A Highly Available Cluster Storage System Using Scavenging*

Xubin (Ben) He, Li Ou
Department of Electrical and Computer Engineering
Tennessee Technological University, Cookeville, TN 38505, USA
{hexb,lou21}@tntech.edu

Stephen L. Scott, Christian Engelmann
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
{scottsl,engelmannc}@ornl.gov

Abstract

Highly available data storage for high-performance computing is becoming increasingly more critical as high-end computing systems scale up in size and storage systems are developed around network-centered architectures. A promising solution is to harness the collective storage potential of individual workstations much as we harness idle CPU cycles due to the excellent price/performance ratio and low storage usage of most commodity workstations. For such a storage system, metadata consistency is a key issue assuring storage system availability as well as data reliability. In this paper, we present a decentralized metadata management scheme that improves storage availability without sacrificing performance.

1 Introduction

Data intensive scientific applications, such as simulations for climate, fusion or biology research, constitute an essential part of high performance computing. The increasing demand to deploy distributed storage systems has led to high performance storage [13], wide area mass storage [21, 22], and cluster-based storage [10, 11, 14, 18, 23]. These high-end storage solutions offer an excellent performance and sufficient parallel support, while they also have issues in increasing administration costs, specialized software, and central points of failures and control.

To address this issue, Vazhkudai [20] proposes to harness the collective storage potential of individual workstations in the same way we harness the idle CPU cycles for distributed computing. This solution constructs aggregate storage from numerous commodity workstations as shown in figure 1. Individual workstation users volunteer to contribute an amount of their storage space for a certain duration of time to a manager (server) using a scavenging process. Each benefactor workstation participates by registering itself with the manager and by constantly updating the manager with keep-alive messages using a soft-state registration protocol. Instead of a centralized single manager server, this system employs a group of servers that collectively takes on the responsibility of managing the system. This group handles benefactor registrations, client requests, and maintains different kinds of metadata including registration information, directory metadata, mapping information, and so on. The group itself can use various replication strategies to provide high-availability. This scavenging storage system provides a solution for scalable, reliable and highly available distributed storage. However, certain issues regarding metadata consistency [1, 4, 9, 12] and handling Byzantine failures [6] need to be addressed efficiently.

Any I/O request handled by a computer system can be classified into user data request and metadata request. A study by Roselli et al. [17] shows that requests targeting at the metadata can account for up to 83 percent of the total amount of I/O requests in some typical applications. For a scavenged storage system [20], where multiple servers are employed to manage all metadata, including file/directory attributes, data block addresses, consistency control, access control, and registration information, the efficiency of meta-

*This work was supported in part by Research Office under Faculty Research Award and Center for Manufacturing Research at Tennessee Technological University, ORAU under the Ralph E. Powe Junior Faculty Enhancement Award, and the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

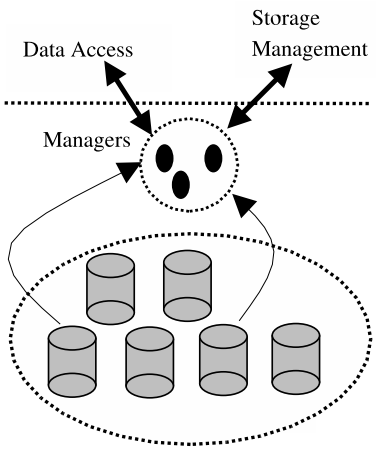

Figure 1. Architecture for a Scavenging Storage System

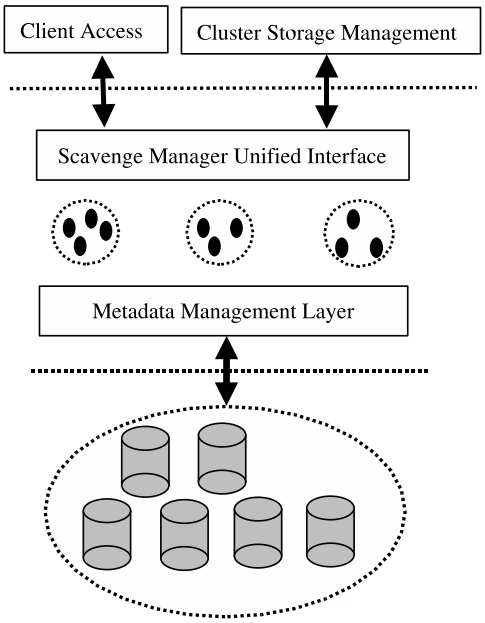

Figure 2. Distributed Metadata Management for a Scavenging Storage System

data management is critical for its overall performance.

With this paper, we continue the effort to develop storage solutions that can take advantage of scavenging techniques. We propose a decentralized, scalable, highly available metadata management scheme with multiple manager servers and Bloom filter algorithms [5].

