SYMMETRIC ACTIVE/ACTIVE METADATA SERVICE FOR HIGHLY AVAILABLE CLUSTER STORAGE SYSTEMS

Li Ou[1], Christian Engelmann[2], Xubin He[1], Xin Chen[1], and Stephen Scott[2]

[1]Department of Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505, USA
{lou21, hexb, xchen21}@tntech.edu
[2]Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
{engelmannc, scottsl}@ornl.gov

ABSTRACT
In a typical distributed storage system, metadata is stored and managed by dedicated metadata servers. One way to improve the availability of distributed storage systems is to deploy multiple metadata servers. Past research focused on the active/standby model, where each active server has at least one redundant idle backup. However, interruption of service and loss of service state may occur during a fail-over depending on the used replication technique. The research in this paper targets the symmetric active/active replication model using multiple redundant service nodes running in virtual synchrony. In this model, service node failures do not cause a fail-over to a backup and there is no disruption of service or loss of service state. We use a fast delivery protocol to reduce the latency of total order broadcast. Our prototype implementation shows that high availability of metadata servers can be achieved with an acceptable performance trade-off using the active/active metadata server solution.

KEY WORDS
Metadata management, high availability, fast delivery protocol

1 Introduction

A file system typically consists of two types of data: user data and metadata. Metadata is very important, because it defines how a file system utilizes the storage space to manage the user data. Since metadata is ”the data of the data”, the disruption of metadata could result in the failure of the entire I/O system, while the loss of the user data normally only affects some user files. Any I/O request can be classified into either user data or metadata request. A study by Roselli et al. [20] shows that requests targeting at the metadata can account for up to 83% of the total amount of I/O requests in some applications.

In a typical parallel storage system [1, 8], metadata is stored and managed by dedicated metadata servers. There are three major components in such a typical storage system. Metadata servers maintains information about the files and directories in a file system. Data servers store file data. Clients send requests to the metadata server and data servers to store and retrieve the file data. This system architecture has been proved to be very efficient. However, it also implies several reliability deficiencies resulting in system-wide availability and serviceability issues [12]. An entire distributed storage system depends on the metadata server to function properly. This is a single point of failure.

One way to improve the availability of parallel storage systems is to deploy multiple metadata servers. Multiple servers backup each other. As long as at least one metadata server is alive, the entire system does not fail. Several models exist to perform reliable and consistent replication of service state to multiple redundant servers for high availability. Past research focused on the active/standby model [2, 3, 14, 15], where each service server has at least one redundant idle backup. However, interruption of service and loss of service state may occur during a fail-over depending on the replication technique (hot-, warm- or cold-standby).

The research presented in this paper targets the symmetric active/active replication model [12, 14] for high availability metadata servers using multiple redundant service nodes running in virtual synchrony[18]. In this model, server failures do not cause a fail-over to a backup and there is no disruption of service or loss of service state. All servers are active and ready to serve requests from clients. This architecture improves availability and reliability.

The total order communication[5, 11] is essential for active/active replication model, but the agreement on a total order usually bears a cost of performance. We use a *fast delivery* protocol [19] to reduce the latency of totally ordered broadcasting. The protocol performs well for both idle and active servers. Our results show that for write requests, the performance degradation is acceptable for typical distributed storage systems, and the throughput of read

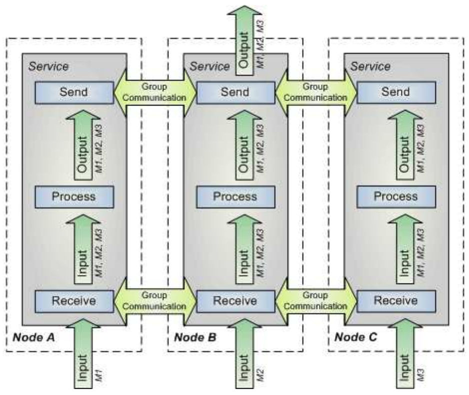


Figure 1. Universal symmetric Active/Active high availability architecture for services (©2006 IEEE).

