JOSHUA: Symmetric Active/Active Replication for Highly Available HPC Job and Resource Management*†

K. Uhlemann[1,2], C. Engelmann[1,2], S. L. Scott[1]
[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
{uhlemannk,engelmannc,scottsl}@ornl.gov
{k.uhlemann,c.engelmann}@reading.ac.uk

Abstract

Most of today's HPC systems employ a single head node for control, which represents a single point of failure as it interrupts an entire HPC system upon failure. Furthermore, it is also a single point of control as it disables an entire HPC system until repair. One of the most important HPC system service running on the head node is the job and resource management. If it goes down, all currently running jobs loose the service they report back to. They have to be restarted once the head node is up and running again.

With this paper, we present a generic approach for providing symmetric active/active replication for highly available HPC job and resource management. The JOSHUA solution provides a virtually synchronous environment for continuous availability without any interruption of service and without any loss of state. Replication is performed externally via the PBS service interface without the need to modify any service code. Test results as well as availability analysis of our proof-of-concept prototype implementation show that continuous availability can be provided by JOSHUA with an acceptable performance trade-off.

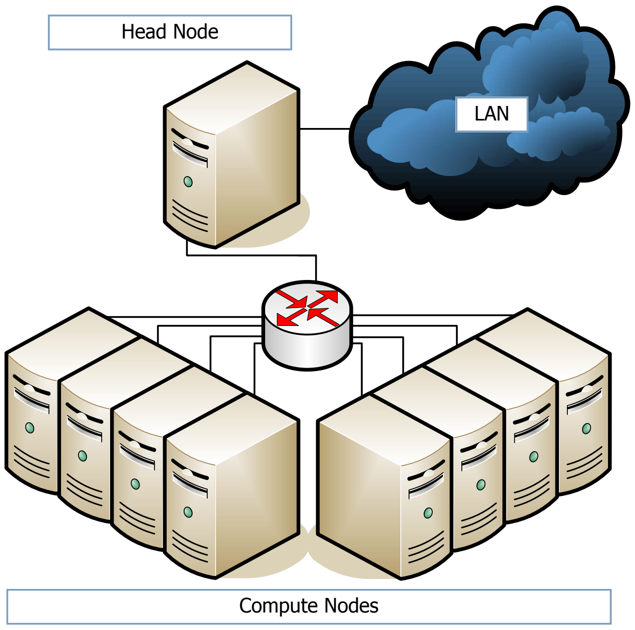

Figure 1. Traditional Beowulf Cluster Architecture with Single Head Node

1 Introduction

During the last decade, high-performance computing (HPC) has evolved into an important tool for scientists world-wide to drive the race for scientific discovery in numerous research areas, such as climate dynamics, nuclear astrophysics, fusion energy, materials sciences, and biology. Today, scientific knowledge can be gained without the immediate need or capability of performing physical experiments using computational simulations of real-world and theoretical experiments based on mathematical abstraction models.

The emergence of cluster computing in the late 90's made low- to mid-end scientific computing affordable to

*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.

everyone, while it introduced the Beowulf cluster system architecture [36, 37] (Figure 1) with its single head node controlling a set of dedicated compute nodes. This architecture has been proven to be very efficient as it permits customization of nodes and interconnects to their purpose. Many HPC vendors adopted it either completely in the form of high-end computing clusters or in part by developing hybrid high-end computing solutions.

Most of today‘s HPC systems employ a single head node for control and optional service nodes to offload specific head node services, such as a networked file system. This head node represents a *single point of failure* as it interrupts an entire HPC system upon failure. Furthermore, it is also a *single point of control* as it disables an entire HPC system until repair. This classification is also applicable for any service node offloading head node services.

The impact of a head node failure is severe as it not only causes significant system downtime, but also interrupts the entire system. Scientific applications experiencing a head node failure event typically have to be restarted as they depend on system services provided by the head node. Loss of scientific application state may be reduced using fault-tolerance solutions on compute nodes, such as checkpoint/restart [4] and message logging [25]. The introduced overhead depends on the system size, *i.e.*, on the number of compute nodes, which poses a severe scalability problem for next-generation petascale HPC systems.

A more appropriate protection against a head node failure is a redundancy strategy at the head node level. Multiple redundant head nodes are able to provide high availability for critical HPC system services, such as user login, network file system, job and resource management, communication services, and in some cases the OS or parts of the OS itself, *e.g.*, for single system image (SSI) systems. Furthermore, the availability of less critical services, like user management, software management, and programming environment, can be improved as well.

One of the most important HPC system service running on the head node is the job and resource management service, also commonly referred to as batch job scheduler or simply the scheduler. If this critical HPC system service goes down, all currently running jobs (scientific applications) loose the service they report back to, *i.e.*, their logical parent. They have to be restarted once the HPC job and resource management service is up and running again.

The research presented in this paper aims at providing continuous availability for HPC system services using symmetric active/active replication on multiple head nodes based on an existing process group communication system for membership management and reliable, totally ordered message delivery.

Specifically, we focus is on a proof-of-concept solution for Portable Batch System (PBS) compliant job and resource management services as a primary example. While the solution presented in this paper focuses only on the TORQUE/MAUI HPC job and resource management combination, the generic symmetric active/active high availability model our approach is based on is applicable to any deterministic HPC system service, such as to the metadata server of the parallel virtual file system (PVFS) [29].

