Alternatives of Implementing a Cluster File Systems

Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami
Fujitsu Laboratories LTD.
1-1, Kamiodanaka 4-Chome, Nakahara-ku, Kawasaki 211-8588, Japan
shinkai,tsuchiya,takeo@flab.fujitsu.co.jp
tel +82-44-777-1111
fax +81-44-754-2793
Jim Williams
Amdahl Corporation
1250 East Arques Avenue Sunnyvale, California 94088-3470
jaw10@amdal.com
tel +1-408-737-5005
fax +1-408-737-6595

Abstract

With the emergence of Storage Networking, distributed file systems that allow data sharing through shared disks will become vital. We refer to Cluster File Systems as a distributed file systems optimized for environments of clustered servers. The requirements such file systems is that they guarantee file systems consistency while allowing shared access from multiple nodes in a shared-disk environment. In this paper we evaluate three approaches for designing a cluster file system – conventional client/server distributed file systems, symmetric shared file systems and asymmetric shared file systems. These alternatives are considered by using our prototype cluster file system, HAMFS (Highly Available Multi-server File System). HAMFS is classified as an asymmetric shared file system. Its technologies are incorporated into our commercial cluster file system product named SafeFILE. SafeFILE offers a disk pooling facility that supports off-the-shelf disks, and balances file load across these disks automatically and dynamically. From our measurements, we identify the required disk capabilities, such as multi-node tag queuing. We also identify the advantages of an asymmetric shared file system over other alternatives.

1 Introduction

Historically, large corporations have deployed and distributed UNIX systems in a manner leading to isolated islands of processing. The management cost of these systems is exceedingly high because of the resulting overall complexity and a lack of a global management capability. Consequently, many of these same companies are now re-centralizing their critical Unix systems to contain management costs. One of the benefits credited to SANs (Storage Area Networks) is that they facilitate re-centralization of storage by aggregating it onto a common interconnect. We use the term SAN to describe a dedicated storage network utilizing a storage protocol such as SCSI over Fibre Channel, apart from LAN, which allow servers to communicate using a networking protocol such as TCP/IP. SANs are typically composed of Fibre Channel switches and hubs.

To extract the full potential of a SAN, data sharing, where multiple servers share a common file system on a directly-connected disk, and disk-pooling, where users can place data on any disk (disk-pool) without suffering management overhead, is required. Cluster file systems are the means for achieving these requirements. In this paper we

describe our experiences from developing HAMFS. HAMFS [1] is a prototype cluster file system, supporting disk pooling through a shared-disk capability. What was learned from HAMFS has been incorporated into our commercial file system product called SafeFILE. The rest of this paper is organized as follows: Section two describes alternatives to implementing a cluster file system and their characteristics. Section three presents overall HAMFS architecture. Section four shows measurement results from evaluating alternatives and necessary hardware capabilities using HAMFS. Section five describes related works. Finally, section six offers some brief conclusions.

2 Alternatives

As O'Keefe has shown [2], there are three alternatives to achieving data sharing between nodes in a shared-disk environment, client/server distributed file systems, symmetric shared file systems, and asymmetric shared file systems (Figure 1). In a conventional client/server distributed file system, a server node manages disk storage. Other nodes access data through the dedicated server across a communications network. While traditional client/server distributed file systems, such as NFS, cannot distribute user data across multiple nodes, some recent client/server distributed file systems, such as xFS [8], Zebra [9] and Frangipani [7], use multiple nodes for improved scalability. Alternatively, a second approach is symmetric shared file systems, such as GFS [3, 4, 20]. GFS allows every node equal access to all disks directly. The third alternative, asymmetric shared file systems, supports partial disk sharing. This approach was used in HAMFS. In this approach, a dedicated node manages disk blocks containing metadata, but all other disk blocks, containing user data, are accessed directly from all nodes.

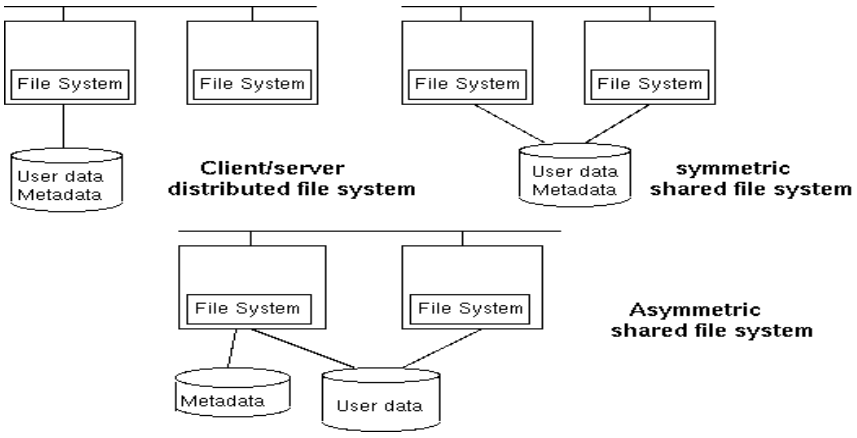

Figure 1: Alternatives for implementing a cluster file system

2.1 Characteristics of each Distributed File System Architecture

2.1.1 Complexity

The most challenging task for a designer of a cluster file system is maintaining consistency while guaranteeing good performance in a multi-node environment. Client caches, heavily used in UNIX file systems for improving performance, pose a major burden for designers. Although the symmetric shared file system has the simplest apparent organization, client caches, required for good performance, requires a complicated distributed lock manager. This leads to a rather complicated file system

organization. An example of this complexity is the difficulty of implementing a distributed logging mechanism. Other examples include support for atomic transactions with deadlock detection and recovery.

