High Availability for Ultra-Scale High-End Scientific Computing *

Christian Engelmann
Computer Science and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN 37831-6164, USA
Phone: +1 865 574 3132, Fax: +1 865 576 5491
Department of Computer Science
The University of Reading
P.O. Box 217, Reading, RG6 6AH, UK
engelmannc@ornl.gov

Stephen L. Scott
Computer Science and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN 37831-6367, USA
Phone: +1 865 574 3144, Fax: +1 865 576 5491
scottsl@ornl.gov

ABSTRACT

Ultra-scale architectures for scientific high-end computing with tens to hundreds of thousands of processors, such as the IBM Blue Gene/L and the Cray X1, suffer from availability deficiencies, which impact the efficiency of running computational jobs by forcing frequent checkpointing of applications. Most systems are unable to handle runtime system configuration changes caused by failures and require a complete restart of essential system services, such as the job scheduler or MPI, or even of the entire machine. In this paper, we present a flexible, pluggable and component-based high availability framework that expands today's effort in high availability computing of keeping a single server alive to include all machines cooperating in a high-end scientific computing environment, while allowing adaptation to system properties and application needs.

Keywords

Scientific computing, high availability, virtual synchrony, distributed control, group communication

1. MOTIVATION

A major concern in efficiently exploiting ultra-scale architectures for scientific high-end computing (HEC) with tens to hundreds of thousands of processors, such as the IBM Blue Gene/L [2, 4, 27], the Cray X1 [39] or the Cray XT3 [41], is the potential inability to identify problems and take preemptive action before a failure impacts a running job. In fact, in systems of this scale, predictions estimate the mean time to interrupt (MTTI) in terms of hours. Timely and current propagation and interpretation of important events will improve system management through tuning, scheduling, resource usage, and self-adaptation for unexpected events, such as node failures.

Current solutions for fault-tolerance in HEC focus on dealing with the result of a failure. However, most are unable to handle runtime system configuration changes caused by failures and require a complete restart of essential system services, such as the job scheduler or MPI, or even of the entire machine. High availability (HA) computing strives to avoid the problems of unexpected failures through preemptive measures. Today's effort in HA computing is mostly directed toward keeping a single server alive. This effort needs to be expanded to include all machines cooperating in a HEC environment.

There are various techniques [37] to implement high availability for computing services. They include *active/hot-standby* and *active/active*. *Active/hot-standby high availability* follows the fail-over model. Process state is saved regularly to some shared stable storage. Upon failure, a new process is restarted or an idle process takes over with the most recent or even current state. This implies a short interruption of service for the time of the fail-over and may involve a rollback to an old backup.

Asymmetric active/active high availability further improves the reliability, availability and serviceability (RAS) properties of a system. In this model, two or more processes offer the same service without coordination, while an idle process is ready to take over in case of a failure. This technique allows continuous availability with improved performance. However, it has limited use cases due to the missing coordination between participating processes.

Symmetric active/active high availability offers a continuous service provided by two or more processes that run the same service and maintain a common global process state using *distributed control* [18] or *extended virtual synchrony* [30]. The symmetric active/active model is superior in many areas including throughput, availability, and responsiveness, but is significantly more complex.

*Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

While there are major architectural differences between individual HEC systems, like vector machines vs. massively parallel processing systems vs. Beowulf clusters, they all have similar high availability deficiencies, i.e. *single points of failure (SPoF)* and *single points of control (SPoC)*.

A failure at a SPoF impacts the complete system and usually requires a full or partial system restart. A failure at a SPoC additionally renders the system useless until the failure is fixed. A recovery from a failure at a SPoC always involves repair or replacement of the failed component, i.e. human intervention. Compute nodes are typical single points of failure. Head and service nodes are typical single points of failure and control.

The overall goal of our research is to expand today's effort in HA for HEC, so that systems that have the ability to hot-swap hardware components can be kept alive by an OS runtime environment that understands the concept of dynamic system configuration. With the aim of addressing the future challenges of high availability in ultra-scale HEC, our project intends to develop a proof-of-concept implementation of an active/active high availability system software framework that removes typical single points of failure and single points of control.