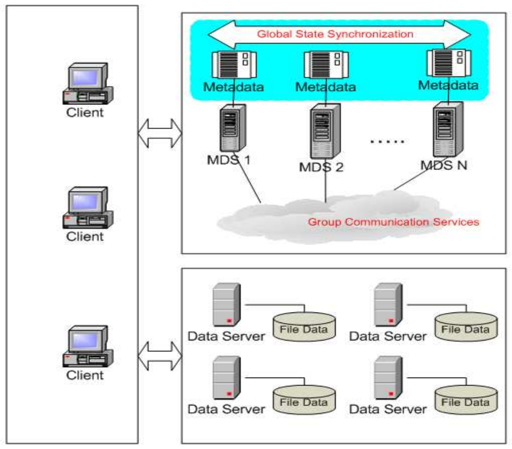


Figure 2. Active/Active metadata servers in a distributed storage system.

requests increases linearly with the number of servers. We are able to show that high availability of metadata servers can be achieved without interruption and with an acceptable performance trade-off using the active/active metadata server solution.

2 Symmetric Active/Active Replication

The symmetric active/active high availability architecture for services (Fig. 1[14]) allows more than one redundant service to be active, i.e., to accept state changes, while it does not waste system resources as seen in an active/standby model. Furthermore, there is no interruption of service and no loss of state, since active services run in virtual synchrony without the need to fail over.

The symmetric active/active replication uses virtual synchrony [18] and group communication [6, 11, 17] to guarantee the safety of global state updating. 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 system architecture of active/active metadata servers utilizing symmetric active/active replications is shown in Fig. 2.

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 according to state changes.

3 Fast Delivery Protocol for Total Order Broadcasting

Total order communication is essential for the symmetric active/active replication, but the agreement on a total order usually bears a cost of performance: a message is not delivered immediately after being received, until all the communication machines reach agreement on a single total order of delivery. Generally, the cost is measured as latency of totally ordered messages, from the point the message is ready to be sent, to the time it is delivered at the sender machine.

Among several algorithms implementing the total ordering, a communication history algorithm [10] is perfered to order requests among multiple active/active servers, since such algorithm performs well under heavy communication environments with concurrent requests, but the post-transmission delay of the algorithm is most apparent when the system is relatively idle, and in the worst case, the delay may be equal to the interval of heart beat messages from a idle machine.

We use a *fast delivery* protocol [19] to reduce the post-transmission delay. The *fast delivery* protocol forms the total order by waiting for messages only from a subset of the machines in the group, thus it fast deliverers total order messages. Furthermore, the fast acknowledgment aggressively acknowledges total order messages to reduce the latency when some machines are idle, and it is smart enough to hold the acknowledgments when the network communication is heavy. The protocol performances well for both idle and active servers.

4 Active/Active Metadata Server Design

Conceptually, the active/active metadata server software architecture (Fig. 3) consists of several major parts, to handle client requests, update global state, and manage membership of the server group. The current proof-of-concept prototype implementation uses Transis [11] with *fast delivery* protocol to provide total order and virtual synchrony services, and PVFS [8] to provide basic metadata services for each server.

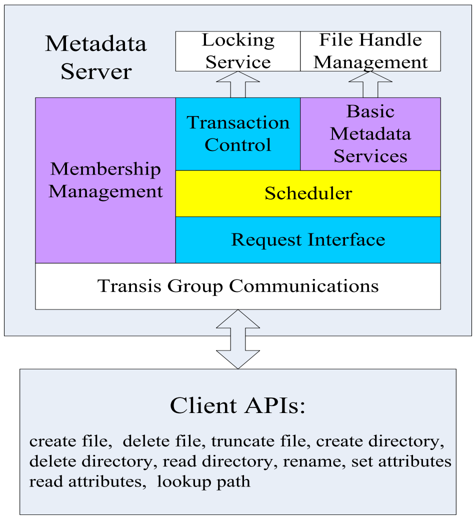

Metadata Server
Locking Service
File Handle Management
Transaction Control
Basic Metadata Services
Membership Management
Scheduler
Request Interface
Transis Group Communications
Client APIs:
create file, delete file, truncate file, create directory, delete directory, read directory, rename, set attributes read attributes, lookup path

Figure 3. Module design of Active/Active metadata servers.

