Active/Active Replication for Highly Available HPC System Services*†

C. Engelmann[1,2], S. L. Scott[1], C. Leangsuksun[3], X. He[4]
[1] *Computer Science and Mathematics Division*
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[2] *Department of Computer Science*
The University of Reading, Reading, RG6 6AH, UK
[3] *Computer Science Department*
Louisiana Tech University, Ruston, LA 71272, USA
[4] *Department of Electrical and Computer Engineering*
Tennessee Technological University, Cookeville, TN 38505, USA
engelmannc@ornl.gov, scottsl@ornl.gov, box@latech.edu, hexb@tntech.edu

Abstract

Today's high performance computing systems have several reliability deficiencies resulting in availability and serviceability issues. Head and service nodes represent a single point of failure and control for an entire system as they render it inaccessible and unmanageable in case of a failure until repair, causing a significant downtime. This paper introduces two distinct replication methods (internal and external) for providing symmetric active/active high availability for multiple head and service nodes running in virtual synchrony. It presents a comparison of both methods in terms of expected correctness, ease-of-use and performance based on early results from ongoing work in providing symmetric active/active high availability for two HPC system services (TORQUE and PVFS metadata server). It continues with a short description of a distributed mutual exclusion algorithm and a brief statement regarding the handling of Byzantine failures. This paper concludes with an overview of past and ongoing work, and a short summary of the presented research.

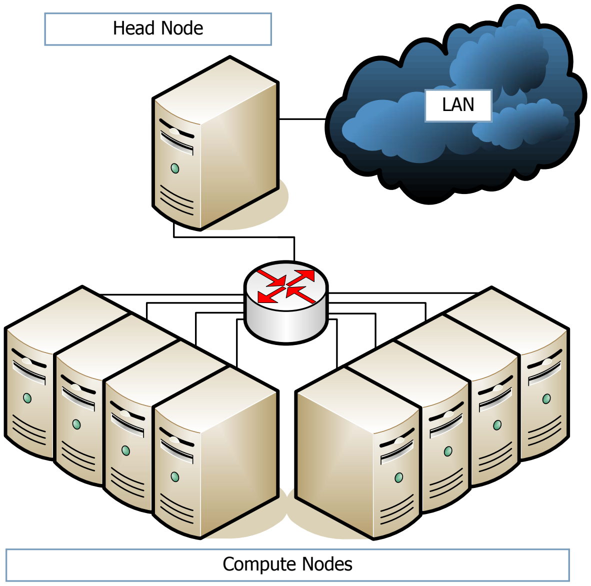

Figure 1. Beowulf Cluster Architecture

1. Introduction

High performance computing (HPC) exploits multi-processor parallelism on a large scale in order to enable research in computational sciences in various areas, such as nanotechnology, quantum chemistry, nuclear fusion and astrophysics. Simulations of real-world problems using mathematical abstraction models allow scientists to gain knowledge without the need or the capability to perform physical experiments.

A HPC system typically consists of several nodes

*Research sponsored in part 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.

†This research is sponsored in part by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed jointly at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725, The University of Reading, Louisiana Tech University and Tennessee Tech University.

(Figure 1), where each node has at least one processor, some memory and at least one network interface. While a significant number of compute nodes perform the actual parallel scientific simulation, a single head node and optional service nodes handle system management tasks, such as user login, resource management, job scheduling, data storage and I/O.

This system architecture has been proven to be very efficient as it permits customization of nodes and interconnects to their purpose. However, it also implies several reliability deficiencies resulting in system-wide availability and serviceability issues [8].

Due to the fact that an entire HPC system depends on each of its nodes and on the network to function properly, single node or link interrupts trigger system-wide disruptions. Furthermore, single head or service node failures may lead to system-wide outages.

While each compute node typically represents a *single point of failure* interrupting the entire system upon failure, fault tolerance mechanisms, such as checkpoint/restart [2] and message logging/replay [16], allow continued operation without failed component(s) after reconfiguration with limited (progress since the last checkpoint) or no loss of application state.

However, head and service nodes are additionally a *single point of control* for the entire HPC system as they render it inaccessible and unmanageable in case of a failure until repair, causing a significant downtime (up to several percent in some instances).