For symmetric shared file system organizations there are two approaches to achieving file system consistency. The first one is a distributed lock manager, as implemented in VAXcluster [5]. For high performance, VAXcluster offers a sophisticated distributed lock function. To maintain cache consistency, the VAXcluster's distributed lock manager uses both lock versioning for passive cache invalidation, and a callback mechanism for active invalidation. Another approach for serialization, as proposed in GFS, uses special device locks.

For guaranteeing file system consistency, the GFS[1] implementation basically uses a read-modify-write schema and disables the client cache. While this approach is relatively simple, its performance, particularly for short file access environments, is degraded from not having a client cache.

In general, symmetric shared file systems require that all client nodes share a common semantic view of data on disks, including location, record format, and meaning of each field and update sequence. This creates maintenance complexity and makes data sharing more difficult for heterogeneous environments. Any operating system vendor wishing to implement a particular symmetric shared file system must incur this complexity.

Client/server distributed file systems and asymmetric shared file systems avoid much of this complexity by localizing metadata access to a common node. Asymmetric shared file systems make use of a well-known fact that user data is rarely accessed by multiple nodes concurrently from multiple nodes. However, metadata is frequently accessed from multiple nodes concurrently [6]. Consequently, asymmetric shared file systems utilize a dedicated node for metadata management. This approach alleviates the disk contention resulting from concurrent metadata access. Additionally, shared direct access to disks, containing user data, reduces network overhead associated with conventional client/server distributed file systems.

2.1.2 Performance

Conventional client/server distributed file systems typically consume considerable network bandwidth and processor resources for both the client and server. Furthermore, these file systems cannot derive the maximum performance of the underlying disks. And because of their client/server organization, these file systems are not easily scaled through adding additional disks. Balancing performance after adding new servers typically require a manual file system reconfiguration. Both symmetric and asymmetric file systems are easier to scale than conventional client/server distributed file systems like NFS. The reason is because they support common disk pools accessible from multiple nodes that make adding disks far simpler.

A variation of a client/server distributed file system, Frangipani [7], xFS [8], and Zebra [9] solve the scaling problem by distributing data across multiple nodes. However, they still inherit the drawback related to transferring data across a network. While asymmetric

1 Although a new GFS implements lock versioning for permitting cached data, it does not solve the inherent performance problem. [20]

shared file systems require a means for transmitting control messages across a communications network, the amount of data transmitted is small compared with client/server distributed file systems. Comparatively speaking, asymmetric shared file systems require greater message exchange across a networking than symmetric shared file systems. This is because of the communication between the metadata manager and the other nodes. However, by improving protocols within the asymmetric shared file system, such as a space reserve function, fine grain tokens and token escalation, byte range logging, and support for a secondary buffer cache, much of the performance degradation is minimized. Our experience with HAMFS indicates that these optimizations are possible without significant additional complexity.

2.1.3 Scalability

Both client/server distributed file systems and asymmetric shared file systems have scalability problems related to localized loading of management function on a dedicated node. Distributing the management function across multiple nodes eliminates this drawback. This distribution is akin to the distributed lock manager used in symmetric shared file systems. For instance, both Frangipani and xFS partition file system space into separate segments. A separate node manages each space segment.

2.1.4 Reliability

Both client/server distributed file systems and asymmetric shared file system have a single point of failure because they rely on a single dedicated node. This drawback can be eliminated for asymmetric shared file systems by using a replication schema. Moreover, by deploying an improved logging mechanism, as implemented in HAMFS, and replicating only metadata, asymmetric file systems outperform local file systems. This is a significant advantage over client/server distributed file systems since they would otherwise require mirroring all data.

Disk contention and distributed lock management overhead leads to performance problems for symmetric shared file systems. However, with asymmetric shared file systems, these issues can be addressed through various compromises. The following section describes HAMFS' organization and describes some of these compromises.

3 HAMFS file system

HAMFS is classified as an asymmetric shared file system. Describing the detailed operation of HAMFS is beyond the scope of this paper, however we do briefly review its high-level organization.

3.1 Configuration

HAMFS divides the contents of the file system into three parts, metadata, user data, and log data, which are stored on Meta volumes, Data volumes, and log volumes, respectively.