We provide basic metadata manipulation APIs for clients. To balance workloads among multiple servers, a client randomly chooses a server to send a request. All client requests are sent to the request interface module. It interprets the requests, creates new jobs for them, then either dispatches the jobs directly to basic metadata services or requests the Transis to broadcast them. The jobs are first put into an active queue. The scheduler choses one active job to execute, until it is blocked by I/O operations and thus put to the idle queue. After I/O operations finish, the job is put back to the active queue and waiting to be scheduled. The scheduling mechanism guarantees that a metadata server is not blocked by any I/O operation and multiple concurrent requests could be interleaved, and thus improve the throughput of the server.

The basic metadata services module is responsible for updating local metadata and provides basic metadata management functions, such as create new object (file or directory), add a new entry into a directory, get attribute, and so on. Some client requests could be mapped to basic metadata services directly, such as get attribute, set attribute, but some updating requests involves several basic metadata services, and are considered as atomic operations. For example, a create new file request involves three basic services: reading directory to make sure no object has same name, create object, and add the handle of new object into the parent directory. It is an atomic operation, because failure of any step requires roll back of all submitted operations updating the local metadata. The transaction control module is responsible for processing such kind of requests, handling roll back if failure happens. The module ensures that transactions are processed consistently across all active/active servers by using group communication services. All servers make the same decision for a transaction, either submit, or rollback. The module coordinates transaction

Table 1. Incompatibility of lock modes. (Conflicts indicated as X)

	read	update	write
read			X
update		X	X
write	X	X	X

processing, and dispatches any real metadata operation to the basic services module.

Jobs are interleaved by the scheduler, but concurrent operations on the same objects are serialized by a locking service. The locking service provides three lock modes: read, write, and update. Incompatibility of three modes is shown in Table 1. Update lock is designed to improve performance of transactions. A transaction first applies an update lock, without blocking other read requests, then upgrades to write mode only when operations modifying local objects are ready to be submitted. Our design allows disabling the locking service if the parallel file system itself provides other means of locking at the client side or if POSIX file operation semantics are relaxed. Both may lead to further performance improvement.

The file handle space is managed by a dedicated module. Each metadata server allocates and releases file handles independently, but the file handle management must be consistent among all servers of the group. The handle management module is responsible for allocating and releasing handles consistently for all servers and maintaining global state of the handle space.

The membership management module is responsible for maintaining integrity of the service group. Every time when new servers join the group or current members leave the group, the module is notified by the view change messages from Transis demon. The metadata is a global state which must be consistent across all servers at any time, so the service group does not allow multiple partitions. Even if a network partition exists, the active/active metadata server group should only enable one primary group. Any server either belongs to a default primary group, or disable itself, until it rejoins the primary group. If a server crash, it is already disabled automatically. If a server leaves because of a network error, it must also stop responding to any client requests. If a client happens to connect a server not belonging to the primary group because of network partition, the client gets a negative response from the server, thus the client either tries to find other active servers belonging to the primary group, or is notified with an operation failure event. In either case, the metadata keeps consistent under the network partition. After joining the primary group, a server gets current state of metadata from other members of the group, and thus updates its local copy. Since the view change messages are totally ordered to messages from clients, the current state of metadata obtained from other members is exactly consistent to global state, which is just previous to the state updated by the first client request after the point when the new server joins.

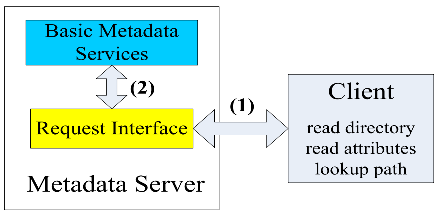


Figure 4. Read requests of Active/Active metadata servers.