Several models exist to perform reliable and consistent replication of service state to multiple redundant nodes for high availability. Past research focused on the *active/standby* model [12, 19, 23] (Figure 2), where each head and/or service node has at least one redundant idle backup node, waiting to failover upon failure of its primary node. However, interruption of service and loss of service state may occur during a failover depending on the used replication technique (hot-, warm- or cold-standby). Furthermore, additional idle backup nodes are needed in order to be able to tolerate multiple simultaneous or subsequent failures.

The research presented in this paper targets the *symmetric active/active* replication model for high availability [8] using multiple redundant active head and service nodes running in *virtual synchrony*. In this model, head or service node failures do not cause a failover to a backup and there is no disruption of service or loss of service state. Furthermore, the number of active head and service nodes is variable at runtime allowing adaptation, such as adding new nodes when old nodes become less reliable, and reconfiguration, for example live upgrades of services.

This paper first discusses the virtual synchrony paradigm and its application to the active/active replication model. It continues with an introduction of two distinct replication methods (internal and external) for providing active/active high availability for HPC system services. We further present a comparison of both methods in terms of expected correctness, ease-of-use and performance based on early results from ongoing work. We continue with a short description of a distributed mutual exclusion algorithm, which ensures that output produced by multiple redundant active services is delivered only once, and a brief statement regarding the handling of Byzantine failures. We conclude with a brief overview of past and ongoing research in this area, and a short summary of the presented work.

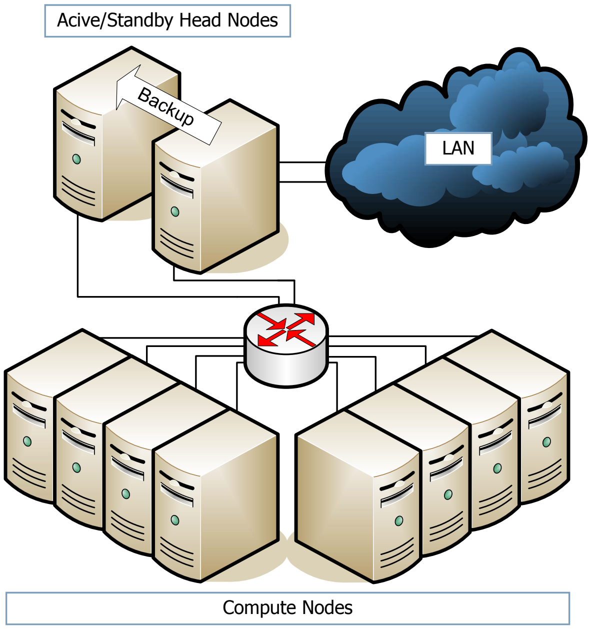

Figure 2. Enhanced Beowulf Cluster Architecture with Active/Standby High Availability for Head Node System Services

2. Virtual Synchrony

The *virtual synchrony paradigm* was first established in the early work on the ISIS [1] group communication system. It defines the relation between regular-message passing in a process group and control-message passing provided by the system itself (e.g. reports on process joins or failures).

Process group membership is dynamic, i.e. processes may join and leave the group. Whenever group membership changes, all the processes in the new membership observe a membership change event.

Conceptually, the virtual synchrony paradigm

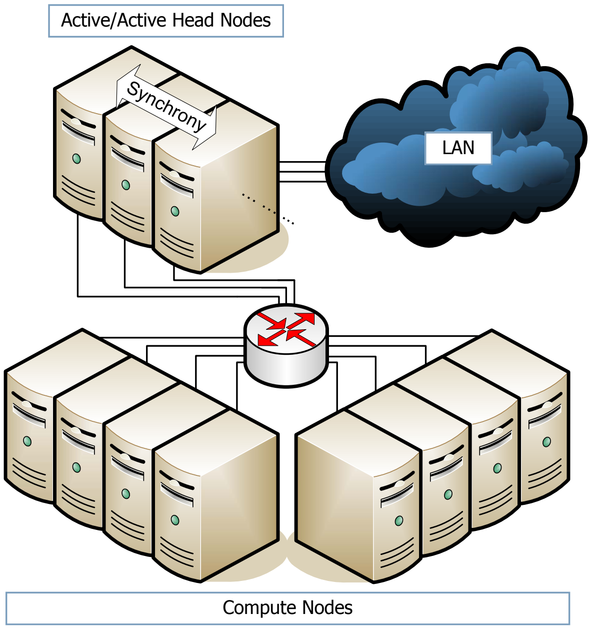

Figure 3. Advanced Beowulf Cluster Architecture with Symmetric Active/Active High Availability for Head Node System Services

guarantees that membership changes within a process group are observed in the same order by all the group members that remain connected. Moreover, membership changes are totally ordered with respect to all regular messages that pass in the system.

