SARCNFS: Self-Adaptive Redundancy Clustered NAS File System

CaiBin, Changsheng XIE, RenJin, FaLing YI

National Storage System Laboratory, Department of Computer Science,
Huazhong University of Science and Technology. Wuhan, P.R China, 430074
Email: hust_caibin@sohu.com

Abstract. In this paper, we describe the design and implementation of SARCNFS File System for network-attached clustered storage system. SARCNFS stripes the data and metadata among multiple NAS nodes, and provides file redundancy scheme and synchronization mechanism for distributed RAID5. SARCNFS uses a self-adaptive redundancy scheme for file data accesses that uses RAID5-level for large writes, and RAID1-level for small write so as to dynamically provide flexible switch between RAID1 and RAID5 to provide the best performance. In addition, SARCNFS proposed a simple distributed locking mechanism that uses RAID5-level for full stripe writes, and RAID1-level for temporary storing data from partial stripe updates. As a result, low response latency, high performance and strong reliability are achieved.

1 Introduction

The distributed RAID concept was proposed by Stonebraker and Schloss [1]. Examples of early-distributed RAID systems include Swift/RAID [2], Petal [3] and Tertiary Disk [4]. In distributed RAID5 implementation, there are two significant issues:

1. *Small writes accesses latency* is high because of the extra reads.
2. *Synchronization problem* is presented in cluster environment for simultaneous writes.

In this paper, we design and implement a file system for clustered NAS storage environment called SARCNFS that distributes user data and metadata among multiple NAS nodes, similar to xFS [5], [6], Zebra [7], Swarm [8], Frangipani [9], and PVFS[10]; meanwhile, we employ three redundancy schemes in SARCNFS file system: the first scheme is a striped, file-mirroring scheme like RAID1. In this scheme, the user data are stored twice by file system; the second scheme is a RAID5-like scheme, which uses parity-based partial redundancy; finally, we adopt a self-adaptive scheme that uses RAID5-level for large writes, and RAID1-level for small write so as to dynamically provide flexible switch between RAID1 and RAID5 to provide the minimal performance degradation. In addition, we proposed a simple locking mecha-

nism that uses RAID5-level for full stripe writes, and RAID1-level for temporary storing data from partial stripe updates. Since partial stripe writes use the RAID1 scheme, we avoid the synchronization necessary in the RAID5 scheme for this access pattern to addresses the synchronization problem in distributed RAID5 environment.

2 ARCNFS Clustered NAS File System

2.1 System Architecture Overview

SARCNFS is designed as a server-less system [5], [6], in which multiple NAS nodes deal with storage of file data and manage metadata. Each SARCNFS file is striped across a set of nodes in order to facilitate parallel access. This set of modes is selected in a random way, and the data are distributed with a round-robin police using the set of nodes. The layout of SARCNFS is shown in Fig 1.

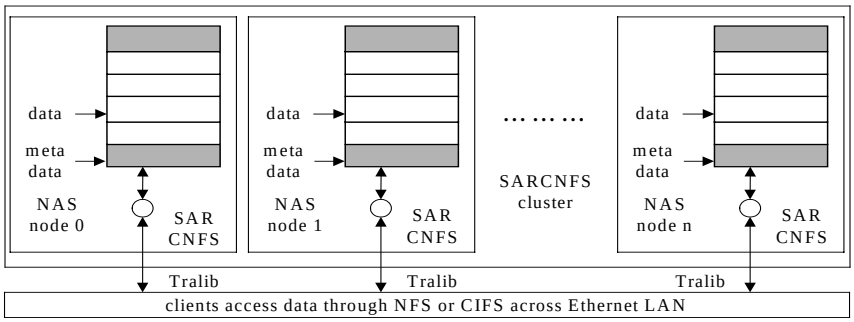

Fig. 1. Physical layout of SARCNFS in Clustered NAS Environment

The specifics of a given file distribution are described with three metadata parameters: base node number, number of nodes, and stripe size. These parameters, together with ordering of the modes for the file system, allow the file distribution to be completely located. To access SARCNFS file data, the client first obtains the metadata of the SARCNFS file on the NAS nodes, and then, the client sends requests directly to the NAS nodes storing the relevant portions of the file.

2.2 Self-Adaptive Redundancy Scheme

In RAID1 scheme implementation in SARCNFS, each NAS node stores two files per client file. One file is the data file used to store the data, just like the case in PVFS. The other file is the redundancy data file used to store redundancy. The contents of a redundancy block are identical to the contents of the corresponding data block. As a

result, the RAID1 scheme in SARCNFS has ability to utilize all the available bandwidth on a read operation to provide parallel read performance. On a write, all the nodes must be written twice.

