NFSv4 as the Building Block for Fault Tolerant Applications

Alexandros Batsakis and Randal Burns
Department of Computer Science
Johns Hopkins University

Abstract

We propose extensions to the NFSv4 client architecture that provide recovery services, checkpointing and logging to parallel applications. Fault-tolerance relies on NFSv4 clients forming *client clusters* that share delegation state and transfer data among their local caches. We contend that parallel extensions to NFSv4 should not be limited to file system technologies, such as parallelizing or virtualizing data paths. Rather, NFSv4 should consider semantic extensions that fulfill the recoverability requirements of parallel applications.

1 Overview

Fault tolerance in parallel environments is a subject that has been studied extensively in theory, but practical implementations are relatively scarce. Clusters of commodity computers are commonly used to implement cost-effective, high-performance systems. However, the low-cost, commodity nature of clusters leads to their principal drawback – numerous low cost workstations are prone to unexpected failures. It becomes critical to provide some degree of fault tolerance to the high performance applications running on these clusters. We propose a model in which the file system, specifically a slightly modified version of NFSv4 [8, 16], provides built-in support for fault tolerant applications. Elements such as data availability, output commit, repeatable reads and recoverability, formerly addressed in the application domain, are now delegated to the file system.

The delegation of fault-tolerant capabilities to the file system simplifies high performance applications, because developers no longer have to build their own complex mechanisms. Applications use the asynchronous file system interfaces for both file sharing and fault tolerance. The file system manages persistence and communication for checkpointing and logging. Applications still have control over their own fault tolerance semantics, *e.g.* when and what to log.

Current file systems do not meet the requirements of high performance parallel applications. These applications usually communicate through both message passing and file sharing. Conventional file servers, including the NFSv4 server, do not support file sharing efficiently for parallel applications. Conventional rollback recovery protocols incur substantial overhead when used for applications in which nodes communicate through file sharing. Furthermore, current parallel file systems do not assist recoverability since they do not provide support for repeatable reads and fast output commits.

Most previous attempts to incorporate fault tolerance into parallel environments focus on message passing interfaces, such as MPI [17] or PVM [9]. These techniques offer fault tolerance that is transparent to applications, but they apply only to a specific message passing interface.

The idea to enhance NFS with fault tolerance support is not new [11, 13, 2]. Previous proposals focus on data availability by replicating files to many servers. While data availability is an important aspect of fault tolerance, applications have further requirements. These include message logging, checkpointing, repeatable reads and output commit. Currently, NFS related methods provide fault tolerance support for file system data only and, therefore, fail to satisfy the needs of applications.

Our decision to provide fault tolerance support in the file system intends primarily to simplify the development of high performance parallel applications by hiding low-level fault tolerance characteristics from the developer, such as the exact logging location or support for repeatable reads. Also, the choice of the ubiquitous NFS protocol as the underlying file system, which can be found in almost every popular operating system, enables recoverability among heterogeneous clients with different operating systems and architectures.

The final contribution of our proposal is the modification of the NFSv4 client to support file sharing more efficiently. NFSv4 introduce new capabilities for more aggressive client side caching in order to improve performance over high latency networks. Specifically, file delegation transfers all responsibility for locking, read and write operations to the client, requiring no further interaction with the server for the duration of the delegation. We enhance the concept of delegation by allowing a file to be delegated to a cluster of clients. Clients exchange the delegation interactively to support read and write sharing without contacting the server. We refer to this enhanced version of NFSv4 delegation as *group delegation*. This feature particularly benefits high performance, parallel applications.

2 Related Work

We include a brief treatment of NFS recoverability and some related fault-tolerance techniques in parallel environments. Elnozahy *et al* [7] offer a thorough review of rollback recovery protocols.

Some research augments message passing interfaces with fault-tolerance capabilities [3, 18, 15, 10], which preserves application interfaces and results in application transparent fault tolerance. These solutions apply to a specific interprocess communication model only and do not address I/O or other communication paths. Application level toolkits, such as Manetho [6], share the same disadvantage.

Other methods require modification or recompilation of the application either by offering a library that provides check-pointing, restart and replication capabilities [14] or by using a special-purpose compiler that produces "fault-tolerant" code [4]. Modification and recompilation is burdensome and the use of the systems in parallel environments is not straightforward.

Fault-tolerant NFS systems address the availability and recoverability of file system data only, and, do not address application concerns, such as checkpointing and logging. FT-NFS [13] uses a number of backup NFS servers and replicates the file state among this group. Expand [11] virtualizes a set of NFS server as a unique parallel file system with a common namespace. HA-NFS [2] employs redundant hardware, dual-ported disks, and mirroring to take over on a hot-standby server in case the primary fails.