The *extended virtual synchrony* paradigm [15], which additionally supports crash recoveries and network partitions, has been implemented in the Transis [7] group communication system.

In the context of high availability for HPC system services, the virtual synchrony paradigm allows replication of a system service to multiple head/service nodes (Figure 3) by providing reliable total order message delivery to all members of a process group.

Service state changes are events, delivered in the form of totally ordered message muilticasts. All members of a group perform the same state change, since they receive all messages in the same order. New members obtain the current service state from the process group. The service runs in virtual synchrony, i.e. all members have a consistent replica and the service is provided as long as one group member is still alive.

3. Active/Active High Availability

High availability is based on redundancy [21]. If a service fails, the system is able to continue to operate using a redundant one. As a result, its mean time to recover can be decreased, loss of state can be reduced and single points of failure and control can be eliminated.

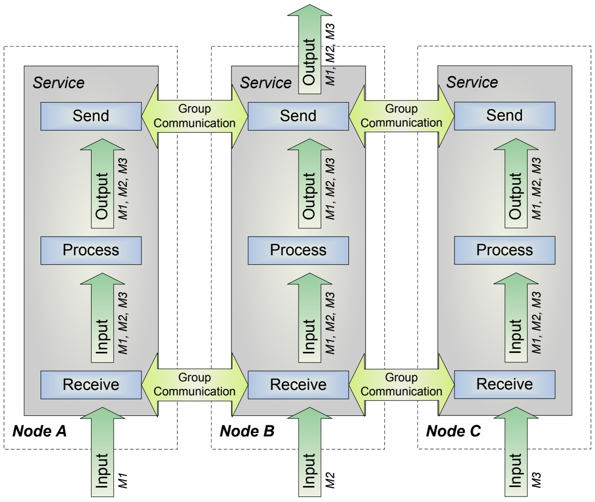

Figure 4. Universal Symmetric Active/Active High Availability Architecture for Services

Symmetric active/active high availability [8] (Figure 4) allows more than one redundant service to be active, i.e. to accept state changes, while it does not waste system resources by relying on an idle standby. Furthermore, there is no interruption of service and no loss of state, since active services run in virtual synchrony without the need to failover.

Service state replication is performed by totally ordering all state change messages and reliably delivering them to all redundant active services. A process group communication system is used to ensure total message order and reliable message delivery as well as service group membership management.

Furthermore, consistent output produced by all active services, i.e. messages sent to other parts of the system or return messages related to service state changes, is routed through the group communication system, using it for a distributed mutual exclusion to ensure that output is delivered only once.

The number of active services is variable at runtime and can be changed by either forcing an active service to leave the service group or by joining a new service with the service group. This allows adaptation of the service group size, such as adding new nodes when old nodes become less reliable, and reconfiguration of the service group behavior, for example when performing live upgrades in order to fix discovered vulnerabilities or improve throughput performance.

As long as one active service is alive, state is never lost, state changes can be performed and output is produced accordingly to state changes.

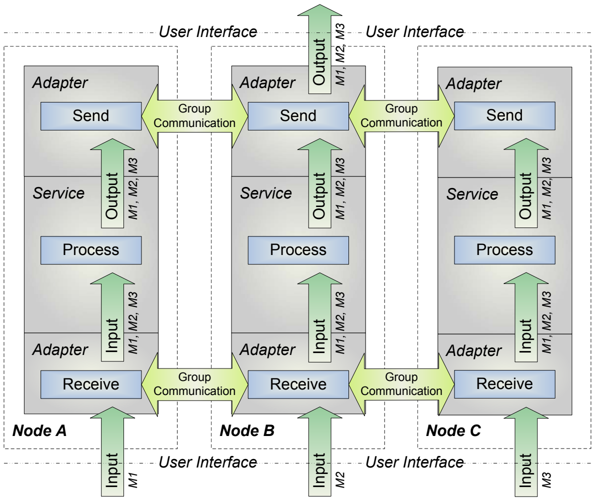

Figure 5. Symmetric Active/Active High Availability Architecture using Internal Replication by Service Modification/Adaptation