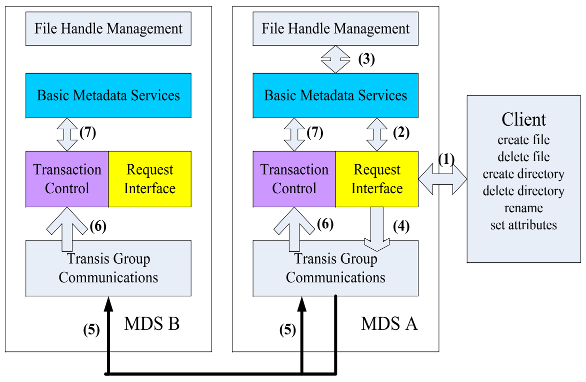


Figure 5. Write requests of Active/Active metadata servers.

Since read requests do not modify metadata, any active server may handle the requests independently and locally (Fig. 4). However, write requests arriving at any metadata server have to be totally ordered by group communication services before submitted to the basic metadata service (Fig. 5). A typical write request is processed by following steps:

(1) Client sends request to the request interface.
(2) The MDS performs necessary checking before actual modification, for example, does same file already exist upon the create file request?
(3) Allocates a new handle if the request is to create a new file or directory.
(4) Requests Transis to broadcast the request.
(5) Transis orders the requests and broadcasts them to all metadata servers, including itself.
(6) Transis notifies Transition Control of the new requests.
(7) Transition Control updates local metadata through the basic metadata services.

5 Experimental Results

To verify above model, a proof-of-concept prototype for active/active metadata servers has been implemented using the Parallel Virtual File System (PVFS) 2 [8] and deployed on the XTORC cluster at Oak Ridge National Laboratory, using up to 4 metadata servers and 32 client nodes in various combinations for functional and performance testing. The computing nodes of the XTORC cluster are IBM IntelliStation M Pro servers. Individual nodes contain a Intel Pentium 2GHz processor with 768MB memory, and a 40GB hard disk. All nodes are connected via Fast Ethernet (100MBit/s full duplex) switches. Although the Fast Ethernet is pretty slow, the network performance will not be the bottleneck of the system, since we only measure read/write performances of metadata, which is very small compared to whole file (less than 1KB in most requests). Federa Core 5 has been installed as the operating system. Transis v1.03 with *fast delivery* protocol is used to provide group communication services. Failures are simulated by unplugging network cables and by forcibly shutting down processes.

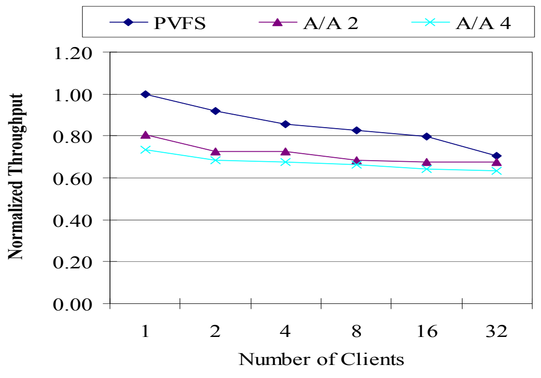
Excessive functional testing revealed correct behavior during normal system operation and in case of single and multiple simultaneous failures. New servers are allowed to join the service group, leave the group voluntarily, and fail, while metadata is maintained consistently at all servers and high availability service is provided to clients.

The proof-of-concept prototype showed a comparable latency and throughput performance. In the experiment, we disabled both the client and server caches to avoid interference. A MPI-based benchmark is used to send concurrent read/write requests from multiple clients. We compared the latency and throughput of original PVFS metadata server and our active/active metadata servers. The results under various configurations are provided for comparison among 2 and 4 Active/Active metadata servers. The latency is measured with blocked requests, and an average value is calculated from 100 requests of each clients. We did not measure the read latency, because read requests are independently handled by each server, and thus theoretically there is no difference of read latency between original PVFS metadata server and our active/active metadata servers. The throughput is measured with unblocked requests. The total requests sent to the servers are $5000 * N$, where N is the number of metadata servers. Each client sends $\frac{5000*N}{n}$ unblocked requests to servers (n is number of clients), and then waits for the completion of all requests. An average throughput is calculated in terms of requests/second.

