitor that controls the status of an accelerator board, and periodically communicates this status to a resource monitor. The resource monitor transfers this information periodically or by request to a Job scheduler, which uses this information to match each job that requires acceleration with an appropriate host. Job requirements regarding the new reconfigurable resource are specified during a job submission to a user server, and are enforced by a job scheduler the same way as requirements regarding default built-in resources.

3.3. Extending LSF, PBS, and CODINE

Capability of defining new dynamic resources can be used to extend LSF, PBS, and CODINE to manage FPGA-based accelerator boards. The new resource that needs to be added to a given JMS represents the availability of the accelerator board for JMS users.

An external resource monitor needs to 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 daemon is started by a local resource manager (LIM in LSF, cod_execd in CODINE, and MOM in PBS), and communicates with the resource monitor using standard output. Extending Condor to provide the similar functionality would require changes in the source code of this system.

4. LSF Experimental Case Study

4.1. Extending LSF to support reconfigurable hardware

The general architecture of LSF is shown in Fig. 3. Load Information Monitors (LIMs), running on all execution hosts in the system, monitor and collect information about the current status of all static and dynamic resources available on the execution hosts. This information is periodically forwarded from every LIM to a single Master Load Information Monitor (MLIM) residing on the master host. The combined report about the current status of all system resources, collected by MLIM, is used by the Master Batch Daemon (MBD) to match available resources with resource requirements specified during the job submission. When a job waiting in the queue is

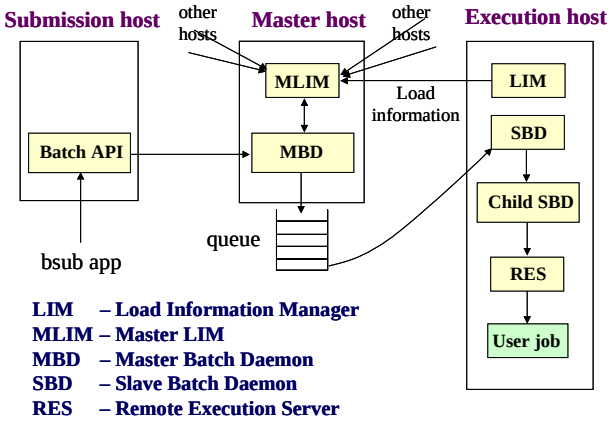

Figure 3. General architecture of LSF

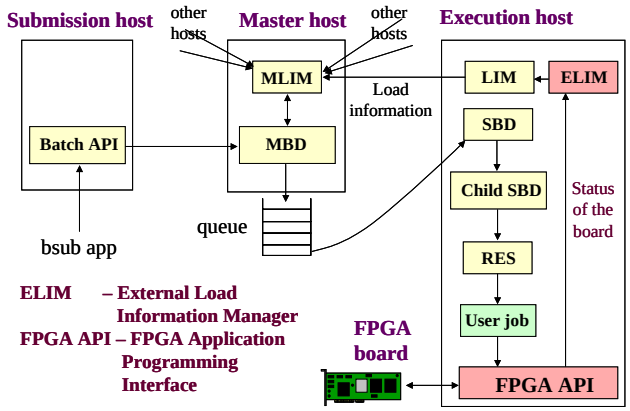

Figure 4. General architecture of LSF after extension to support reconfigurable hardware

matched with an execution host containing the required resources, this job is being dispatched by MBD to the appropriate execution host. The job is prepared for execution by the Slave Batch Daemon (SBD), and started by the Remote Execution Server (RES). SBD is responsible for enforcing local LSF policies and maintaining the status of the job.

To support reconfigurable resources, such as FPGA-based accelerator boards, the LSF system needs to be extended with two extra components: External Load Information Monitor (ELIM) and an FPGA Board Application Programming Interface (API), as shown in Fig. 4. ELIM is a program or script that must be run on each execution host that contains a non-standard dynamic resource, such as an FPGA board. The task of ELIM is to monitor the availability of the FPGA board and to report this availability in the predefined format to LIM. To perform this task, ELIM uses functions of the FPGA Board API. These functions communicate with the FPGA board driver in order to determine whether the board is currently occupied by any job. If this is the case, ELIM reports through LIM to Master LIM (MLIM) that the FPGA board is temporarily unavailable. Otherwise, the information about the availability of the FPGA board is passed to MLIM.