Our approach in providing symmetric active/active high availability for HPC system services is based on the *fail-stop model* assuming that system components, such as individual services, nodes, or communication links, fail by simply stopping. We do not guarantee correctness if a single faulty service violates this assumption by producing false output.

In the following, we give a short overview of models for providing service-level high availability and outline the features of the symmetric active/active replication technique our work is based on. We continue with a detailed description of the JOSHUA solution for providing symmetric active/active replication for highly available HPC job and resource management. We present functional and performance test results as well as a theoretical availability analysis of the developed proof-of-concept prototype system. We close with a brief description of past and ongoing related work and a short summary of the presented research.

2 High Availability Models

Conceptually, high availability of a service is based on redundancy [30]. If it fails, the system is able to continue to operate using a redundant one. As a result, the mean time to recover can be decreased, loss of state can be reduced, and single points of failure and control can be eliminated. The level of high availability depends on the replication strategy. There are various techniques to implement high availability for services [10]. They include *active/standby* and *active/active*.

Active/standby high availability (Figure 2) follows the failover model. Service state is saved regularly to some shared stable storage. Upon failure, a new service is restarted or an idle one takes over with the most recent or even current state. This implies a short interruption of service for the time of the failover and may involve a rollback to an old backup. This means that all currently running scientific applications have to be restarted after a head node failover (see Section 6 for related work).

Asymmetric active/active high availability (Figure 3) further improves the availability properties of a system. In this model, two or more active services offer the same capabilities at tandem without coordination, while an optional idle service is ready to take over in case of a failure. This technique allows continuous availability of a stateless service with improved throughput performance. However, it has

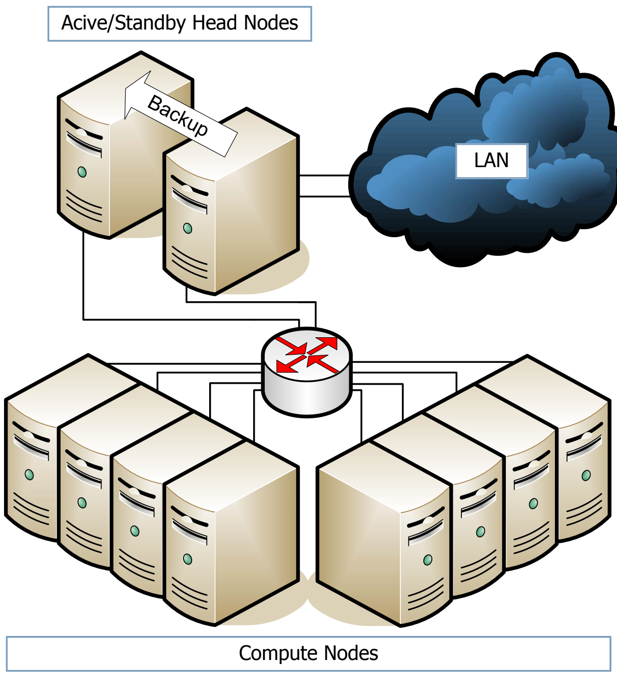

Figure 2. Enhanced Beowulf Cluster Architecture with Active/Standby High Availability for Head Node System Services

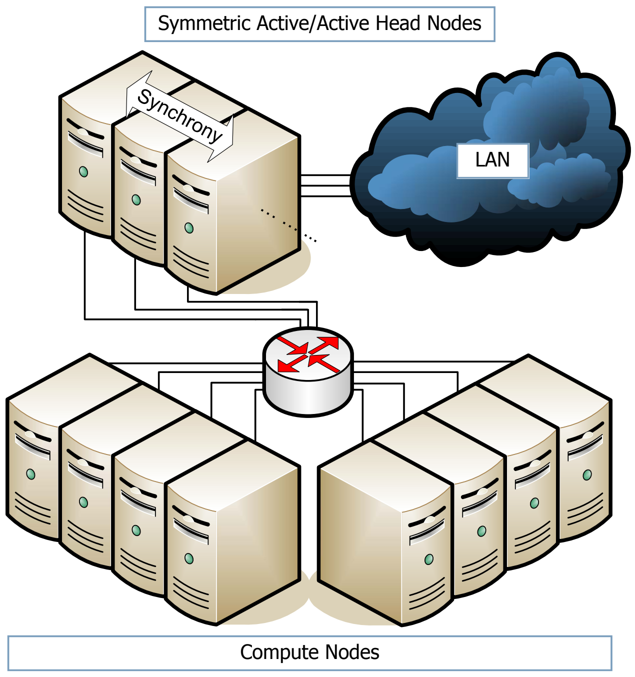

Figure 4. Advanced Beowulf Cluster Architecture with Symmetric Active/Active High Availability for Head Node System Services