2 Highly Available Metadata Management

In our design, we focus on a generic heterogeneous cluster consisting of a number of commodity workstations connected by a TCP/IP network. All workstations form a scavenged system as shown in figure 2. They come in all flavors ranging from operating system diversity to machine characteristics. Each node has its own storage device(s) while there are no functional differences among all participating nodes. The role of clients, storage scavengers, and scavenge managers can be carried out by any node.

Regarding the scavenged storage system design, Vazhkudai [20] has highlighted many positioning aspects, such as soft-state registration, space reclaim, relocation, availability, and so on. In this study, we concentrate on the necessary distributed metadata management for multiple manager servers in multiple domains.

To avoid single point of failure and provide fault tolerance as well as high availability, we employ a group of scavenge manager servers to collectively take on the responsibility of managing the system and its respective state information. There are a number of problems that must be considered when having more than one manager with data coherency being one of the more difficult issues to resolve. We envision four potential manager schemes.

In the first scheme, all managers are equivalent peers. In this model, each manager maintains its own local copy of the metadata representing the entire scavenged storage state. All managers are organized using a distributed peer-to-peer approach [7, 11]. Since coherency is guaranteed, any manager may service a request without consulting the others. While this is relatively straightforward in the case of read accesses, it becomes significantly more complex in the case of updates or writes. To address inconsistency problems of potential conflicting accesses, we propose the use of a global ordering technique already known from distributed databases [3]. Distributed transactions alter tables on several servers simultaneously while allowing concurrent accesses. To ensure data consistency, all transactions are globally ordered and processed in this order at all servers. This order can be determined using timestamps [19] or more complex message numbering algorithms [7]. We apply such an ordering technique for transaction operations to our manager group, thus providing coherency for simultaneous accesses. The read, update and write operations are ordered in a similar manner. Essentially a group manager state change is performed using an atomic transaction, which results in a modified local state that is simultaneously propagated to the other managers using any number of techniques including point-to-point and multicast. A significant drawback of this technique is that the network traffic necessary to maintain data coherency may negatively impact performance.

A second strategy is to actively replicate changes only to a subset of the manager group in an active/hot-standby

fashion [15], leaving all other manager members to retrieve a copy of the new state on demand or scheduled at a time when the network is less loaded. While this system will gain in availability as the number of active managers increases, it will likewise jeopardize performance. This tradeoff may be "tuned" by adjusting the number of active nodes. Ultimately, consistency must be enforced when any update/write transaction is performed so that any manager may service a request. In this scheme, when an active server fails any hot-standby server may become active as needed to satisfy a specific level of high availability. An added benefit of peer-to-peer active/hot-standby schemes is that read performance improves, since read requests may be serviced by a nearby hot-standby server that acts as a local cache.

A third approach is, instead of replicating all of the information, we partition the entire system into several subdomains. In this scheme, each manager (or group of managers) is responsible for only its respective sub-domain. This results in a hierarchical manager structure with a master manager group as root. Furthermore, in this scheme, no single manager has the knowledge of the entire system. This approach is completely orthogonal to the prior two approaches in this manner. A failure in this method will then be confined to a sub-domain and high availability will be addressed via the master manager group.

A fourth solution is to nominate a leading manager (leader) that has a complete view of all metadata, while several help managers (helpers) have only the metadata of their respective sub-domain. To improve availability, the leader may be replicated via mirroring. Here a failure will result in a mirror being activated as the active leader manager (failover). Each helper must maintain contact with the leader in some manner. Should a helper fail, a new one may be easily added and its data reconstructed via the leader.

All of the aforementioned schemes provide fault tolerance using replication and some level of high availability. However, there is always a trade-off between performance and availability. Our work is currently exploring the first solution but we plan to spend time considering all of the above alternatives and any others we may discover.

Group communication services [2, 7, 16] are critical to achieve high availability and reliability. For the three discussed management schemes, we may use different algorithms and network topologies. In this study, we concentrate on the peer-to-peer distributed control [7] for the first solution, where each manager has a complete and consistent copy of all the metadata information.

We use group communication services, such as Reliable Broadcast and Atomic Broadcast, to simplify the maintenance of consistently replicated state. Atomic transaction operations guarantee metadata integrity despite random communication delays, failures, and recoveries. The manager group consistently maintains a global state using asynchronous peer-to-peer communication.