For example, cluster head nodes may be added or removed by the system administrator at any time. Services, such as scheduler and load balancer, automatically adjust to system configuration changes. Head nodes may be removed automatically upon failure or even preemptively when a system health monitor registers unusual readings for hard drive or CPU temperatures. Multiple highly available head nodes will also ensure access to compute nodes. As long as one head node survives, computational jobs can continue to run without interruption.

2. RELATED WORK

Related past and ongoing research can be separated into two paths: solutions for fault-tolerance and for high availability. Conceptually, fault tolerant computing enables recovery from failures with an accepted loss of recent computation activity and an accepted interruption of service. In contrast, high availability computing provides an instant failure recovery with little or no loss and with little or no interruption. The grade of availability is defined by the overall downtime, i.e. the interruption (outage) time plus the time it takes to catch-up to the state when the failure occurred. Additional overhead during normal system operation, such as for checkpointing and message logging, also exists and needs to be counted as downtime. There are no fixed boundaries between both research paths.

Research in the area of fault tolerant scientific computing includes fault tolerant message passing layers, such as PVM [23, 36] and FT-MPI [19, 20, 21], message logging systems and checkpoint/restart facilities, like MPICH-V [31] and BLCR [29]. Advanced technologies in this area deal with diskless checkpointing [10, 15, 35] and fault tolerant scientific algorithms [7, 17].

High availability computing has its roots in military, financial, medical and business computing services, where real-time computing resources, such as air traffic control and stock exchange, or data bases, like patient or employee records, need to be protected from catastrophic failures. Research in this area includes process state and data base replication with various real-time capabilities.

Active/hot-standby high availability is the de facto standard for business and telecommunication services, such as Web-servers. Solutions for Beowulf-type systems are also available for high-end scientific computing. Examples are HA-OSCAR [25, 26] and Carrier Grade Linux [9].

Furthermore, recent solutions for ultra-scale architectures include the Cray RAS and Management System (CRMS [41]) of the Cray XT3. The CRMS integrates hardware and software components to provide system monitoring, fault identification, and recovery. By using the PBSPro [33, 34] job management system, the CRMS is capable of providing a seamless failover procedure without interrupting the currently running job. Redundancy is built in for critical components and single points of failure are minimized. For example, the system could lose an I/O PE, without losing the job that was using it.

Asymmetric active/active high availability is being used for stateless services, such as read/search-only database server farms. Ongoing research in this area for scientific high-end computing focuses on a multiple head node solution for high throughput Linux clusters.

Research in symmetric active/active high availability concentrates on distributed control and extended virtual synchrony algorithms based on process group communication [1, 11, 12]. Many group communication algorithms and several software frameworks, such as Ensemble [6], Transis [13], Xpand [40], Coyote [5] and Spread [38] have been developed, but most target a specific network technology/protocol (e.g. UDP) and/or group communication algorithm (e.g. Totem [3]). Only very few allow easy modification of the group communication algorithm itself via micro-protocol layers or micro-protocol state-machines.

3. TECHNICAL APPROACH

In order to provide active/active as well as active/hot-standby high availability for ultra-scale high-end scientific computing, we are in the process of developing a flexible, modular, pluggable high availability component framework that allows adaptation to system properties, like network technology and system scale; and application needs, such as programming model and consistency requirements.

Our high availability framework (Figure 1) consists of four major layers; 1-to-1 and 1-to-n communication drivers, a group (n-to-n) communication system, virtual synchrony interfaces and applications.

At the lowest layer, communication drivers provide single-cast and multicast messaging capability. They may also provide messaging related failure detection. The group communication layer offers group membership management, external failure detectors, reliable multicast mechanisms and atomic multicast algorithms. The virtual synchrony layer builds a bridge between the group communication system