The extensions we propose are orthogonal to fault-tolerance in NFS and can be used to improve existing fault tolerant systems. We build application fault tolerance within the NFSv4 protocol and rely on NFSv4 servers to manage the availability and persistence of file system data. The extensions to NFSv4 allow rollback recovery systems to use the file system efficiently and extend recoverability guarantees to file I/O.

Alvisi *et al* [1] describe a theoretical model with recovery properties similar to the proposed system. Their approach uses a client/server file system to share logging and checkpointing information. However, they use message passing protocols for volatile replication and client-to-client data sharing outside the file system. Message passing avoids the overhead of synchronous I/O and reduce server interactions. In contrast, our system builds these benefits into the file system, making recovery services efficient and application transparent.

3 NFSv4 client modifications

In our attempt to provide efficient fault tolerance support in the NFS file system, some modifications to the NFSv4 client are necessary. No server-side modifications are required.

The first change is group delegation. Current NFS implementations (including NFSv4) require the client possessing the most recent version of the file to write it back to the server before it can be sent to the next client. In order to improve performance of write sharing, we propose a enhanced version of delegation (group delegation) in which clients share files directly with other clients without additional interaction with the server.

Group-delegation extends the current delegation model by allowing clients that modify the delegated file to perform a COMMIT operation without contacting the server. Instead of committing to the server's stable storage, clients copy a file to a configurable number of remote memories. We refer to this operation as fast commit, because updates are logged immediately to remote memories and will be written to stable storage at a later time. Fast commit is an application of family-based logging protocols (FBL) [1]. FBL protocol implements stable storage by replicating data to the volatile memory of nodes and guarantees that the file can be recovered at any time unless multiple concurrent failures occur.

As delegated files are now distributed to multiple client memories, we need an effective way to locate which client has the desired instance of the file. In order to address this issue, we use a form of cooperative caching [5] between the collaborating NFS clients. The cooperative caching algorithm coordinates the contents of the caches and allows requests to be satisfied by the memory of remote clients, if possible.

4 NFSv4 in rollback recovery protocols

By using fault-tolerance extensions to NFSv4, applications and user-level toolkits avoid the obligation to maintain persistence and communication for checkpointing and logging. NFS hides the details of the inter-node interactions, allowing applications to focus on the high-level semantics of fault tolerance (*e.g.* when and what to log, checkpoint frequency and coordination).

Fault-tolerant extensions to NFSv4 solve the performance problems of storing application state in shared file systems and simplify the implementation and management of replication and persistence. Consider a client that stores state information in local storage. When such a client fails, its disk becomes unreachable and other clients cannot access data to recover application state. Rollback recovery systems [1] avoid this problem by replicating state outside the file system through message passing. Such an approach requires complex protocols to create and invalidate replicas. As an alternative to replication through message passing, clients can store all state to a shared file system. However, client/server interactions are expensive relative to I/O to a local disk or messaging within a cluster. Our extensions to NFSv4 allow the shared file system to encapsulate the advanced features of rollback recovery protocols. Group delegation, fast commit, and cooperative caching solve the performance problems of shared file systems. Shared data are made persistent seamlessly by asynchronous writes from client caches. NFS clients store shared data in their local memory, in a number of remote peer memories to guarantee availability across node failures, and eventually to the file server to guarantee persistence.

Shared message logging is the first fault tolerance function implemented by our NFSv4 client extensions. One approach to message logging has all clients sharing a single log, appending causal events to a file. In this approach, group delegation serializes writes. Cooperative caching allows nodes to exchange log data between local memories. Fast commit replicates data for availability. All together, the features allow clients to share a single log file without server interactions, except for asynchronous write-back to make modified file data persistent. Other specific enhancements, such as efficient append writing modes and truncate from the front of a file, fit into this logging model, but are not the focus of this proposal.

NFSv4 enhancements also benefit per-node message logging architectures in which each node writes state changing events to its own log in the shared file system. This architecture is an alternative to shared logging. After a node failure, other nodes access a log to recover the state of the failed application. In this model, performance benefits come from fast commit. Alvisi *et al* [1] describe a file-sharing protocol that eliminates synchronous output commits and write-backs. Their approach builds a middleware service that replicates determinants (causal events) in user space. Furthermore, middleware has to manage replicas, including invalidation and writing data to persistent storage. We observe that this middleware is managing data outside of the file system to fix performance problems. Implementing similar services in NFSv4 hides the complexity from applications and manages persistence seamlessly.