HAMFS software consists of two components, HAMFS client, and the Name Server. To prevent contention from frequent Meta volume writing, the Name Server, which manages Meta volumes and log data, runs on a dedicated node in a cluster system. Conversely, HAMFS client (client) code embedded in the kernel of each node as a Virtual File System (VFS), resides on every node in a cluster and directly accesses Data volumes. Data volumes access is managed with the file extent information provided by the Name Server. This information reflects data location on each Data volumes.

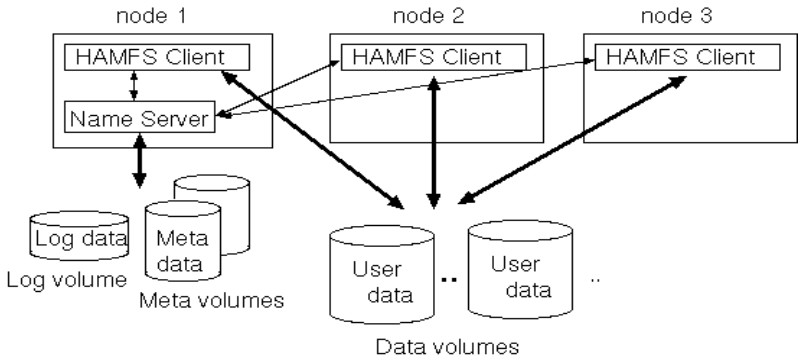

Figure 2: Configuration of HAMFS file system

The Name Server processes requests from the client as atomic transactions using a log schema. The interface between the client and the Name Server is a high-level protocol and is independent of metadata format. This protocol is similar to the NFS protocol. For improved availability, the user may replicate the Name Server on a secondary node. When a secondary Name Server is deployed, Meta volumes are replicated on the secondary Name Server by transmitting the log data. HAMFS file organization permits users to select the disk organization used for each data type as well as whether metadata is replication.

3.2 Disk pooling

HAMFS offers the following disk pooling capabilities.

3.2.1 Disk Pool

A disk-pool is defined as a set of data volumes comprising a HAMFS file system. HAMFS manages each member disk. Data placement across the disk pool is managed automatically according to a placement policy. Current placement policy equalizes the amount of free space across volumes.

3.2.2 File RAID

Because HAMFS manages the underlying disk devices, it's possible to allow files to have different RAID properties in a common name space. File RAID is the function for permitting this file placement policy. A user can specify RAID type of None (default), RAID 0 (striping), RAID 1, and RAID 5 through a new CHATTR command. All directories and files inherit a parent directory's RAID type. With RAID five the client allocates a parity block for each stripe in a file. Since this works like a RAID type one for a small files having only one data block, common in UNIX environments, parity recalculation overhead is greatly reduced.

3.2.3 Dynamic Expansion

An installation may dynamically expand a file system with new disks even though user applications are running. Because HAMFS automatically balances load across the old and new disks, file name tree reconstruction is unnecessary. A newly added Data volume remains offline until all client nodes send ADDVOL requests for the new volume. The Data volume is then brought online only after the Name Server receives an ADDVOL requests from every client.

3.3 Tokens

For improved file system performance, each client caches user data and file information (file data) in their local memory. The consistency of cached file data is guaranteed with tokens that are managed by the Name Server.

There are five token types, NAME, TIME, SIZE, ATTR and DATA. NAME and ATTR tokens correspond to directory entry and file attributes (or directories) that are not represented by other tokens.

Every file has a set of associated DATA tokens. There is a corresponding DATA token for each file data block. Owning a DATA token guarantees accuracy of file extent information reflecting the location of data on the Data volumes. Using this file extent information, clients access Data volumes directly and independently. The Name Server provides file extent information along with the token when a client requests a DATA token. When new data is written, the client must first obtain the corresponding DATA write token, which allows the client to cache the user data locally. Afterwards, if another client wants to read this cached data, the Name Server callbacks the write DATA token from the client owning it. On receipt of a callback request, the client allocates new disk space (file extent) and writes any cached user data. The client honoring the callback request also updates any newly allocated file extent information.

The TIME token allows multiple clients to concurrently access common files while guaranteeing the accuracy of file access times. The SIZE token permits independent programs running on different nodes to both write and read from different parts of the same file.

On a data token request, the Name Server provides all the required non-data related tokens plus, if available, all DATA tokens for the entire file (Token Escalation). Through token escalation, most clients obtain the needed tokens, including file extent information, at file open time.

3.4 Space Allocation

When a client must allocate additional disk space, it selects a pre-allocated extent based on the amount of cached data to be written. In most cases, only one extent with consecutive disk blocks is allocated at file close time. The Name Server is notified of this through the CLOSE request. On notification of their usage, pre-allocated (reserved) extents become file extents when the DATA write token is released or the file is closed. When the number of pre-allocated extents drops below a threshold, clients replenish their free pool with a RESERVE request made to the Name Server. On receipt of a RESERVE request, the Name Server provides free extents on a Data volume with the greatest free space.

To reduce wasted space related to allocation, every extent, including free extents, is represented with a B-tree data structure. The Btree structure is stored in the inode for files consisting of only a few contiguous extents. With this space allocation function, combined with deferred file writes, reduces message overhead and permits allocating contiguous disk blocks for files.