Each user job that makes use of reconfigurable resources needs at the beginning of its execution check the availability of the board. If the board is unavailable, the job exits with an error code, and is resubmitted by LSF at a later time. If the board is available, the job reserves the board for exclusive use, and then configures the board using the configuration bitstream residing on the execution host or downloaded from the submission host using the stage-in capability of LSF. As soon as the board is configured, its clock is started and the FPGA circuit starts communicating with the job running on the execution host. Inputs are sent to the board, and outputs generated by the FPGA circuit are sent back to the job. After the FPGA circuit completes execution, it communicates this fact to

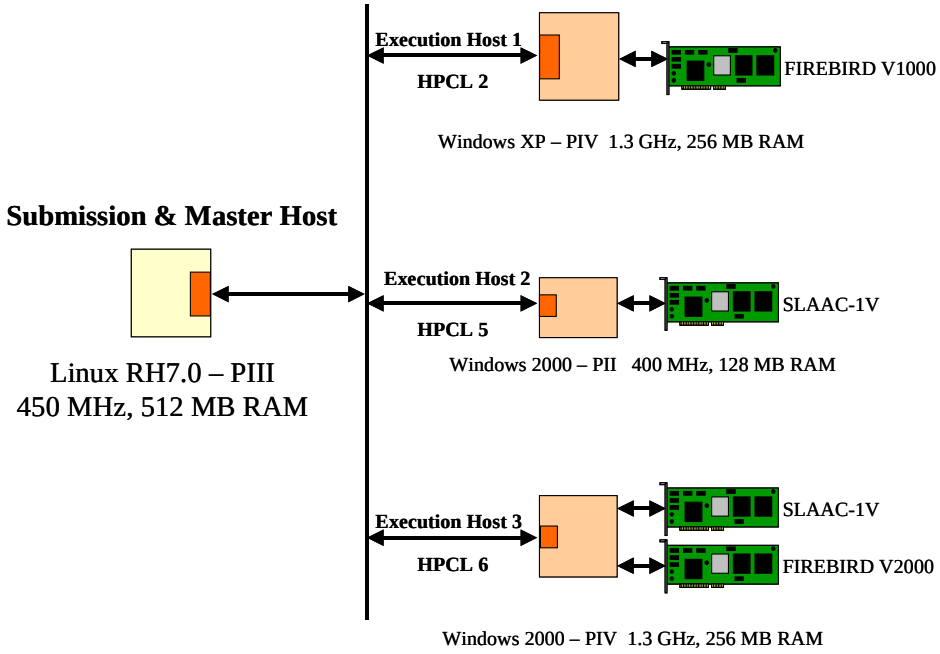

Figure 5. Experimental testbed

the job, which makes final postcomputations, frees the board for use by other jobs, and finishes execution. All described above operations are facilitated by the FPGA board APIs.

4.2 Experimental setup

Our testbed consists of three machines configured as execution hosts, and one machine configured as a submission and master host as shown in Fig. 5. All execution hosts contain one or two FPGA boards, including the SLAAC1-V FPGA accelerator board from the USC-Information Sciences Institute [8, 9], and Firebird V1000, and Firebird V2000 from Annapolis Microsystems, Inc. [10].

The benchmark used in our experiments is a hardware implementation of an exhaustive key search attack against Data Encryption Standard (DES). Exhaustive key search is an attack aimed at breaking a cipher by checking all possible keys one by one. To be able to perform this attack, an opponent must know a short fragment of the message and a corresponding fragment of the ciphertext (encrypted message). By decrypting a fragment of the ciphertext with a given key, and comparing the result with a known fragment of the message, a single key can be verified. By repeating the same operation with all possible key values, one is guaranteed to find the correct key. The number of all possible keys in DES is $2^{56} \approx 7.2 \cdot 10^{16}$. This large number of repetitions calls for parallelization of computations. Additionally, since DES was designed to be efficient in hardware rather than in software, an FPGA based hardware accelerator can speed up the required computations by orders of magnitude compared to the purely software parallel implementation.