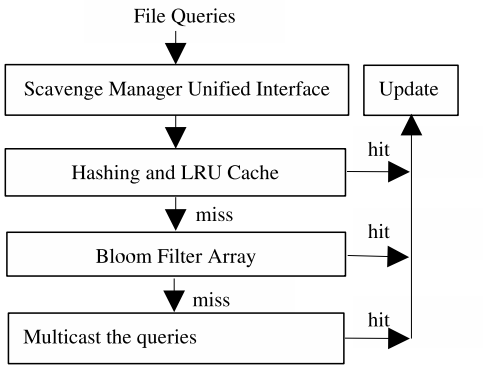

Figure 3. Scheme of Bloom Filter Array

3 Metadata Management using Bloom Filter Arrays

A Bloom filter is a fast and efficient method for representing a set $A = \{a1, a2, ..., an\}$ of n elements to support membership queries. It was invented by Burton Bloom in 1970 [5] and was proposed for use in the web context by Marais and Bharat [15] as a mechanism for identifying which pages have associated comments stored within a web server, and thereafter used by many researchers [23]. The basic idea of a Bloom filter is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions, $\{h1, h2, ..., hn\}$, each with the range $\{1, ...m\}$. For each element a, the bits at positions $h1(a), h2(a), ..., hk(a)$ in v are set to 1. To find out whether a given element x belongs to the set A, one simply tests whether the k bits addressed by $hi(x)$, are all set to 1.

A Bloom filter is very efficient since it reduces the memory requirement of representing a set of elements at the nominal cost of a very small probability of false hits [8].

We use an array of Bloom filters (BFA) on each scavenge manager to efficiently manage the metadata. Each manager builds a bloom filter that represents all files whose metadata is stored locally and the replicates this filter to all the other scavenge managers. A manager stores all Bloom filters including the replicas of the Bloom filters from all the other managers in an array. When a client has a meta data request through the unified interface, the client's request will directed to one of the managers and this manager will perform the membership query against this array.

To speed up the search, a LRU cache can be used to explore the locality before the Bloom filter array. In a typical workload, many files are written again and again, which means same metadata information may be accessed frequently. The design of this scheme is shown in figure 3.

4 Conclusions

We have discussed the distributed metadata management for a scavenging storage system that harnesses the collective storage potential of individual workstations. We have presented four varying solutions that use multiple metadata managers to provide high availability. We also proposed a decentralized metadata management scheme using Bloom filters to speed up the metadata search. It is our desire to use this workshop paper and talk to open a dialogue regarding the various techniques outlined in this paper in order to solicit feedback from both the high-availability and the storage community.

References

[1] M. Ahamad and R. Kordale. Scalable consistency protocols for distributed services. *IEEE Transactions on Parallel and Distributed Systems*, 10:888, 1999.

[2] 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.

[3] P. Bernstein, V. Hadzilacos, and N. Goldman. Concurrency control and recovery in database systems. *Addison Wesley*, 1987.

[4] E. Bilir, R. Dickson, Y. Hu, M. Plakal, D. Sorin, M. Hill, and D. Wood. Multicast Snooping: A new coherence method using a multicast address network. *Proceedings of ICSA*, pages 294–304, 1999.

[5] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[6] M. Castrol and B. Liskov. Practical byzantine fault tolerance. *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

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

[8] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[9] M. Feeley et al. Implementing global memory management in a workstation cluster. *Proceedings of perating Systems Principles*, 1999.

[10] Gabber et al. StarFish: Highly-available block storage. *Proceedings of the FREENIX track of the 2003 USENIX Annual Technical Conference*, pages 151–163, 2003.

[11] X. He, M. Zhang, and Q. Yang. DRALIC: A peer-to-peer storage architecture. *Proceedings of PDPTA*, II:908–913, 2001.

[12] X. He, M. Zhang, and Q. Yang. Stics: Scsi-to-ip cache for storage area networks. *Parallel and Distributed Computing*, 64(9):1069–1085, 2004.

[13] H. Hulen et al. Storage area networks and the high performance storage system. *Proceedings of Tenth NASA Goddard Conference on Mass Storage Systems*, 2002.

[14] M. Lauria, K. Bell, and A. Chien. A high-performance cluster storage server. *Proceedings of HPDC-10*, pages 311–320, 2002.

[15] H. Marais and K. Bharat. Supporting cooperative and personal surfi ng with a desktop assistant. *ACM Symposium on User Interface Software and Technology*, pages 129–138, 1997.

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

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

[18] F. Schmuch and R. Haskin. GPFS: A shared-disk fi le system for large computing clusters. *Proceedings of FAST*, 2002.

[19] P. Sobe. Data consistency up- and downstreaming in a distributed storage system. *Proceedings of International Workshop on Storage Network Architecture and Parallel I/Os*, 2003.

[20] S. Vazhkudai. On-demand grid storage using scavenging. *New Trends in Distributed Data Access*, 2004.

[21] R. Wang and T. Anderson. xFS: A wide area mass storage fi le system. *Proceedings of 4th Workshop on Workstation Operating Systems*, pages 71–78, 1993.

[22] D. Zhou, L. Ou, X. He, and S. Scott. Online remote data backup for iSCSI-based storage systems. *Proceedings of the Internet Computing Workshop*, 2004.

[23] Y. Zhu, H. Jiang, and J. Wang. Hierarchical Bloom Filter (HBA): A novel, scalable metadata management system for large cluster-based storage. *Proceedings of International Conference on Cluster Computing*, 2004.