3.5 Transaction processing

For increase performance and availability, the Name Server processes file operations requested by clients as a single atomic transaction using logging and a two-phase lock with deadlock detection mechanism.

3.5.1 Logging

These transactions complete quickly because they require writing only a small amount of data to the log volume. Actual updates to Meta volumes are deferred as long as possible and are performed completely and asynchronously.

Log data generated by HAMFS is an after image log containing only the modified range of data instead of the entire contents of the modified blocks. This byte-range log significantly reduces the amount of log data and improves metadata update performance. The log data is written on a log volume in a cyclic fashion, synchronously before a command response is returned to the client. Actual metadata updating of Meta volumes is deferred as long as possible by using a secondary buffer cache. The secondary buffer cache caches any modified and committed metadata that has not been written back to a Meta volume. Dirty metadata, cached in the secondary buffer cache, is asynchronously written when the amount of available space in the log volume or the number of non-dirty blocks in the secondary cache reaches a threshold. Furthermore, since the metadata writes are aggregated, the total I/O load is reduced.

3.5.2 Deadlock detection

With the HAMFS token schema, deadlock avoidance, as used in conventional file systems, would be difficult to implement. For example, assume the following scenario. The Name Server processes a file remove request for one client while another client is writing to the file. To complete the remove request, the Name Server must callback any DATA write tokens related to the file from other clients. On receipt of this callback request, the client requests the Name Server to pre-allocate disk space with a RESERVE request for writing back cached data. After the Name Server replies to the RESERVE request, the client writes its cached data to the pre-allocated extent returned from the Name Server. Afterwards, it notifies the Name Server of any allocated extents. In such circumstances, determining the access order to resources required for preventing deadlocks, while probably possible, would be difficult.

In HAMFS, we developed a deadlock detection mechanism for easing development and maintenance. The Name Server uses several threads for processing client requests. When requests arrive, idle threads process the requests. A two-phase lock technique maintains consistency among these requests. Deadlocks result when two or more threads obtain multiple vnode locks, buffer cache locks, or tokens. Deadlocks are detected by maintaining the following information in a linked list by the Name Server.

- Identifier of the thread owning a resource for vnode or buffer cache lock.
- Resource identifier for threads waiting.
- Vnode address for tokens a thread is waiting on.

When a deadlock situation occurs, the Name Server cancels one of the conflicting transactions that led to the deadlock and retries it from the beginning. Memory resident control blocks, such as in-memory inodes, are automatically restored. With HAMFS

deadlock detection and recovery, the order for updating metadata can be selected from a performance viewpoint as opposed to a consistency and deadlock avoidance viewpoint. Therefore, complicated error recovery and deadlock avoidance logic, scattered throughout the file system, is avoided.

3.6 Replication

For fast recovery and improved performance in cluster environments, HAMFS replicates the Name Server. There can be separate primary and a secondary Name Servers for each file system. However, HAMFS clients only communicate with the primary Name Server.

The secondary Name Server possesses a replicated copy of the metadata. If the primary Name Server crashes, the secondary Name Server takes over the primary role using the replicated metadata. When a secondary Name Server is deployed, instead of writing log data to a log volume, before signaling an operation complete, the primary Name Server simply transfers the log to the secondary Name Server. The secondary Name Server acknowledges its receipt. After receiving the acknowledgment, the primary Name Server is free to signal a completion to the client. We call this technique Early Commit. Actual metadata updating on Meta volumes is deferred and done asynchronously using a secondary buffer cache on both name servers. If a power failure occurs, modified metadata, in the secondary buffer cache, is written back to the Meta volumes with the aid of an UPS.

3.7 Crash Recovery

When a client recognizes that the primary Name Server has crashed, it begins communicating with the secondary Name Server (new primary Name Server). The new primary Name Server reconstructs tokens and file lock status using information sent by the clients. Afterwards, the Name Server resumes processing. If the new primary Name Server detects requests already committed by the old primary Name Server it simply replies with the saved reply status transmitted by the old primary Name Server.

When the Name Server detects a client crashed, the Name Server releases any tokens and file locks held by the failed client. After this step, the Name Server releases pre-allocated extents also owned by the crashed client.

3.8 Status

The current HAMFS prototype is operating in our laboratory with the following configuration. The Name Server running as a user mode daemon with multiple threads on Solaris. Clients are running as a VFS file system in the FreeBSD kernel. All functions, except for file RAID described in this paper, have been implemented.

4 Measurement Results

We evaluated various file system alternatives by measuring performance of an asymmetric file system (HAMFS) and a distributed file system (NFS). We did not measure performance of a symmetric shared file system. We can only comment on the performance of a symmetric file system where it relates to our other measurements.

We used three PCs with 64MB memory for these measurements. For the NFS measurements, two PCs were running FreeBSD with one operating as a NFS client and the other as a NFS server, respectively. For the HAMFS measurements the three PCs

were configured with the first PC running FreeBSD and acting in a client role and the other two PCs operating as primary and secondary Name Servers running the Solaris operating system. Consequently, the measured data represents HAMFS performance for a cluster environment.