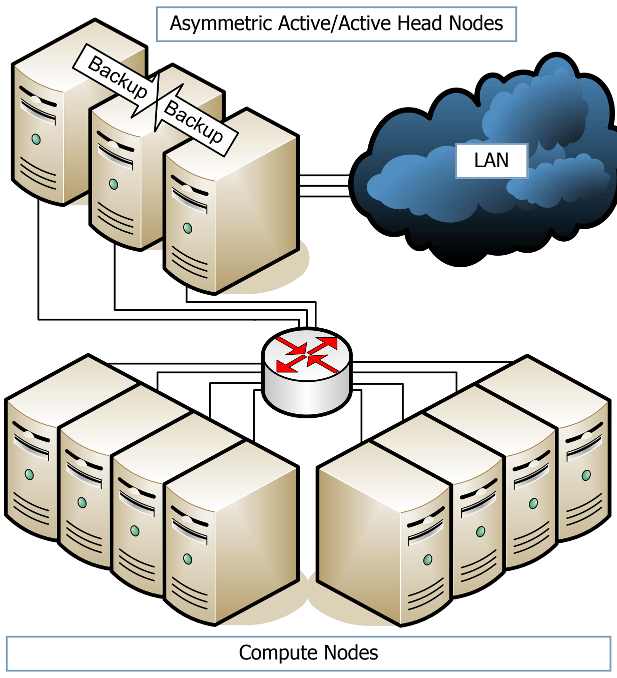

Figure 3. Advanced Beowulf Cluster Architecture with Asymmetric Active/Active High Availability for Head Node System Services

limited use cases due to the missing coordination between active services. In case of stateful services, such as the HPC job and resource management, this model only provides active/standby high availability with multiple active head nodes for improved throughput performance (see Section 6 for earlier prototype).

Symmetric active/active high availability (Figure 4) offers a continuous service provided by two or more active services that supply the same capabilities and maintain a common global state using *distributed control* [13] or *virtual synchrony* [24] via a process group communication system. The symmetric active/active model provides continuous availability for all kinds of services, but is significantly more complex due to need for advanced commit protocols.

The following section outlines mechanisms and properties of the symmetric active/active replication technique our work is based on.

3 Symmetric Active/Active Replication

While previous research in high availability for HPC system services (Section 6) primarily targeted active/standby and asymmetric active/active solutions, the work presented in this paper entirely focuses on the symmetric active/active model in order to provide *continuous availability* without

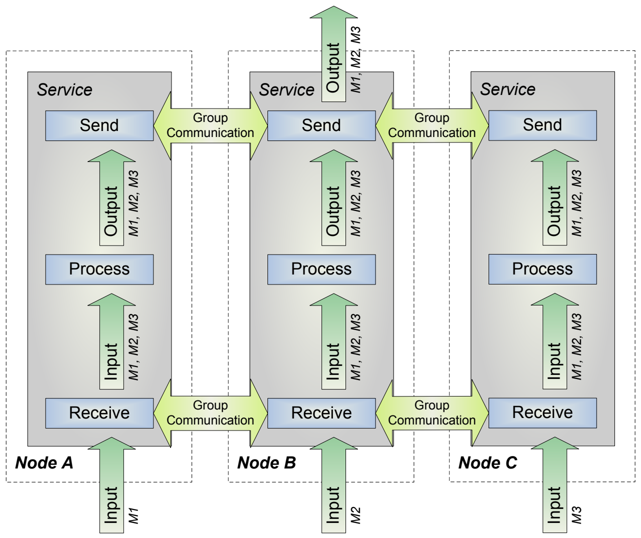

Figure 5. Universal Symmetric Active/Active High Availability Architecture for Services

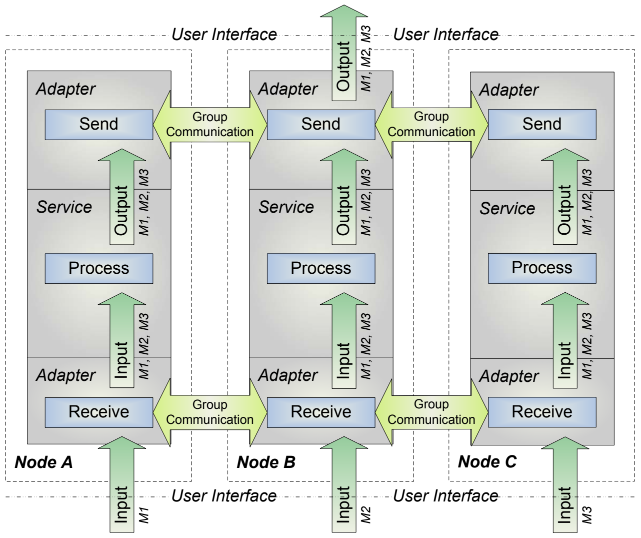

Figure 6. Symmetric Active/Active High Availability Architecture using Internal Replication by Service Modification/Adaptation

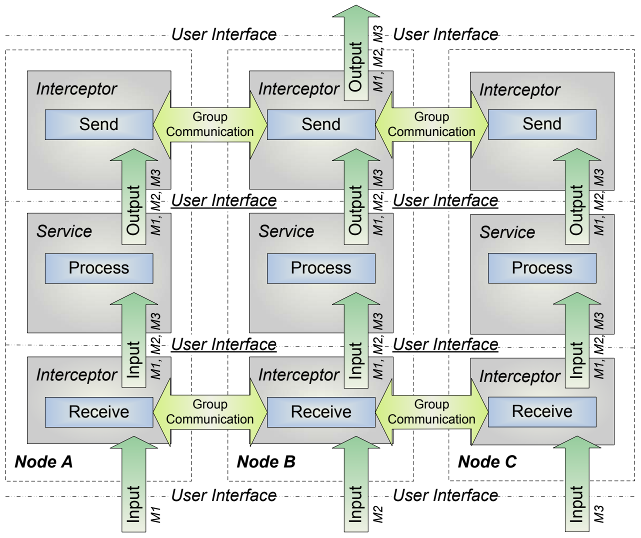

Figure 7. Symmetric Active/Active High Availability Architecture using External Replication by Service Interface Utilization

any interruption of service and without any loss of state.