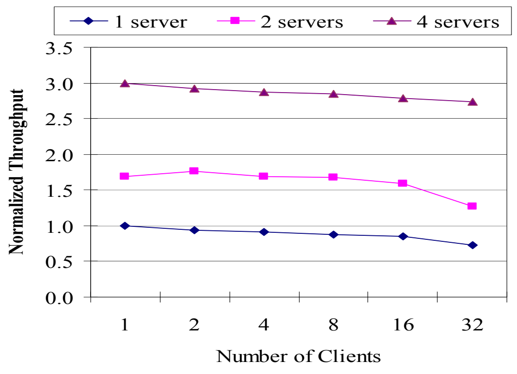
The write request latency overhead (Table 2) is within an acceptable range. We use the performance of a single original PVFS metadata server as a baseline, and data is normalized to the latency of one PVFS server with one client. In the configuration of only two metadata servers with the active/active design, the latency overhead (compared to baseline) mainly comes from processing cost of the Transis group communication service. When the number of metadata servers increases, additional overhead is introduced by the network communication and total order communication algorithm to reach agreement among servers. Although latency increases with the number of metadata servers, the *fast deliver* protocol works well to keep the overall overhead acceptable for any HPC system. The overhead is consistent for both, small and large number of clients. The fast acknowledgment aggressively acknowledges total order messages to reduce the latency of idle

Table 2. Write request latency comparison of single vs. multiple metadata servers. The data is normalized to the latency of one PVFS server with one client.

System	number of clients					
	1	2	4	8	16	32
PVFS 1 server	1.0	2.1	4.7	9.5	20.8	42.7
Active/Active 2 servers	1.3	2.6	5.1	10.0	21.5	43.6
Active/Active 4 servers	1.5	3.0	6.1	11.9	23.3	44.5



(a) Write request



(b) Read request

Figure 6. Throughput comparison of single vs. multiple metadata servers. A/A means Active/Active servers. The data is normalized to the throughput of one PVFS server with one client.

servers when number of clients is small. The protocol is smart enough to hold acknowledgments when the network communication is heavy because more clients are involved.

The write throughput overhead (Fig. 6(a)) reflects similar characteristics. On the contrary, the read throughput (Fig. 6(b)) increases linearly with the number of servers. It is not surprising, because multiple servers can process concurrent read requests simultaneously.

6 Related Work

Past research in high availability for HPC systems primarily focused on the active/standby model. HA-OSCAR [15, 13] and SLURM [3] provide active/standby solutions for the HPC job and resource management system in a warm-standby fashion. PBSPro for the Cray XT3 [2] offers a hot-standby solution with transparent fail-over. Recent research of symmetric active/active replication model [12, 14, 23] uses multiple redundant service nodes running in virtual synchrony [18]. Particularly, the JOSHUA solution [23] for active/active replication HPC job was a precursor for the research presented in this paper.

Previous file systems distribute and replicate metadata and user data to improve availability. XFS file system [4] distributes metadata into multiple managers across the system on a per-file granularity by utilizing a new serverless management scheme. Furthermore, location independence provides high availability by allowing any machine to take over the responsibilities of a failed component after recovering its state from the redundant log-structured storage system. Active/standby model is used in XFS to organize redundant storage system. Frangipani file system [22] uses the large, sparse disk address space of the substrate Petal storage system [16] to simplify its data structures. The data and metadata of Frangipani are stored and managed on top of the virtual address space provided by Petal, similar to traditional file systems on top of the address space of hard disks, but the real data is physically distributed to multiple Petal storage servers. High availability of both user data and metadata is provided by a replication based redundancy scheme called chained declustering of the Petal system.

Various research efforts in file systems have shown that total-ordering can be used to provide high availability. Deceit file system [21] behaves like a plain NFS server. The deceit servers are interchangeable and collectively provide the illusion of a single server to any clients. It uses the ISIS [7] distributed programming environment for all totally ordered communication and process group management. Non-volatile replicas of each file are stored on a subset of the file servers. A practical replication algorithm [9] is designed to tolerate Byzantine faults. The algorithm works in asynchronous environments to improve the response time and reduce the latency. A Byzantine-fault-tolerant NFS service using the algorithm is implemented and compared with a standard non-replicated NFS.