These three PCs were connected together with a 100Mbps Ethernet. In every case, a common disk was connected to the first two PCs. This disk contained all HAMFS volumes, except for replicated Meta volume. This was done to create a fair comparison with NFS. The replicated Meta volume resided in a dedicated disk on the third PC running the secondary Name Server. This replication configuration is justified because separate disk should be used in a replicated environment.

4.1 Distributed file system vs. Asymmetric shared file system

We measured HAMFS and NFS performance for small and large files. This was done to compare access performance of a distributed file system to an asymmetric shared file system. We chose NFS version 3 to represent the distributed file system. For NFS measurements, two PCs running FreeBSD were used. Since NFS invalidates cached data on a predetermined time interval, it generates more control traffic over the network than other distributed file systems. But we believe the measured results apply to other client/server distributed file systems.

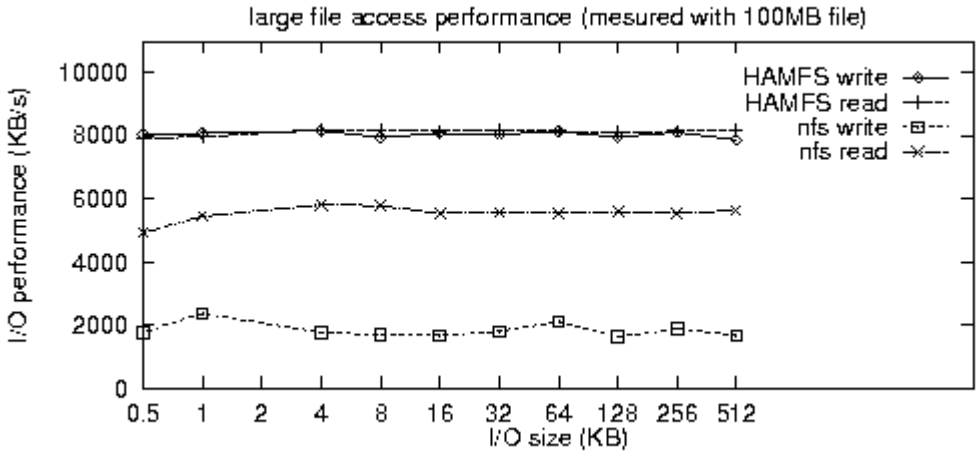

Figure 3: Large file write performance

Figure 3 shows the performance for large file access environments. In these measurements, a 100MB file is created and read sequentially from the beginning. HAMFS shows far greater performance, compared with NFS, because it is accessing data directly from disk through a high-performance disk path. An interesting point is that NFS write performance is poor although writing data on server side is done asynchronously with the NFS v3 commit feature. The reason for this is that the network cannot drive the disk with enough data necessary to derive its maximum performance, resulting in additional rotational delay. On the other hand, as the track buffer in the disk unit offsets the network overhead, NFS reads reflect relatively good performance. As disk transfer

rates continue to improve, delays due to networks will further impact performance. However, the client cache provides a speed-matching buffer, which alleviates this problem, somewhat by aggregating data into larger chunks and writing them as contiguous blocks on disk. We believe asymmetric shared file systems have an advantage in this regard over symmetric shared file systems because they can make better use of the client cache.