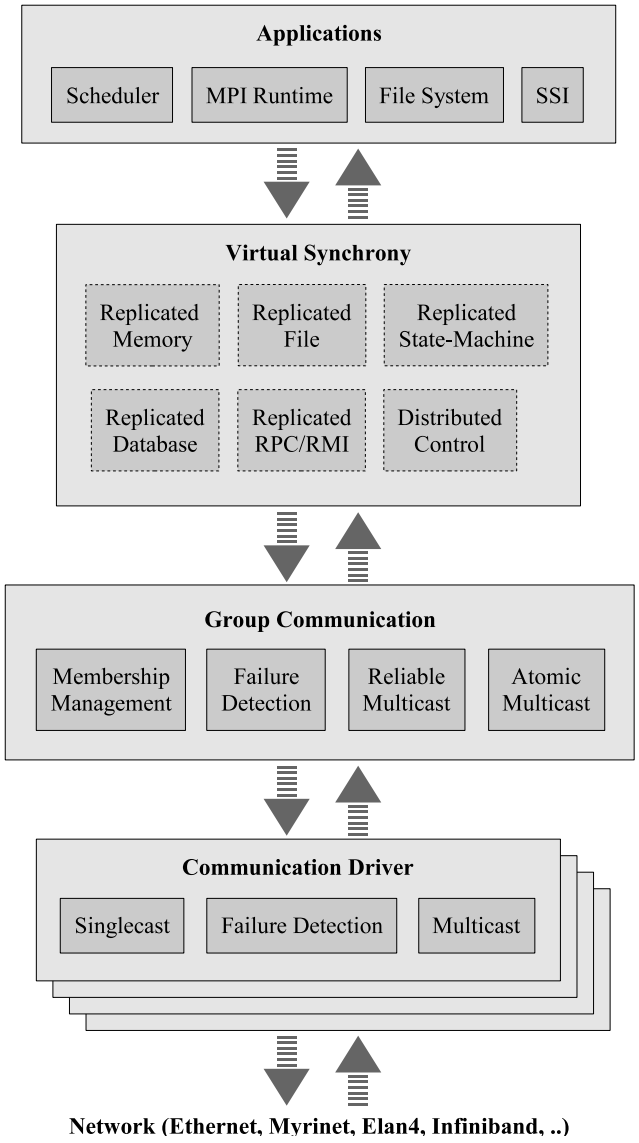

Figure 1: High Availability Framework

and applications using easy-to-use interfaces common to application programmers.

Our high availability framework itself is component-based, .i.e. individual modules within each layer may be replaced with other modules providing different properties for the same service. Our framework allows exchanging software modules using plug-in technology previously developed in the Harness project [14, 16, 24].

In the following sections, we describe each of the four high availability framework layers in more detail.

3.1 Communication Drivers

Today's high-end computing systems come with a variety of different network technologies, such as Myrinet, Elan4, Infiniband and Ethernet. Our high availability framework is capable of supporting vendor supplied network technologies as well as established standards, such as TCP and UDP, using communication drivers, thus enabling efficient communication between participating processes running in virtual synchrony within an application.

The concept of using communication drivers to adapt specific APIs of different network technologies to a unified communication API in order to make them interchangeable and interoperable is not new. For example, Open MPI [22, 32] uses a component-based framework and encapsulates communication drivers using interchangeable and interoperable components.

We are currently investigating if our high availability framework is able to profit from Open MPI communication driver technology by using the Open MPI framework. This also provides an opportunity for Open MPI to benefit from our high availability framework using active/active high availability for essential Open MPI services.

Furthermore, we are also going to consider heterogeneity aspects, such as byte ordering and high-level protocols. For the moment, communication drivers offer an interface that deals with raw data packets only. The use of high-level protocols is managed in the group communication layer. Future work in this area will also reuse recent research in adaptive, heterogeneous and reconfigurable communication frameworks, such as RMIX [28].

3.2 Group Communication Layer

The group communication layer contains all essential protocols and services to run virtual synchronous processes for active/active high availability. It also offers necessary services for active/hot-standby high availability with multiple standby processes using coherent replication. The group communication layer provides group membership management, external failure detectors, reliable multicast mechanisms and atomic multicast algorithms.

Many (60+) group communication algorithms/systems can be found in literature. Our pluggable component-based high availability framework provides an experimental platform for comparing existing solutions and for developing new ones. Implementations with various replication and control strategies using a common API allow adaptation to system properties and application needs. The modular architecture also enables other researchers to contribute their solutions based on the common API.