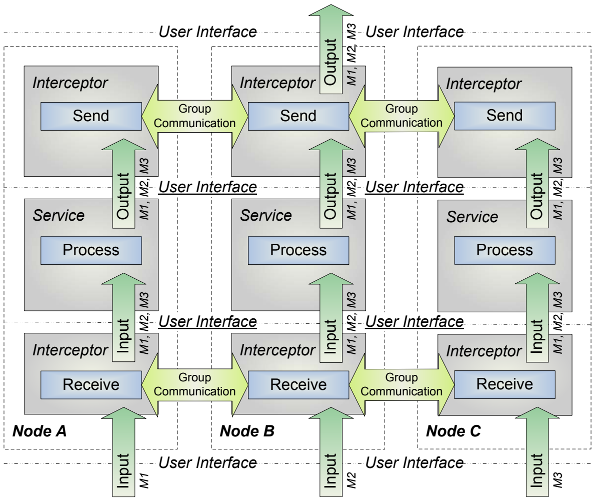

Figure 6. Symmetric Active/Active High Availability Architecture using External Replication by Service User Interface Utilization

4. Replication Methods

Implementing symmetric active/active high availability using virtual synchrony supported by a group communication system implies event-based programming, where a service only reacts to event messages using uninterruptible event handler routines.

More advanced programming models for virtual synchrony, such as distributed control [10], use the replicated remote procedure call abstraction to provide a request/response programming model.

However, both programming models assume that a service supplies the necessary hooks to perform uninterruptible state transitions.

While this is typically the case for networked services that have some sort of event notification, remote procedure call or remote method invocation interface, command line based services, such as the batch job scheduler in HPC systems, and proprietary network services, such as data storage and I/O in HPC systems, do not necessarily offer these hooks.

Adaptation to the event-based or request/response programming model can be performed either internally by modifying the service itself or externally by wrapping it into a virtually synchronous environment.

4.1. Internal Replication

Internal replication (Figure 5) allows each active service of a service group to accept external state change requests individually, while using a group communication system for total message order and reliable message delivery to all members of the service group. All state changes are performed in the same order at all services, thus virtual synchrony is given.

Services may also choose fine-grain synchronization using the group communication system to perform state changes in multiple stages by splitting them into smaller atomic operations.

Since state changes are handled in an uninterruptible fashion, splitting them up allows interleaving in order to gain performance and reduce response latency. However, only non-conflicting state changes may be interleaved in order to maintain correctness.

Interleaving state changes is a concurrency problem similar to pipelined execution in processors.

4.2. External Replication

External replication (Figure 6) avoids modification of existing code by wrapping a service into a virtually synchronous environment. Interaction with other services or with the user is intercepted, totally ordered and reliably delivered to the service group using a group communication system that mimics the service interface using separate event handler routines.

For example, the command line interface of a service is replaced with an interceptor command that behaves like the original, but forwards all input to an interceptor group. Once totally ordered and reliably delivered, each interceptor group member calls the original command to perform operations at each service group member. Service group output is routed through the interceptor group for at most once delivery.

External replication implies coarse-grain synchronization. Interleaving state changes is not possible, since the interceptor group forces atomicity for all service interface operations.

4.3. Comparison

Both methods, external and internal replication, rely on a group communication system for totally ordered reliable message delivery. Assuming correctness of the group communication system, both methods are still susceptible to implementation errors due to the adaptation to the event-based or request/response programming model. This may cause incorrect run time behavior and inconsistent state.

Implementers have to carefully examine the properties of a service before adapting it to the active/active high availability model. The replication methods presented in this paper are for deterministic services only. Non-determinism of a service is often caused by reliance on non-replicated components, such as a local random number generator or clock, or by multithreading. For example, the HPC job management system service may rely on a local clock for job scheduling.

Internal replication requires modification of existing code, which may be unsuitable for complex and/or large services. The amount of modification necessary may result in a complete redesign and reimplementation of an existing service. In contrast, the external replication method wraps an existing solution into a virtually synchronous environment without modifying it.

Our ongoing work focuses on providing symmetric active/active high availability for the TORQUE [25] job and resource management system using external replication and for the Parallel Virtual File System (PVFS) [20] metadata server using internal replication (both using the Transis group communication system). While we have not performed extensive performance benchmarking, early results show that internal replication offers higher performance with interleaved state changes and slightly reduces the response latency introduced by the group communication system. However, interleaving state changes requires substantial knowledge about the internal behavior of a service.