The symmetric active/active high availability architecture for services (Figure 5) 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 in case of a failure, since active services run in virtual synchrony without the need to failover.

Service state replication is performed using a process group communication system for totally ordering and reliably delivering all state change messages to all redundant active services. Consistent output produced by these services is routed through the group communication system, using it for a distributed mutual exclusion to ensure that output is delivered only once.

The size of the active service group is variable at runtime, *i.e.*, services may join, leave or fail. Its membership is maintained by the group communication system in a fault tolerant, adaptive fashion ensuring group messaging properties. 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.

Symmetric active/active high availability using virtual synchrony supported by a process group communication system may be implemented in two distinctive ways, internally or externally [14]. *Internal replication* (Figure 6) requires modification of the service itself, while *external replication* (Figure 7) wraps the service into a virtually synchronous environment utilizing the service interface.

Internal replication 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. External replication 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. Internal replication may yield better throughput and latency performance, while external replication allows reusing the same solution for services with the same interface and without modification of existing service code.

Our approach for providing high availability for HPC job and resource management focuses on external replication. HPC job and resource management solutions typically support the Portable Batch System (PBS) [27] service interface. Our solution allows any PBS compliant job and resource management system to operate inside the virtually synchronous environment in a highly available fashion as it is based on external replication. Our test results (Section 5) show that the performance impact caused by the external replication is still in an acceptable range.

Furthermore, HPC job and resource management system implementations tend to be rather complex. Our non-intrusive approach does not modify any service code, *i.e.*, the service and the virtually synchronous environment can be further improved independently.

4 JOSHUA

The JOSHUA (job scheduler for high availability using active replication) solution is a generic approach for offering symmetric active/active high availability for HPC job and resource management services with a PBS compliant service interface. It represents a virtually synchronous environment using external replication based on the PBS service interface (Figure 8) providing continuous availability without any interruption of service and without any loss of state.

Conceptually, the JOSHUA software architecture (Figure 9) consists of three major parts: a server process (joshua) running on each head node, a set of control commands (jsub, jdel, and jstat) reflecting PBS compliant behavior to the user, and a set of scripts (jmutex and jdone) to perform a distributed mutual exclusion during job launch. Furthermore, JOSHUA relies on the Transis [9, 40] group communication system with its extended virtual synchrony implementation [24] for reliable, totally ordered message delivery.

We denote that JOSHUA does not provide a control command for signaling an executing batch job (qsig equivalent)

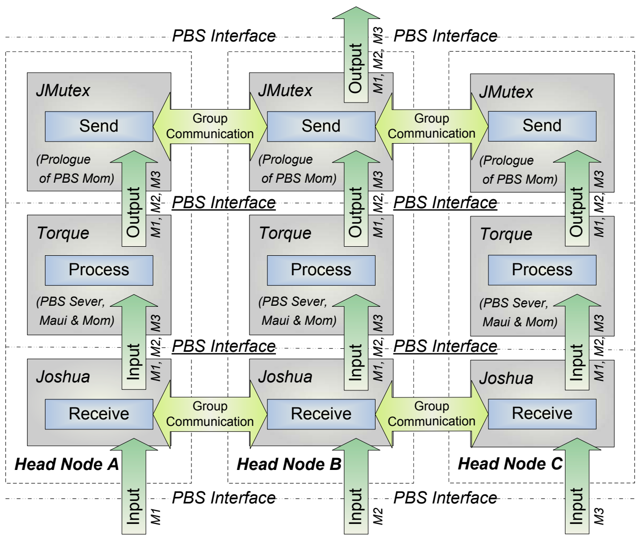

Figure 8. Symmetric Active/Active High Availability Architecture of the JOSHUA solution for HPC Job and Resource Management

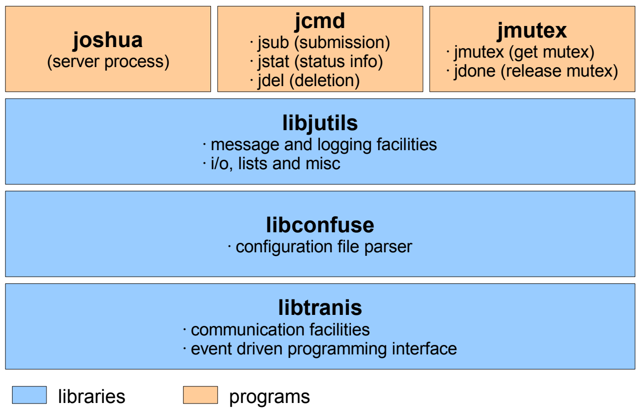

Figure 9. Individual Software Components of the JOSHUA solution for highly available HPC Job and Resource Management

at the moment as this operation does not appear to change the state of the HPC job and resource management service. The original PBS command may be executed independently of JOSHUA.