We will also incorporate previous research in adaptive group communication frameworks using micro-protocol layers and micro-protocol state-machines.

3.3 Virtual Synchrony Layer

The supported APIs at the virtual synchrony layer are based on application properties. Deterministic and fully symmetrically replicated applications may use replication interfaces for memory, files, state-machines and databases. Nondeterministic or asymmetrically replicated applications may use replication interfaces for distributed control and replicated remote procedure calls.

These application properties are entirely based on the group

communication systems point of view and its limited knowledge about the application.

For example, a batch job scheduler that runs on multiple head nodes in a Beowulf-type cluster maintains a global application state among the participating processes. Each scheduler process has the same application state and receives state changes in the same (total) order. A job scheduling request sent to one of these processes results in a state change that schedules the job on all processes. Multiple requests are ordered and their state changes are executed in the same order. The job scheduler‘s behavior is obviously deterministic from the group communication system point of view and the state-machine interface can be used.

However, only one participating scheduler process is actively starting jobs in our example, while the others only accept job scheduling requests from users. Conceptually, the job start part of the entire job management system is organized in a fail-over chain, while the job scheduling part is provided in a symmetric fashion by all head nodes.

A job is started using a replicated RPC that goes out to all participating processes, but executes the job start only on one node. The other processes wait for the replicated RPC return in order to decide whether the job start was successful or not. The job start and its result are based on locality as the application is not operating entirely symmetric.

In case of failure, the surviving job scheduler processes continue to provide their service and a new leader process may be elected to start future jobs. However, a failure of the leader process during a job launch may require clean-up procedures on compute nodes depending on the distributed process model (e.g. bproc [8]) used. Also, currently running jobs need to be associated with the new leader process.

The job scheduler‘s mode of operation may be optimized by load balancing job starts among the participating processes. This may significantly increase performance if the job scheduler is capable to start individual computational processes directly.

As we can see from this simple batch job scheduler example, the active/active high availability model is inherently complex. An API that allows an application to be viewed as a state-machine or that provides a replicated RPC facility significantly improves usability.

3.4 Applications

There are many, very different, applications for our high availability framework. Typical single points of failure and control involve head and service nodes. We previously described the active/active job scheduler example. Other applications include: essential fault-tolerant MPI services (e.g. name server), parallel file system services (e.g. metadata server) and services collecting data from compute nodes (e.g. system monitoring).

Another application area is more deeply involved with the OS kernel itself. For example, single system image (SSI) solutions run one virtual OS on many processors in SMP mode. Memory page replication is needed for a highly available SSI. A similar challenge poses global addressable memory, like in the Cray X1 system. Both applications target head and service nodes as well as compute nodes.

Applications on compute nodes include: super-scalable diskless checkpointing, localized MPI recovery and coherent caching of checkpoints on multiple service nodes. High availability on compute nodes will become more important in the future due to the growing size of high-end computing systems. An example is the recently deployed IBM Blue Gene/L with its 130,000 processors on compute nodes and several thousand processors on service nodes. The mean time to failure is going to outrun the mean time to recover without high availability capabilities.

Our high availability framework will be implemented using a set of shared and static libraries. Depending on the application area it may be used within the application by direct linking, via a daemon process by network access, or via an OS interface, such as /sys or /dev.

4. PROTOTYPE IMPLEMENTATION

We are currently in the process of implementing an initial prototype using the lightweight Harness kernel [16] as a flexible and pluggable backbone (Figure 2) for the described software components.