The distributed RAID5 scheme in our SARCNFS also has redundancy file on each node in addition to the data file, like the RAID1 scheme. In the distributed RAID5 scheme, however, these files contain the parity data for specific portions of the data files. On a write operation, the client checks the offset and size to judge if any stripes are about to be updated partially. There can be at most two partially updated stripes in a given write operation. The client reads the data in the partial stripes and also the corresponding parity region, and then, it computes the parity for the partial and full stripes, and writes out the new data and new parity.

In the Adaptive Redundancy scheme, the level of redundancy is selected depended on the following rule: every client write access is broken down into three portions: (1) a partial stripe write at the start (2) a portion that updates an integral number of full stripes (3) a trailing partial write. Depending on data alignment and size, portions can be empty. For the portions of the write that updates full stripes, we compute and write the parity, just like in the RAID5 case. For the portions involving partial stripe writes, we write the data and redundancy like in the RAID1 case, except that the updated blocks are written to an overflow region on the nodes. The blocks cannot be updated in place because the old blocks are needed to reconstruct the data in the stripe in the event of a crash. When a file is read, the nodes return the latest copy of the data, which could be in the overflow region.

2.3 Distributed Locking Mechanism

We implemented a simple distributed locking mechanism to ensure that two clients writing concurrently to disjoint portions of the same stripe do not leave the parity for the stripe in an inconsistent state in distributed RAID5 scheme. When a node receives a read request for a parity block, it knows that a partial stripe update is taking place. If there are no outstanding writes for the stripe, the node sets a lock to indicate that a partial stripe update in progress for that stripe. It then returns the data requested by the read. Subsequent read requests for the same parity block are put on a queue associated with the lock. When the node receives a write request for a parity block, it writes the data to the parity file, and then checks if there are any blocked read requests waiting on the block. If there are no blocked requests, it deletes the lock; otherwise it wakes up the first blocked request on the queue. The client checks the offset and size of a write to determine the number of partial stripe writes to be performed. If there are two partial stripes involved, the client serializes the reads for the parity blocks, waiting for the read for the first stripe to complete before issuing the read for the last stripe. This ordering of reads avoids deadlocks in the locking protocol.

The RAID1 scheme does not require any additional synchronization. The same is true of the Adaptive Redundancy scheme since it uses mirroring for partial stripe writes.

3 Conclusion

SARCNFS uses a self-adaptive redundancy scheme for file data access that uses a combination of RAID5 and RAID1 writes to store data. Full stripe writes use the RAID5 scheme. RAID1 is used to temporarily store data from partial stripe updates. As part of the SARCNFS implementation, we have proposed a simple locking mechanism that addresses the consistency problem of distributed implementations of RAID5 redundancy. Since partial stripe writes use the RAID1 scheme, we avoid the synchronization necessary in the RAID5 scheme for this access pattern. In addition, unlike many other clustered file system, our SARCNFS file system is not dependent on client modifications since it is full compatible with standard distributed file systems such as NFS and CIFS. As a result, SARCNFS dynamically provide the high performance, low latency and the strong reliability.

References

1. M. Stonebraker and G. Schloss.: Distributed RAID:-A New Multiple Copy Algorithm. In sixth IEEE Conference Data Engineer, IEEE Press, pages-430-437, 1990.
2. D. D. E. Long, B. Montague, and L.-F. Cabrera.: Swift/RAID: A Distributed RAID System. Computing Systems, 7(3), Summer, 1994.
3. E. K. Lee and C. A. Thekkath.: Petal: Distributed Virtual Disks. In Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, pages 84-92, Cambridge, MA, 1996.
4. N. Talagala, S. Asami, D. Paucrson, and K. Lutz.: Tertiary Disk: Large Scale Distributed Storage. Technical Reports, no UCB/CSD-98-989, University of California, at Berkeley, 1998.
5. T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Young.: Serverless Network File Systems. ACM Transactions on Computer Systems, Feb. 1996.
6. T. Anderson, M. Dahlin, J. Neefe, D. patterson, D. Roselli, and R. Wang.: Serverless Network File Systems. In Proceedings of the Symposium on Operating System Principles, pages 109–126, Dec. 1995.
7. J. Hartman and J. Ousterhout.: The Zebra Striped Network File System. ACM Transactions on Computer Systems, Aug. 1995.
8. J. H. Hartman, I. Murdock, and T. Spalink.: The Swarm Scalable Storage System. Proceedings of the 19th International Conference on Distributed Computing Systems, May. 1999.
9. C. A. Thekkath, T. Mann, and E. K. Lee.: Frangipani: A Scalable Distributed File system. In Proceedings of the Symposium on Operating System Principles, pages 224-237, 1997.
10. Philip H. Carns, Walter B. Ligon III, Robert B. Ross, Rajeev Thakur.: PVFS: A Parallel File System for Linux Clusters. In Proceeding of the Extreme Linux Track: 4th USENIX Annual Linux Showcase and Conference, Oct. 2000.