Our proof-of-concept implementation prototype, which has been implemented as part of a Master's thesis [41], is based on the PBS compliant TORQUE [39] HPC job and resource management system that employs the TORQUE PBS server together with the Maui [22] scheduler on each active head node and a set of PBS mom servers on compute nodes.

Each PBS mom server is capable of communicating to each TORQUE PBS server on every active head node (TORQUE v2.0p1 feature), which allows the reuse of PBS mom servers. However, this is not a requirement of the JOSHUA solution. Dedicated PBS mom servers may be used for each TORQUE PBS server instance. The Maui scheduling policy is set to FIFO (default) to produce deterministic scheduling behavior on all active head nodes. Furthermore, Maui is configured to give each job exclusive access to our test cluster to produce deterministic allocation behavior. This restriction may be lifted in the future if deterministic allocation behavior can be assured.

During normal operation, the JOSHUA control commands (jsub, jdel, and jstat) perform job submission, deletion and statistics retrieval by connecting to the JOSHUA server group, issuing the respective command (qsub, qdel, and qstat) locally at all active head nodes, and relaying the output back to the user. Fundamentally, the JOSHUA control commands and server act in concert as an interceptor for PBS user commands to provide global ordering of user input for virtual synchrony on all active head nodes. The JOSHUA control commands may be invoked on any of the active head nodes or from a separate login node as they contact the JOSHUA server group via the network. They may even replace the original PBS commands in the user context using a shell alias (e.g. `'alias qsub=jsub'`) in order to offer 100% PBS service interface compliance.

Once a submitted job is first in the PBS TORQUE job queue and there is no other job currently running, each PBS TORQUE server connects to a PBS mom server on the compute nodes to start the job. The JOSHUA scripts are part of the job start prologue and perform a distributed mutual exclusion using the Transis group communication system to ensure that the job gets started only once, and to emulate the job start for all other attempts for this particular job. Once the job has finished, the distributed mutual exclusion is released and all PBS TORQUE servers receive the respective job statistics report.

Upon failure of an active head node, the Transis group communication system informs all JOSHUA servers to exclude the failed head node from any further communication. The PBS mom servers simply ignore the failed head node when sending job statistics reports, while the distributed mutual exclusion performed by the JOSHUA scripts relies on the Transis group membership management for correctness. There is no failover necessary as the healthy active head nodes continue to provide the service and the system parts on the compute node are able to adapt. Since the Transis group communication system is able to deal with multiple simultaneous failures in the same way it deals with multiple sequential failures, continuous HPC job and resource management service availability is provided transparently as long as one head node survives.

Head node failures degrade the overall availability of the system by reducing the number of redundant components (Figure 12 in Section 5). Replacement of failed head nodes or of head nodes that are about to fail allows to sustain and guarantee a certain availability. The JOSHUA solution permits head nodes to join and leave using the Transis group communication system for coordination. Leaving the active service group is actually handled as a forced failure by causing the JOSHUA server to shutdown via a signal. Joining the active service group involves copying the current state of an active service over to the joining head node.

The current JOSHUA/TORQUE proof-of-concept prototype implementation uses configuration file modification and user command (message) replay to copy the state of one PBS TORQUE server over to another. This is due to the fact that PBS TORQUE does not provide an easy solution for starting up a replica on a system with a different host name and IP address. As a result, we were also unable to provide the capability of holding and releasing jobs as the user command replay causes inconsistencies in the job queue of the joining PBS TORQUE server when holding jobs. Future work will concentrate on using a unified and location independent PBS job and resource management state description, *e.g.*, the Scalable System Software [5, 32] interface specification.

5 Test Results

The JOSHUA/TORQUE proof-of-concept prototype implementation has been deployed on a small dedicated Linux test cluster environment using up to 4 head nodes and 2 compute nodes in various combinations for functional and performance testing. Individual nodes contained dual Intel Pentium III (Katmai) 450MHz processors with 512 MB of memory and 8 GB of disk space, and were connected via a single Fast Ethernet (100MBit/s full duplex) hub. Debian GNU/Linux 3.1 (sarge) has been used as operating system, while Transis v1.03, TORQUE v2.0p5, and Maui v3.2.6p13 were used in conjunction with JOSHUA v0.1. Failures were simulated by unplugging network cables and by forcibly shutting down individual processes.