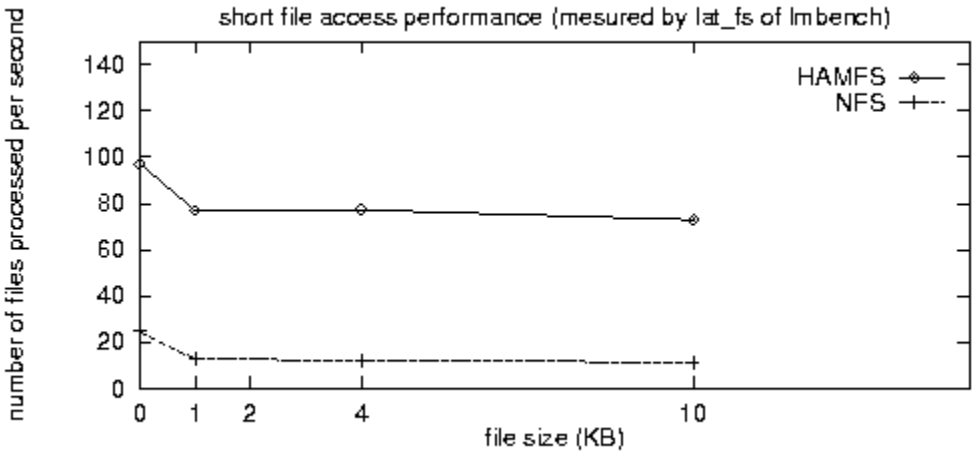

Figure 4: Short file access performance

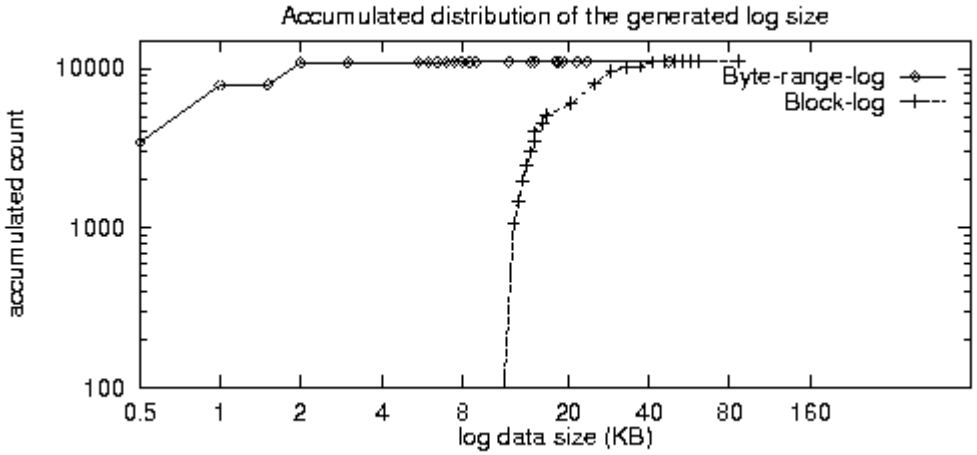

Figure 5: The amount of log generated

Figure 4 presents the performances for short file access environments. These measurements were conducted using the lat_fs program in lmbench [10] micro benchmarks. This micro bechmark program creates 1000 files with various sizes and then removes them. The vertical line reflects how fast this program runs. As shown in Figure 4,

HAMFS, with a secondary Name server, outperformed NFS by a factor of five. The reason is that metadata updating through Early Commit is faster than doing actual disk I/O.

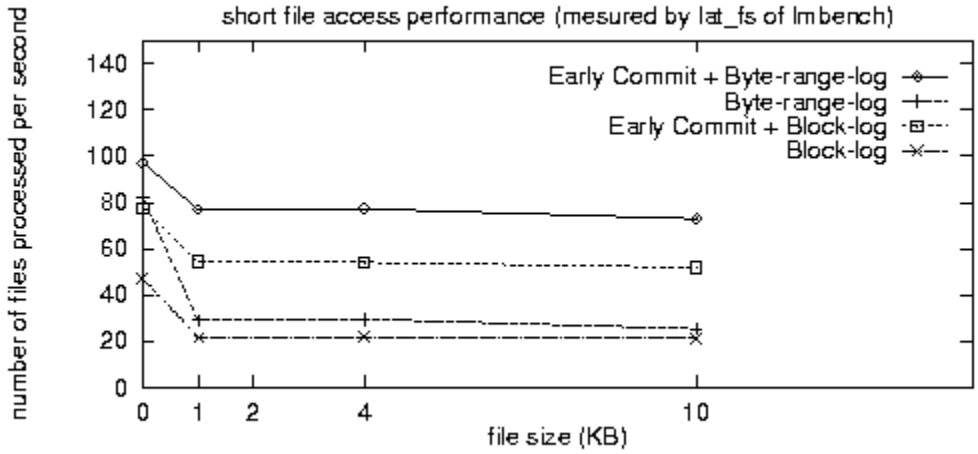

Figure 6: Effect of a Byte-range log

Figure 5 and 6 shows how size of log affects total performance.

Since HAMFS uses a Btree data structure for representing space information on Meta volumes, more disk blocks are updated on behalf of a file operation when compared with UNIX file systems. UNIX uses an array of block address lists and bit maps for representing available disk blocks. Figure 5 illustrates the distribution of generated log data size per file operation in this measurement. Without a byte-range log, most log data writes would be 30K data. Less than 2K bytes of log data is generated with a byte-range log (that is when the log is produced with byte granularity). This difference is apparent even in a single name server environment (the second line vs. the last line in Fig. 5) [2]. It has a more drastic effect on the performance in a duplicated name server environment (the first line vs. the third line in Fig. 5). Additionally, this figure indicates the superiority of asymmetric shared file systems over traditional client/server distributed file systems for improving availability through replication. Traditional distributed file systems must transfer all user data, as well as metadata, making it difficult to reduce data replicated over the network. Finally, we believe this measurement also reflects the superiority of asymmetric file systems over symmetric file systems since logging puts additional implementation burden for symmetric shared file systems.

4.2 Limitations of asymmetric shared file systems

Figure 7 shows aggregate write performance for large files in a shared environment. In this measurement, two PCs are used as clients with a third PC running as a name server.

[2] In actual single node environments this difference may have more impact on performance. Because HAMFS measurements were conducted in a cluster environment with the Name Server and the client running on separate nodes, the logging overhead has less effect on total performance.