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. In our experiments, key range was set to values that guaranteed the execution time of single jobs equal to 120 s.

Our implementation consists of two parts. Hardware part was written in VHDL, and was transformed into the FPGA configuration bitstream using Xilinx tools. Software part is responsible for reserving an FPGA board for an exclusive use, downloading the configuration bitstream to the board, transferring input parameters to the hardware part, collecting results generated by the board, and releasing the board. During the majority of the time, the program is idle and its only function is to wait for a board to complete execution. This way, the only resource of the execution hosts which is fully utilized during the benchmark execution is the time of the FPGA-based accelerator.

Each experiment consisted of running 100 jobs submitted to LSF one at a time in the pseudorandom time intervals. All jobs were submitted from the same Linux machine, and belonged to a single user of the system. The rate of the job submissions was chosen to have a Poisson distribution. The submission rate was relatively high with an average interval between consecutive job submissions equal to 5 seconds.

All jobs were the instances of the exhaustive key search benchmark, and differed only with values of input parameters. All these jobs required acceleration by the FPGA boards. The same Linux machine was used as the submission host and the master host. The primary job requirement specified during the job submission was an availability of the specific type of the FPGA board. The second parameter specified during the job submission was the estimated execution time of the job.

In all experiments, LSF was configured as follows: A maximum number of LSF jobs that can be dispatched to a single CPU was set to one. The scheduling policy was "first come first served". The configuration bitstreams used to reconfigure FPGA boards were transferred to the execution hosts using the stage-in/stage-out capabilities of LSF. The dispatching interval, which determines how often the LSF scheduler attempts to dispatch pending jobs, MBD_SLEEP_TIME, was set to 2 seconds. The FPGA board availability was declared as a new dynamic resource. A value of this resource was separate for each execution host and was updated by ELIM every second.

5. Experimental Results

The behavior and performance of the extended Job Management System is shown in Fig. 6. For each FPGA board, two timing traces are presented. The bottom trace shows timing intervals when jobs dispatched to the given execution host are executed. The numbers above these intervals are the numbers of jobs in the order of their submission. The top trace shows time intervals when ELIM reports to LIM that the FPGA board is free for use by another job. The very bottom trace in Fig. 6 is common for the entire system, and shows points in time when jobs are being submitted to LSF from the submission host.

In all experiments, all jobs are being submitted to JMS shortly after the beginning of the experiment, and as a result spend most of the time waiting in the queue for their turn to execute. At the beginning of every experiment ELIM daemons running on each execution host report to LSF that all FPGA-boards are available for scheduling. As soon as a job is dispatched to the given machine for