Another issue emerges when considering *live upgrades* of service nodes. Upgrading a highly available system service while it is running in a symmetric active/active fashion requires removal, upgrade and rejoin of each individual service group member, one at a time. A strict prerequisite of such a procedure is that the new version of the service fully supports the interface (including its semantics) of the old one. Similar to live upgrades, individual symmetric active/active high availability solutions may be reused for different service implementations that provide the same user interface, for example different HPC job and resource management systems supporting the Open PBS [18] interface.

Maintaining a consistent interface between service and group communication system over a significant period of time is easier using external replication as it is based on the external user interface of a service, while the internal design of the service is not affected and may change with a new version.

Overall, we recommend the internal replication method when high throughput performance is needed, for example for file system servers, except where the prospect of extensive code modification or foreseeable major service design changes prohibits it.

However, we recommend using external replication when high throughput performance is not a major requirement and the symmetric active/active high availability solution should be reusable for other services that have exactly the same user interface.

5. Distributed Mutual Exclusion

The purpose of distributed mutual exclusion in the context of symmetric active/active high availability using virtual synchrony is to ensure at most once delivery of service output even in the presence of failures.

Distributed mutual exclusion can be simply implemented with the help of a group communication system by maintaining a replicated lock at each member of a process group. This lock is acquired and released at all process group members in virtual synchrony using atomic multicast operations containing the member id of the requesting process group member. A lock held by a failed process group member is automatically released at all process group members when receiving the atomic multicast of the respective view change.

When running multiple services in virtual synchrony, the lock assures mutually exclusive access for service output operations by relying on an output transaction counter that is maintained as part of the replicated service state. This output transaction counter starts with zero and is increased for every output produced by a service. Each locking request is atomically multicast to all members of the service group containing the member id of the requesting service group member and the current output transaction counter value. Once the lock has been acquired, the output is sent only if this transaction counter value is higher than the one associated to the last successfully sent output. If this is not the case, the lock is immediately released at all members of the service group without the need for communication.

Once the service output has been successfully received, the lock is released and the transaction counter of the last successfully sent output is updated at all members of the service group using an atomic multicast by the system component that received the service output. However, if the service holding the lock fails, the lock is released at all members of the service group without updating the transaction counter of the last successfully sent output. In this case, the next service that acquires the lock sends the respective output.

Some group communication systems, such as Transis, have advanced built-in capabilities that can drastically simplify the implementation of distributed mutual exclusions. Most distributed mutual exclusion implementations follow the same or a similar algorithm.

6. Byzantine Failures

Our approach in providing symmetric active/active high availability for HPC system services on head and service nodes using virtual synchrony primarily focuses on tolerating *benign failures*, assuming that system components, such as individual services, nodes or communication links, fail by simply stopping.

Our solution presently does not guarantee correctness if a single faulty component violates this assumption by producing false output.

Process group communication systems that tolerate *Byzantine failures* provide a complete answer to this problem by making no assumptions about the behavior of faulty components. However, most solutions are impractical and tend to be inefficient.

We are currently in the process of analyzing past research in practical Byzantine fault tolerance [3] for networked services and its application to our symmetric active/active high availability architecture for HPC system services on head and service nodes.

However, due to the fact that symmetric active/active high availability solutions have not been used until now in HPC systems, there is not much experience with the occurrence of Byzantine failures in this context. Future research also needs to focus on isolating, quantifying and classifying failures in HPC systems.

7. Related Work

Ongoing work in this area includes a collaborative research effort between our teams at Oak Ridge National Laboratory (ORNL), Louisiana Tech University and Tennessee Tech University to design and develop active/active high availability support for existing HPC system software solutions, such as batch job scheduler, resource management, message passing layer and data storage, based on the discussed replication methods.

Related past and ongoing research at ORNL in partnership with North Carolina State University deals with exploration of advanced group communication algorithms [6, 10], flexible and modular group communication frameworks [9], and support for scalable high availability on compute nodes.

Other related past research includes object-oriented replication approaches, such as the Object Group Pattern, Orbix+Isis and Electra.

The Object Group Pattern [14] offers support for replicated objects using a group communication system for virtual synchrony. In this design pattern, objects are designed as state machines and replicated using totally ordered and reliably multicast state changes similar to the approach of this paper. Furthermore, it also provides the necessary hooks for copying object state, which is needed for joining group members.

Orbix+Isis and Electra are follow-on research projects [13] that focus on extending high availability support to CORBA using object request brokers (ORBs) on top of virtual synchrony toolkits.

Further related research also includes operating system extensions to support high availability for cluster computing, such as the OpenAIS effort, and highly available run time environments for distributed heterogeneous computing, like Harness.