Performance measurements are not as good when compared with Figure 3. The last line, annotated UFS, shows how performance degradation is independent of file system type. The reason is that the disk devices used for these measurements cannot efficiently process requests from multiple nodes. To validate our hypothesis, we measured the case when two processes running on the same client write large, but separate files on separate UFS file systems (disk partitions). The second line tagged with UFS single shows this performance impact.

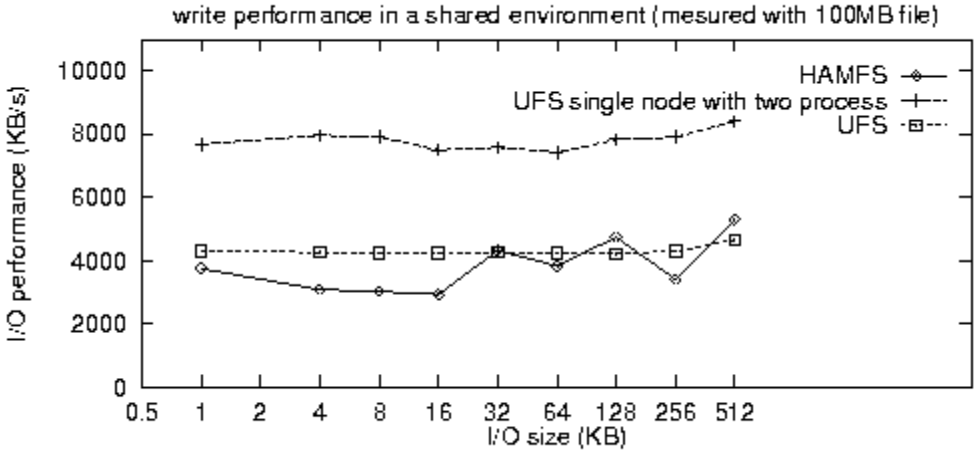

Figure 7: Large file access in a shared environment

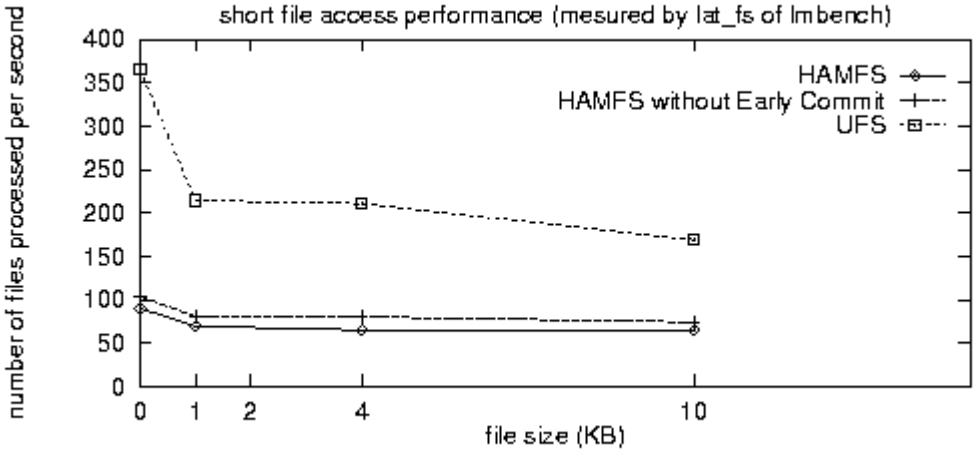

Figure 8: Short file access with high-speed disk array

As the figure 7 shows, disk tag queuing performs well when two processes are running on the same node and writing concurrently. On the other hand, when two processes on separate nodes write large files, the loss of tag queuing is evident. Consequently, for

extracting maximum performance from disks in a cluster environment, tag queuing support across multiple nodes is a critical feature.

Figure 8 shows performance for short file operations on a high speed RAID disk array with 32MB nonvolatile memory.

Compared with Figure 6, HAMFS performance is not as good as indicated in the previous measurements. In previous measurements, HAMFS outperformed the local file system, however, for this measurement, UFS outperformed HAMFS. The reason for this is that writes for short blocks are faster with a nonvolatile cache in a RAID array than having to transfer control data over network. This result also suggests that writing small log data to disk (HAMS without Early Commit) is faster than transferring log data across a network (first line and the second line in Figure 8). Consequently, eliminating any unnecessary communication between the nodes and reducing the amount of data transferred over the network is essential to an efficient cluster file system. An important message is that a cluster file system must adapt to underlying disk topology for best performance. These measurements indicate that the preferred mode of operation is highly dependent on system configuration.

5 Related Works

Devarakonda [11] evaluated alternatives for implementing a cluster file systems. The author compares a symmetric shared file system with a token-based client/server distributed file system. They conclude that their client/server distributed file system (Calypso) provides much better performance than a symmetric shared file system. In our paper we have shown how an asymmetric shared file system can outperform a distributed file system organization.

GFS [3,4, 20] is an example of a symmetric shared file system. It proposes a special hardware feature in the disk providing multiple logical locks. However, an asymmetric shared file system can accommodate off-the-shelf disk devices. Additionally, we expect GFS suffers from low performance as a result of heavy disk contention except in specialized environments such as broadcasting.