Extensive functional testing revealed correct behavior

System	#	Latency	Overhead
TORQUE	1	98 ms	
JOSHUA/TORQUE	1	134 ms	36 ms / 37%
JOSHUA/TORQUE	2	265 ms	158 ms / 161%
JOSHUA/TORQUE	3	304 ms	206 ms / 210%
JOSHUA/TORQUE	4	349 ms	251 ms / 256%

Figure 10. Job Submission Latency Comparison of Single vs. Multiple Head Node HPC Job and Resource Management

System	#	10 Jobs	50 Jobs	100 Jobs
TORQUE	1	0.93s	4.95s	10.18s
JOSHUA/TORQUE	1	1.32s	6.48s	14.08s
JOSHUA/TORQUE	2	2.68s	13.09s	26.37s
JOSHUA/TORQUE	3	2.93s	15.91s	30.03s
JOSHUA/TORQUE	4	3.62s	17.65s	33.32s

Figure 11. Job Submission Throughput Comparison of Single vs. Multiple Head Node HPC Job and Resource Management

during normal system operation and in case of single and multiple simultaneous failures with two exceptions (see next paragraph). Head nodes were able to join the service group, leave it voluntary, and fail, while job and resource management state was maintained consistently at all head nodes and continuous service was provided to applications and to users.

However, after 3-5 days of excessive operation with upto hundreds of job submissions a minute Transis crashed and needed to be restarted. While the reason of this behavior is still under investigation, we suspect incorrect memory allocation/deallocation of Transis to be the primary cause. A more stable group communication system will be used in the future. Furthermore, the PBS mom servers did not simply ignore a failed head node, but rather kept the current job in running status until it returned to service. This deficiency has been communicated to the TORQUE developers and we expect a fix in the near future. Both of these issues are the major reasons why the JOSHUA solution has not yet been deployed on production HPC environments.

Furthermore, the work presented in this paper focuses only on head node high availability. The PBS mom server and the JOSHUA scripts run on compute nodes. The JOSHUA solution is not capable of fully tolerating failures of compute nodes running the PBS mom server. This is due to the fact that the currently running application may loose its parent process and/or a PBS TORQUE server may loose the PBS mom server it currently communicates with. A solution for this problem was outside the scope of this research project, but will be targeted in the future.

The JOSHUA/TORQUE proof-of-concept prototype implementation showed a comparable latency and throughput performance. The job submission latency overhead (Figure 10) introduced by network communication between the JOSHUA commands, the Transis group communication system and the JOSHUA server was in an acceptable range. The latency overhead between TORQUE and JOSHUA/TORQUE on a single head node (37%) can be attributed to communication on the same node, while the significant latency overhead increase between JOSHUA/TORQUE on a single and on two head nodes (439%) can be explained by off-node communication. Overall a latency overhead of only 250ms for a 4 head node system is still acceptable for any HPC system.

The job submission throughput overhead (Figure 11) reflected similar characteristics. Considering high throughput HPC scenarios, such as in computational biology or on-demand cluster computing, adding 100 jobs to the job queue in 33s for a 4 head node system is also an acceptable tradeoff. Furthermore, we have to point out that developers of HPC job and resource management solutions have acknowledged throughput deficiencies of their systems for scenarios of submitting a large number of jobs at once. One already deployed solution allows a command line job submission to contain a number of individual jobs.

The JOSHUA solution needs to be deployed on a production-type HPC environment and respective reliability, availability and serviceability (RAS) metrics have to be recorded in order to measure its true availability impact. However, the JOSHUA solution is not yet mature enough and RAS metrics in a HPC environment are not well defined.

A theoretical availability analysis of the JOSHUA solution can be performed based on availability (A_{node}), mean time to failure ($MTTF_{node}$), and mean time to restore ($MTTR_{node}$) of an individual head node (Equation 1). The overall service availability ($A_{service}$) can be calculated based on parallel redundancy (Equation 2), since the JOSHUA solution provides continuous availability without interruption, *i.e.*, without increasing $MTTR_{node}$ and without introducing an additional system-wide $MTTR$. The estimated service downtime per year ($t_{service\ down}$) of the JOSHUA solution directly depends on its service availability ($A_{service}$) and relates to the event of all head nodes being down at the same time (Equation 3).

$$A_{node} = \frac{MTTF_{node}}{MTTF_{node} + MTTR_{node}} \tag{1}$$

$$A_{service} = 1 - (1 - A_{node})^{number\ of\ nodes} \tag{2}$$

$$t_{service\ down} = 8760 \cdot (1 - A_{service}) \tag{3}$$

#	Availability	Nines	Downtime/Year
1	98.6%	1	5d 4h 21min
2	99.98%	3	1h 45min
3	99.9997%	5	1min 30s
4	99.999996%	7	1s

(Based on MTTF=5000h and MTTR=72h)

Figure 12. Availability/Downtime Comparison of Single vs. Multiple Head Node HPC Job and Resource Management

Using a rather low MTTF of 5000 hours and a MTTR of 72 hours for an individual head node, the expected downtime of a single head node is over 5 days within a year (Figure 12). Deploying the JOSHUA solution on two head nodes reduces the annual downtime to only 1 hour and 45 minutes with a latency overhead of only 158 milliseconds. Adding another head node decreases the downtime to 1 1/2 minutes with a latency overhead of 206 milliseconds. Finally, a 4 head node solution offers an availability of 7 nines with an annual downtime of 1 second and a still acceptable latency overhead of 251 milliseconds.

This availability analysis shows that the symmetric active/active high availability for HPC job and resource management provided by the developed JOSHUA solution can provide continuous availability with an acceptable performance trade-off.