7 Conclusions

We presented our recent research in active/active metadata servers as a generic approach for highly available cluster storage systems. Our concept provides a virtually synchronous environment for high availability without any interruption of service and without any loss of state. It guarantees the safety of global state updating by utilizing group communication services and total order broadcasting.

We used a fast delivery protocol to reduce the latency of ordering messages. The protocol optimizes the total ordering process by waiting for messages only from a subset of the machines in the group. The protocol performances well for both idle and active servers.

We presented functional and performance test results with comprehensive experiments under various system configurations. Our results show that for write requests, the overhead of latency and throughput increases with the number of servers, but is still acceptable for typical distributed storage systems. The throughput of read requests increases linearly with the number of servers. We also provide a theoretical availability analysis. The experimental results show that high availability of metadata servers can be achieved without interruption and with an acceptable performance trade-off using the active/active metadata server solution.

Acknowledgements

This research was supported by the U.S. National Science Foundation under Grant Nos. CNS-0617528 and CNS-0720617 and by the TTU office of research. It was also partially sponsored by the Mathematical, Information,and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed in part at the Oak Ridge National Laboratory,which is managed by UT-Battelle, LLC under Contract No. DEAC05-00OR22725. This work was also partially sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory.

References

[1] Lustre: A scalable, high-performance file system. In *Cluster File Systems, Inc.*

[2] PBSPro job management system for the cray XT3 at Altair Engineering, Inc. *http://www.altair.com/pdf/PBSPro Cray.pdf*.

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

[4] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proc. Fifteenth Symposium on Operating Systems Principles*, pages 109–126, December 1995.

[5] R. Baldoni, S. Cimmino, and C. Marchetti. Total order communications: A practical analysis. *Lecture Notes in Computer Science: Dependable Computing*, 3463:38–54, 2005.

[6] K. Berket, D. A. Agarwal, P. M. Melliar-Smith, and L. E. Moser. Overview of the InterGroup protocols. *Lecture Notes in Computer Science: Proceedings of ICCS 2001*, 2073:316–325, 2001.

[7] K. P. Birman and R. van Renesse. Reliable distributed computing with the ISIS toolkit. *IEEE Computer Society Press*, 1993.

[8] P. H. Carns, Walter B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proc. 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, October 2000.

[9] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proc of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, February 1999.

[10] X. Defago, A. Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 2004.

[11] D. Dolev and D. Malki. The transis approach to high availability cluster communication. *Communications of the ACM*, 1996.

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

[13] 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*, 2006.

[14] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. He. Active/active replication for highly available HPC system services. In *Proceedings of* 1^{st} *International Conference on Availability, Reliability and Security (ARES) 2006*, pages 639–645, Vienna, Austria, Apr. 20-22, 2006.

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

[16] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proc. 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, October 1996.

[17] S. Mishra and L. Wu. An evaluation of flow control in group communication. *IEEE/ACM Transactions on Networking*, 6(5):571–587, 1998.

[18] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. *Proc. of DCS*, 1994.

[19] L. Ou, X. He, C. Engelmann, and S. Scott. A fast delivery protocol for total order broadcasting. In *Proceedings of 16th International Conference on computer Communications and Networks (ICCCN 2007)*, Honolulu, Hawaii, August 2007.

[20] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. *Proceedings of USENIX Annual Technical Conference*, pages 41–54, 2000.

[21] A. Siegel, K. P. Birman, and K. Marzullo. Deceit: A flexible distributed file system. Technical Report TR89-1042, Cornell University, 1989.

[22] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proc. Symposium on Operating Systems Principles*, 1997.

[23] K. Uhlemann, C. Engelmann, and S. L. Scott. JOSHUA: Symmetric active/active replication for highly available HPC job and resource management. In *Proceedings of IEEE International Conference on Cluster Computing*, September 2006.