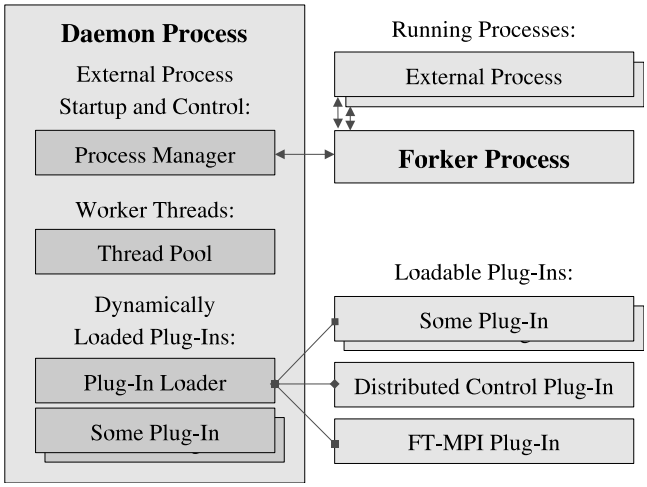

Figure 2: Pluggable Lightweight Harness Kernel

Conceptually, the Harness software architecture consists of two major parts: a runtime environment (kernel) and a set of plug-in software modules. The multi-threaded user-space kernel daemon manages the set of dynamically loadable plug-ins. While the kernel provides only basic functions, plug-ins may provide a wide variety of services.

In fact, our previous research in Harness already targeted a group communication system to manage a symmetrically distributed virtual machine environment (DVM) using distributed control as a form of RPC-based virtual synchrony. The Harness distributed control plug-in provides virtual synchrony services to the Harness DVM plug-in, which maintains a symmetrically replicated global state database for high availability. The accomplishments and limitations of Harness and other group communication middleware projects

were the basis for the flexible, pluggable and component-based high availability framework.

Furthermore, we are also currently investigating different group communication protocols for large-scale applications, such as SSI on compute nodes. Since the virtual synchrony model is based on an all-to-all broadcast problem, scalability is an issue that needs to be addressed. The performance impact for small-scale applications, such as a scheduler running on multiple head nodes, is negligible, but large-scale and/or distributed process groups need to deal with latency and bandwidth limitations.

5. CONCLUSIONS

We presented a flexible, pluggable and component-based high availability framework that expands today's effort in high availability computing of keeping a single server alive to include all machines cooperating in a high-end scientific computing environment. The framework mainly targets symmetric active/active high availability, but also supports other high availability techniques.

Our high availability framework is a proof-of-concept implementation that aims to remove typical single points of failure and single points of control from high-end computing systems, such as single head and service nodes, while adapting to system properties and application needs. It uses pluggable communication drivers to allow seamless adaptation to different vendor supplied network technologies as well as established standards. Its pluggable component-based group communication layer provides an experimental platform for comparing existing group communication solutions and for developing new ones. Furthermore, its virtual synchrony layer provides adaptation to different application properties, such as asymmetric behavior.

We target applications that usually provide services on single head and service nodes, such as schedulers and essential message passing layer services. We also target compute node applications, such as the message passing layer itself and super-scalable high availability technologies for 100,000 processors and beyond.

6. REFERENCES

[1] Special issue on group communications systems. *Communications of the ACM*, 39(4), 1996.

[2] N. R. Adiga et al. An overview of the Blue Gene/L supercomputer. *Proceedings of SC, also IBM research report RC22570 (W0209-033)*, 2002.

[3] D. Agarwal. Totem: A reliable ordered delivery protocol for interconnected local-area networks. *PhD Thesis, University of CA, Santa Barbara*, 1994.

[4] ASCII Blue Gene/L Computing Platform at Lawrence Livermore National Laboratory, Livermore, CA, USA. http://www.llnl.gov/asci/platforms/bluegenel.

[5] N. T. Bhatti, M. A. Hiltunen, R. D. Schlichting, and W. Chiu. Coyote: a system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, 1998.

[6] K. Birman, B. Constable, M. Hayden, J. Hickey, C. Kreitz, R. van Renesse, O. Rodeh, and W. Vogels. The Horus and Ensemble projects: accomplishments and limitations. *Proceedings of DISCEX*, 1:149–161, 2000.

[7] G. Bosilca, Z. Chen, J. Dongarra, and J. Langou. Recovery patterns for iterative methods in a parallel unstable environment. *Submitted to SIAM Journal on Scientific Computing*, 2005.

[8] BProc: Beowulf Distributed Process Space at Sourceforge.net. http://bproc.sourceforge.net.

[9] Carrier Grade Linux Project at Open Source Development Labs (OSDL), Beaverton, OR, USA. http://www.osdl.org/lab_activities/carrier_grade_linux.