However, this analysis does not show the impact of correlated failures, such as caused by overheating of a rack or computer room. The deployment of multiple redundant head nodes also needs to take into account these location dependent failure causes. Furthermore, an availability of 7 nines as provided by 4 active head nodes may not be a realistic target for a single HPC system as disaster scenarios (flood, hurricane, tornado, and terrorist attack) are not considered.

6 Related Work

Past research and development efforts in high availability for HPC system services primarily focused on the active/standby model. Most solutions experience an interruption of service and a loss of service state during a failover. For example, HA-OSCAR [12, 17, 18, 21] and SLURM [34] provide active/standby solutions for the HPC job and resource management system in a warm-standby fashion, *i.e.*, requiring applications to restart after a failover of 3-5 seconds. However, PBSPro for the Cray XT3 [28] offers a hot-standby solution with transparent failover.

Earlier research of our teams at Oak Ridge National Laboratory (ORNL) and Louisiana Tech University focused on an asymmetric active/active solution [20] for HPC job and resource management in a high-throughput computing scenario.

Ongoing work at ORNL, Louisiana Tech University, and Tennessee Tech University focuses on developing symmetric active/active high availability support for other existing HPC system services [14], such as the PVFS metadata server [29].

Further related past and ongoing research at ORNL in collaboration with North Carolina State University deals with advanced group communication algorithms [8, 13], flexible modular high availability frameworks [11, 12], and support for scalable fault tolerance on compute nodes [12, 42].

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

OpenAIS [26] is an implementation of the Service Availability Forums [33] API specification for cluster high availability. It consists of an Availability Management Framework (AMF), Cluster Membership (CLM), Checkpointing (CKPT), Event (EVT) notification, Messaging (MSG), and Distributed Locks (DLOCK). Recently, the OpenAIS effort moved towards using a process group communication system for distributed locks and membership management.

Harness [16, 19, 38] 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 [13] for replication.

Further related work also includes recent research in practical Byzantine fault tolerance for networked services [23, 31]. Solutions tolerating *Byzantine failures* are able to guarantee correctness even if a single faulty or malicious service produces false output. However, they are even more complex and the existence of Byzantine failures in production HPC environments has not yet been verified.

Lastly, there is a plethora of past research on process group communication algorithms and systems [6, 7] dealing with various group messaging semantics, correctness proofs, and efficient implementations for different deployment scenarios. While our work is based on Transis, many others, such as Ensemble [3, 15] and Spread [1, 2, 35], exist and are candidates to replace Transis in order to improve stability. However, any replacement needs to be able to operate in an HPC environment and deal with its restrictions, such as the lack of support for certain programming languages and runtime environments, *e.g.*, Java and OCaml.

7 Conclusion

We presented the JOSHUA solution as a generic approach for providing symmetric active/active replication for highly available HPC job and resource management. JOSHUA provides a virtually synchronous environment using external replication based on the PBS service interface for continuous availability without any interruption of service and without any loss of state.

We gave a short overview of models for providing service-level high availability and outlined the features of the symmetric active/active replication technique JOSHUA is based on. We provided details about the proof-of-concept prototype implementation, which relies on TORQUE and Maui for HPC job and resource management and on Transis for group communication. Further implementation details have been published in a Master's thesis [41].

We presented functional and performance test results as well as a theoretical availability analysis analysis. We were able to show that continuous availability of the HPC job and resource management service can be provided without interruption and with an acceptable performance trade-off using the JOSHUA solution.

Future work will focus on improving JOSHUA for deployment in production HPC environments as well as providing input to HPC system service developers to enable further production-type high availability solutions. Furthermore, we will continue our effort in developing symmetric active/active high availability support for other existing HPC system services.

The experience gained with the design and development of JOSHUA has helped us to understand advanced high availability concepts, their practical application to the HPC world, and their pitfalls when misapplied. It is our hope that this work also provides an educational benefit for other developers of high availability solutions.

References

[1] Y. Amir, R. Caudy, A. Munjal, T. Schlossnagle, and C. Tutu. N-way fail-over infrastructure for reliable servers and routers. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN) 2003*, volume 2862, page 403, San Francisco, CA, USA, June 22-25, 2003.

[2] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The spread toolkit: Architecture and performance. Technical Report CNDS-2004-1, Johns Hopkins University, Center for Networking and Distributed Systems, Baltimore, MD, USA, 2004. Available at http://www.spread.org.

[3] 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 the DARPA Information Survivability Conference and Exposition (DISCEX) 2000*, 1:149–161, Jan. 25-27 2000.

[4] Berkeley Lab Checkpoint/Restart (BLCR) project at Lawrence Berkeley National Laboratory, Berkeley, CA, USA. Available at http://ftg.lbl.gov/checkpoint.

[5] B. Bode, R. Bradshaw, E. DeBenedictus, N. Desai, J. Duell, G. A. Geist, P. Hargrove, D. Jackson, S. Jackson, J. Laros, C. Lowe, E. Lusk, W. McLendon, J. Mugler, T. Naughton, J. P. Navarro, R. Oldfield, N. Pundit, S. L. Scott, M. Showerman, C. Steffen, and K. Walker. Scalable system software: A component-based approach. *Journal of Physics: Conference Series*, 16:546–550, 2005.