OpenAIS [17] is an implementation of the Service Availability Forums [22] API Specification for cluster high availability. It consists of Availability Management Framework (AMF), Cluster Membership (CLM), Checkpointing (CKPT), Event (EVT) notification, Messaging (MSG), and Distributed Locks (DLOCK). OpenAIS recently moved towards using virtual synchrony for distributed locks and membership management.

Harness [11, 24] is a pluggable heterogeneous environment for distributed scientific computing. Conceptually, it consists of two major parts: a runtime environment (RTE) and a set of plug-in software modules. While the RTE provides only basic functions, plug-ins may provide a wide variety of services, such as messaging, scientific algorithms and resource management. Multiple RTE instances can be aggregated into a highly available Distributed Virtual Machine (DVM) using distributed control [10] for replication.

As mentioned before, past research in high availability for HPC system services primarily focused on the active/standby model using at least one redundant idle backup node. Most of these solutions experience an interruption of service and a loss of service state during a failover. A more detailed description and a classification of availability models and their replication strategies in the context of HPC system architectures can be found in an earlier paper [8].

Lastly, there is a plethora of past research on group communication algorithms [4, 5].

8. Conclusions

With this paper, we introduced two distinct replication methods (internal and external) for providing active/active high availability for HPC system services using the virtual synchrony paradigm.

We discussed the virtual synchrony paradigm and its application to the active/active replication model. We presented the internal and external replication methods, and compared them in terms of expected correctness, ease-of-use and performance.

Our findings based on early results from ongoing work in providing symmetric active/active high availability for two HPC system services (TORQUE and PVFS metadata server) recommend using the internal replication method when high throughput performance is needed, except where the prospect of extensive code modification or foreseeable major service design changes prohibits it.

External replication is a feasible alternative with slightly lower performance and higher response latency, and is recommended when the symmetric active/active high availability solution should be reused with other services that have the same user interface, except where the anticipated performance loss prohibits it.

Furthermore, we recommend to carefully examine the properties of a service before adapting it to active/active high availability. Virtual synchrony is event based and requires services to supply the necessary uninterruptible hooks for state changes.

Our ongoing research in this area focuses on the design and development of active/active high availability support for existing HPC system software solutions, such as batch job scheduler, resource management, message passing layer and data storage, based on the replication methods discussed in this paper.

References

[1] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, 1993.

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

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance. *Proceedings of OSDI*, pages 173–186, 1999.

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

[5] X. Defago, A. Schiper, and P. Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[6] N. Desai and F. Mueller. Scalable hierarchical locking for distributed systems. *Parallel and Distributed Computing*, 64(6):708–724, 2004.

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

[8] C. Engelmann and S. Scott. Concepts for high availability in scientific high-end computing. *Proceedings of HAPCW*, 2005.

[9] C. Engelmann and S. Scott. High availability for ultra-scale high-end scientific computing. *Proceedings of COSET-2*, 2005.

[10] C. Engelmann, S. L. Scott, and G. A. Geist. Distributed peer-to-peer control in Harness. *Lecture Notes in Computer Science: Proceedings of ICCS 2002*, 2330:720–728, 2002.

[11] 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.

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

[13] S. Landis and S. Maffeis. Building reliable distributed systems with CORBA. *Theory and Practice of Object Systems*, 3(1):31–43, 1997.

[14] S. Maffeis. The object group design pattern. *Proceedings of USENIX COOTS*, 1996.

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

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

[17] OpenAIS at Open Source Development Labs, Beaverton, OR, USA. http://developer.osdl.org/dev/openais.

[18] OpenPBS at Altair Engineering, Troy, MI, USA. http://www.openpbs.org.

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

[20] PVFS at Clemson University, Clemson, SC, USA. http://www. parl.clemson.edu/pvfs.

[21] R. I. Resnick. A modern taxonomy of high availability. 1996. http://www.generalconcepts.com/resources/reliability/resnick/HA.htm.

[22] Service Availability Forum. http://www.saforum.org.

[23] SLURM at Lawrence Livermore National Laboratory, Livermore, CA, USA. http://www.llnl.gov/linux/slurm.

[24] V. Sunderam and D. Kurzyniec. Lightweight self-organizing frameworks for metacomputing. *Proceedings of HPDC*, pages 113–124, 2002.

[25] TORQUE at Cluster Resources, Inc., Spanish Fork, UT, USA. http://www.clusterresources.com.