[10] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra. Building fault survivable MPI programs with FTMPI using diskless checkpointing. *Submitted to PPoPP*, 2005.

[11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 33(4):1–43, 2001.

[12] Xavier Defago, Andre Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[13] Danny Dolev and Dalia Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, 39(4):64–70, 1996.

[14] W. R. Elwasif, D. E. Bernholdt, J. A. Kohl, and G. A. Geist. An architecture for a multi-threaded Harness kernel. *Lecture Notes in Computer Science: Proceedings of PVM/MPI User's Group Meeting*, 2131:126–134, 2001.

[15] C. Engelmann and G. A. Geist. A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. *Proceedings of CLADE*, pages 47–52, 2003.

[16] C. Engelmann and G. A. Geist. A lightweight kernel for the harness metacomputing framework. *Proceedings of HCW*, 2005.

[17] C. Engelmann and G. A. Geist. Super-scalable algorithms for computing on 100,000 processors. *Proceedings of ICCS*, 2005.

[18] C. Engelmann, S. L. Scott, and G. A. Geist. High availability through distributed control. *Proceedings of HAPCW*, 2004.

[19] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. Harness and fault tolerant MPI. *Parallel Computing*, 27(11):1479–1495, 2001.

[20] G. E. Fagg, A. Bukovsky, S. Vadhiyar, and J. J. Dongarra. Fault-tolerant MPI for the Harness metacomputing system. *Lecture Notes in Computer Science: Proceedings of ICCS 2001*, 2073:355–366, 2001.

[21] FT-MPI Project at University of Tennessee, Knoxville, TN, USA. At http://icl.cs.utk.edu/ftmpi.

[22] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. *Proceedings of 11th European PVM/MPI Users' Group Meeting*, 2004.

[23] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, USA, 1994.

[24] G.A. Geist, J.A. Kohl, S.L. Scott, and P.M. Papadopoulos. HARNESS: Adaptable virtual machine environment for heterogeneous clusters. *Parallel Processing Letters*, 9(2):253–273, 1999.

[25] HA–OSCAR at Louisiana Tech University, Ruston, LA, USA. At http://xcr.cenit.latech.edu/ha-oscar.

[26] I. Haddad, C. Leangsuksun, and S. Scott. HA-OSCAR: Towards highly available linux clusters. *Linux World Magazine*, March 2004.

[27] IBM Blue Gene/L Computing Platform at IBM Research. http://www.research.ibm.com/bluegene.

[28] D. Kurzyniec, T. Wrzosek, V. Sunderam, and A. Slominski. RMIX: A multiprotocol RMI framework for Java. *Proceedings of IPDPS*, pages 140–145, 2003.

[29] Lawrence Berkeley National Laboratory, Berkeley, CA, USA. BLCR Project at http://ftg.lbl.gov/checkpoint.

[30] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. *Proceedings of DCS*, pages 56–65, 1994.

[31] MPICH-V Project at University of Paris South, France. http://www.lri.fr/∼gk/mpich-v.

[32] Open MPI Project. http://www.open-mpi.org.

[33] PBSPro Job Management System at Altair Engineering, Inc., Troy, MI, USA. http://www.altair.com/software/pbspro.htm.

[34] PBSPro Job Management System for the Cray XT3 at Altair Engineering, Inc., Troy, MI, USA. http://www.altair.com/pdf/PBSPro_Cray.pdf.

[35] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, 1998.

[36] PVM Project at Oak Ridge National Laboratory. Oak Ridge, TN, USA. http://www.csm.ornl.gov/pvm.

[37] Ron I. Resnick. A modern taxonomy of high availability. 1996.

[38] The Spread Toolkit and Secure Spread Project at Johns Hopkins University, Baltimore, MD, USA. http://www.cnds.jhu.edu/research/group/secure_spread/.

[39] X1 Computing Platform at Cray Inc., Seattle, WA, USA. http://www.cray.com/products/x1.

[40] Xpand Project at Hebrew University of Jerusalem. Israel. http://www.cs.huji.ac.il/labs/danss/xpand.

[41] XT3 Computing Platform at Cray Inc., Seattle, WA, USA. http://www.cray.com/products/xt3.