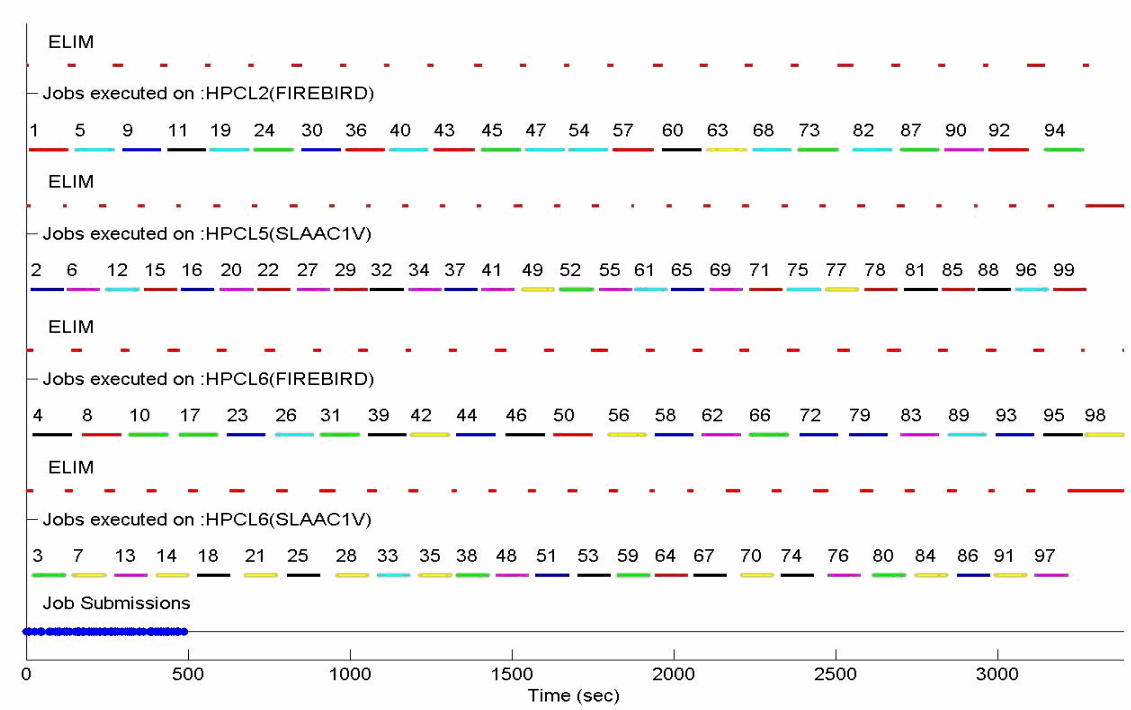

Figure 6. Behavior, performance, and utilization of the extended Job Management System

execution, ELIM running on the same machine becomes aware that the FPGA board is not any longer available. Similarly, as soon as any job completes its execution, ELIM reports to LIM that the board is available for use by another job. We have performed five iterations of the described above experiment, and computed average board utilization during the experiment. This FPGA board utilization varied between 81 and 86% as shown in Table 2.

Table 2. Utilization of the FPGA boards during five iterations of the exhaustive key search experiment

Experiment iteration	Utilization of FPGA boards
1	86%
2	82%
3	82%
4	81%
5	83%

6. Conclusions

Four popular Job Management Systems – LSF, PBS Pro, Sun Grid Engine / CODINE, and Condor – were compared and evaluated with respect to their suitability for being extended to support reconfigurable computing resources and tasks. The general architecture of the extended system was developed. LSF, PBS Pro, Sun Grid Engine / CODINE were shown to be easily extendable without any need for changes in the source code of these systems. An extension of LSF, supporting several popular FPGA accelerator boards was developed and experimentally tested in a testbed consisting of Windows and Linux workstations. Our experiments have proven the correctness of our concept and the feasibility of its implementation using COTS components. The efficiency of the extended system measured in terms of the average utilization of reconfigurable resources appeared to reach 86% for our benchmark based on the exhaustive key search for the DES cipher.

Acknowledgments

The authors would like to acknowledge and thank Pawel Chodowiec and Preeyapong Samipagdi for their contribution to the study described in this paper.

References

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

[2] J. P. 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

[3] K. Hwang, Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill 1998.

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

[5] T. El-Ghazawi, et al., *Conceptual Comparative Study of Job Management Systems*, Technical Report, February 2001, available at http://ece.gmu.edu/lucite/reports.html.

[6] T. El-Ghazawi, et al., *Experimental Comparative Study of Job Management Systems*, Technical Report, July 2001, available at http://ece.gmu.edu/lucite/reports.html.

[7] 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

[8] M. Jones, P. Athanas et al. "Implementing an API for Distributed Adaptive Computing Systems," in *IEEE Workshop on Field-Programmable Custom Computing Machines*, pages 222-230, Napa Valley, CA, April 1999.

[9] B. Schott, S. Crago, et al. "Reconfigurable Architectures for System-Level Applications of Adaptive Computing," In *VLSI Design: Special Issue on Reconfigurable Computing*, pages 265-280, Volume 10, Number 3, 2000.

[10] Annapolis Microsystems, Inc., http://www.annapmicro.com/