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

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

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

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

[10] C. Engelmann and S. L. Scott. Concepts for high availability in scientific high-end computing. In *Proceedings of High Availability and Performance Workshop (HAPCW) 2005*, Santa Fe, NM, USA, Oct. 11, 2005.

[11] C. Engelmann and S. L. Scott. High availability for ultra-scale high-end scientific computing. In *Proceedings of 2^{nd} International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2) 2005*, Cambridge, MA, USA, June 19, 2005.

[12] C. Engelmann, S. L. Scott, D. E. Bernholdt, N. R. Gottumukkala, C. Leangsuksun, J. Varma, C. Wang, F. Mueller, A. G. Shet, and P. Sadayappan. MOLAR: Adaptive runtime support for high-end computing operating and runtime systems. *ACM SIGOPS Operating Systems Review (OSR)*, 40(2):63–72, 2006.

[13] C. Engelmann, S. L. Scott, and G. A. Geist. High availability through distributed control. In *Proceedings of High Availability and Performance Workshop (HAPCW) 2004*, Santa Fe, NM, USA, Oct. 12, 2004.

[14] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He. Active/active replication for highly available HPC system services. In *Proceedings of The First International Conference on Availability, Reliability and Security (ARES) 2006*, pages 639–645, Vienna, Austria, Apr. 20-22, 2006.

[15] Ensemble Project at Cornell University, Ithaca, NY, USA. Available at http://www.cs.cornell.edu/Info/Projects/Ensemble.

[16] 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.

[17] High Availability Open Source Cluster Application Resources (HA-OSCAR) project at Louisiana Tech University, Ruston, LA, USA. Available at http://xcr.cenit.latech.edu/ha-oscar.

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

[19] Harness project at Oak Ridge National Laboratory, Oak Ridge, TN, USA. Available at http://www.csm.ornl.gov/harness.

[20] C. Leangsuksun, V. K. Munganuru, T. Liu, S. L. Scott, and C. Engelmann. Asymmetric active-active high availability for high-end computing. In *Proceedings of* 2^{nd} *International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2) 2005*, Cambridge, MA, USA, June 19, 2005.

[21] K. Limaye, C. Leangsuksun, Z. Greenwood, S. L. Scott, C. Engelmann, R. Libby, and K. Chanchio. Job-site level fault tolerance for cluster and grid environments. In *Proceedings of IEEE International Conference on Cluster Computing (Cluster) 2005*, Boston, MA, USA, Sept. 26-30, 2005.

[22] Maui Cluster Scheduler at Cluster Resources, Inc., Spanish Fork, UT, USA. Available at http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.

[23] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Proceedings of the* 24^{th} *IEEE Symposium on Reliable Distributed Systems (SRDS) 2005*, pages 131–142, Orlando, FL, USA, Oct. 26-28, 2005.

[24] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. *Proceedings of* 14^{th} *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 56–65, June 21-24, 1994.

[25] MPICH-V message logging layer for MPI at University of Paris-South, France. Available at http://www.lri.fr/∼gk/MPICH-V.

[26] OpenAIS at Open Source Development Labs (OSDL), Beaverton, OR, USA. Available at http://developer.osdl.org/dev/openais.

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

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

[29] Parallel Virtual File System (PVFS) 2 at Clemson University, Clemson, SC, USA. Available at http://www.pvfs.org/pvfs2.

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

[31] R. Rodrigues, M. Castro, and B. Liskov. BASE: Using abstraction to improve fault tolerance. In *Proceedings of the* 18^{th} *ACM symposium on Operating Systems Principles (SOSP) 2001*, pages 15–28, Banff, AL, Canada, Oct. 21-24, 2001.

[32] SciDAC SSS: Scalable Systems Software Center of the Scientific Discovery through Advanced Computing program. Available at http://www.scidac.org/ScalableSystems.

[33] Service Availability Forum. Available at http://www.saforum.org.

[34] Simple Linux Utility for Resource Management (SLURM) project at Lawrence Livermore National Laboratory, Livermore, CA, USA. Available at http://www.llnl.gov/linux/slurm.

[35] Spread Toolkit at Spread Concepts LLC. Available at http://www.spread.org.

[36] T. Sterling. *Beowulf cluster computing with Linux*. MIT Press, Cambridge, MA, USA, 2002.

[37] T. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese. *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*. MIT Press, Cambridge, MA, USA, 1999.

[38] V. Sunderam and D. Kurzyniec. Lightweight self-organizing frameworks for metacomputing. *Proceedings of the* 11^{th} *IEEE International Symposium on High Performance Distributed Computing (HPDC) 2002*, pages 113–124, July 24-26, 2002.

[39] TORQUE Resource Manager at Cluster Resources, Inc., Spanish Fork, UT, USA. Available at http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[40] Transis group communication system project at Hebrew University of Jerusalem, Israel. Available at http://www.cs.huji.ac.il/labs/transis.

[41] K. Uhlemann. High availability for high-end scientific computing. Master's thesis, Department of Computer Science, University of Reading, UK, Mar. 2006.

[42] J. Varma, C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Scalable, fault-tolerant membership for MPI tasks on HPC systems. In *Proceedings of* 20^{th} *ACM International Conference on Supercomputing (ICS) 2006*, pages 219–228, Cairns, Australia, June 28-30, 2006.