Checkpointing is an important aspect of fault tolerant systems because it reduces the size of the log and makes recovery faster. In our model, applications are still responsible for the checkpointing semantics. Thus, checkpoint coordination and frequency must be decided by the application. Efficient data sharing benefits checkpointing in the same manner as logging.

An important characteristic of our NFS-based solution is the elimination of read logging. To make I/O recoverable, applications must log the contents of every file-read to stable storage. The concern is that file data has been changed. File I/O is a causal event and the original data must be available during recovery in order to reconstruct application state. We eliminate the need for read logging by supporting repeatable reads. In one approach, we propose to run the NFSv4 server on a versioning file system. The NFSv4 clients obtain a version identifier for each read. Instead of logging read data, the client logs the version identifier. During recovery, the client presents the identifier to the NFSv4 server and reads the original versions. Recent work encapsulates versioning functionality in a disk file system and make versioning transparent [12], so that repeatable reads require no changes to the NFSv4 server. An alternative builds repeatable reads on standard file systems, by having the NFSv4 client create separate files in a copy-on-write manner. This approach is preferable to read-logging, because versions are created at write and hardened asynchronously, as opposed to read-logging which logs data synchronously. While this approach works on any file system, it does not have the data reduction benefits of versioning file systems.

5 Conclusions

We propose a model in which the function and complex optimizations for fault tolerance support, formerly found in the application domain, are implemented in the file system. Specifically, our model is based on NFSv4 to provide persistence and communication for logging and checkpointing. We modify NFSv4 client by adding support for group delegation, fast commit and cooperative caching. The incorporation of fault tolerance functionality into the fi le system greatly reduces complexity and development time, while supporting heterogeneity and fi le sharing.

Acknowledgments

We wish to credit and thank Lorenzo Alvisi for early discussions that helped defi ne the fi le system requirements of rollback recovery protocols. We also thank Charles Lever for his help with NFSv4 concepts and distributions and Zachary Peterson for advising us on versioning strategies.

References

[1] L. Alvisi, S. Rao, and H. M. Vin. Low-overhead protocols for fault-tolerant file sharing. In *Proceedings of the International Conference on Distributed Computing Systems*, 1998.

[2] A. Bhide, E. N. Elnozahy, and S. P. Morgan. A highly available network file server. In *USENIX Winter Technical Conference*, 1991.

[3] J. Casas, D. Clark, P. Galbiati, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MIST: PVM with transparent migration and checkpointing, 1995. Available at http://www.cse.ogi.edu/DISC/projects/mist.

[4] S.-E. Choi and S. J. Deitz. Compiler support for automatic checkpointing. In *Proceedings of the International Symposium on High Performance Computing Systems and Applications*, 2002.

[5] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the Operating Systems Design and Implementation Conference*, 1994.

[6] E. N. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback, and fast output commit. *IEEE Transactions on Computers*, 41(5), 1992.

[7] M. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, 1996.

[8] B. Pawlowski *et al*. The NFS version 4 protocol. *Proceedings of the System Administration and Networking Conference*, 2000.

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

[10] J. Leon, A. L. Fisher, and P. Steenkiste. Fail-safe PVM: A Portable Package for Distributed Programming with Transparent Recovery. Technical Report CMU-CS-93-124, School of Computer Science, Carnegie Mellon University, 1993.

[11] K. Marzullo and F. Schmuck. Supplying high availability with a standard network file system. In *Proceedings of the International Conference on Distributed Computing Systems*, 1988.

[12] Z. Peterson and R. Burns. Ext3cow: The design, implementation, and analysis of metadata for a time-shifting file system. Technical Report HSSL-2003-03, Hopkins Storage Systems Lab, Department of Computer Science, Johns Hopkins University, 2003.

[13] N. Peyrouze and G. Muller. FT-NFS: An efficient fault-tolerant NFS server designed for off-the-shelf workstations. In *Proceedings of the Symposium on Fault-Tolerant Computing*, 1996.

[14] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under UNIX. Technical Report UT-CS-94-242, Department of Computer Science, University of Texas, 1994.

[15] S. Rao, L. Alvisi, and H. M. Vin. Egida: An extensible toolkit for low-overhead fault-tolerance. In *Proceedings of the Symposium on Fault-Tolerant Computing*, 1999.

[16] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. In *Proceedings of the Summer USENIX Conference*, 1985.

[17] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI – The Complete Reference*. MIT Press, 1996.

[18] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium* , 1996.