NASD [12] and HPSS [13] are similar to asymmetric shared file systems. They isolate metadata and user data, and permit shared access to user data directly from clients through a high-speed communications network. However, since they both transfer user data across a communication network they have performance limits that HAMFS does not. In addition, they require special hardware features as GFS does; HPSS utilizes a complicated two-phase commit mechanism suitable only in scientific environments where large files are dominant. NASD depends on intelligence in the disk devices for space allocation.

Zebra [9] and xFS [8], are examples of client/server distributed file systems. They scale performance by distributing data across servers in a RAID schema through a LFS technique. However, user data is transferred over a communications network and their RAID schema is fixed. HAMFS supports file RAID for increased performance and allows the user to specify data striping policy on a file basis. Frangipani [7] also offers a similar capability by distributing data across servers using a network virtual disk function. This approach has many of the same drawbacks as xFS.

HAMFS deploys many of the similar techniques developed for increased performance and availability used in many of the documented file systems, such as Cedar [14], Echo [15], XFS [16], Locus [17], HARP[18], and Spritely NFS [19]. Additionally, HAMFS offers features that others do not, including Token escalation, Space reserve, and automatic deadlock detection.

7 Conclusions

The asymmetric shared file system organization is a superior approach for implementing a commercial cluster file system. They outperform client/server distributed file systems and symmetric shared file systems for many common access environments. Because disk bandwidth improvements have outpaced network bandwidth improvements, asymmetric shared file systems' performance is superior to that of distributed file systems. Additionally, processor overhead associated with distributed file systems is not evenly distributed across clients, but highly localized to a server. In a large cluster environment, this limitation quickly becomes a bottleneck. However, to extract the full performance of an asymmetric shared file system, tag queuing across multiple initiators is required. Also, having an efficient protocol for reducing communication between clients and Name Server is important as well. Finally the current HAMFS prototype may have some scalability limitations because of a single Name Server per file system restriction. We do not expect this limitation to be a real problem in the short term. If required, we could remove this restriction using similar techniques as used in the Frangpani distributed file system.

References

[1] Yoshitake Shinkai, Yoshihiro Tsuchiya, Takeo Murakami, and Jim Williams. HAMFS File System. In proceedings of 18th IEEE Symposium on Reliable Distributed Systems, pages 49-70, Lausanne, Switzerland, October 1999.

[2] Matthew T.O'Keefe. Shared file systems and fibre channel. pages 1-16, In proceedings of Fifteenth IEEE Symposium on Mass Storage Systems, College Park, Maryland, March, 1998

[3] Steven R. Soltis, Grant M. Erickson, Kenneth W. Preslan, Matthew, T. O'Keefe, and Thomas M. Ruwart. The Design and Performance of a Shared Disk File System for IRIX. In proceedings of Sixth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, College Park, Maryland, March, 1998

[4] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The global file system. In proceedings of the 5th NASA Goddard Mass Storage Systems and Technologies Conference, College Park, MD., September 1996.

[5] Nancy P. Kronenberg, Henry M. Levy, and William D. Strecker. VAXclusters: A Clos ely-Coupled Distributed System. ACM Transactions on Computer Systems, 4(2): 130-146, May 1986.

[6] Mary Baker, John H. Hartman, Michael D. Kupfer, Ken Shirriff, and John K. Ouster hout, Measurements of a Distributed File System. pages 198-212, In proceedings of th e Thirteenth ACM Symposium on Operating System, Pacific Grove, California,Octobe r1991.

[7] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A Scalable Distributed File System. In proceedings of sixteenth ACM Symposium on Operating System Principles, pages 224-237. Saint Malo, France, October 1997

[8] Thomas E. Anderson, Michael Dahlin, Jeanna lvi. Neefe, and David A. Patterson. Serverless network file systems. ACM Transactions on Computer Systems, 14(1):41-79, February 1996.

[9] John H. Hartman and John K. Ousterhout. The zebra striped network file system. .ACM Transactions on Computer Systems, 13(3):274-310, Aug. 1995.

[10] Larry McVoy and Carl Staelin. lmbench: Portable tools for performance analysis. In pr oceedings of USENIX 1996 Annual Technical Conference, San Diego, CA., January, 1 996.

[11] Murthy V. Devarakonda, Ajay Mohindra, Jill Simoneaux, and William H. Tetzlaff. Evaluation of design alternatives for a cluster file system. In USENIX 1995 Technical Conference Proceedings, pages 35-46, Berkeley, CA, January 1995.

[12] Garth Gibson, David Nagle, Khalil Amiri, Fay Chang, Eugene Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems, Seattle, WA., June 1997.

[13] David Fisher, John Sobolewski, and Terrill Tyler. Distributed metadata management in the high performance storage system. In Proceedings of the FIRST IEEE METADATA CONFERENCE, Silver Spring, Maryland, April 1996.

[14] Robert B. Hagmann. Reimplementing the cedar file system using logging and group commit. In Proceedings of Eleventh ACM Symposium on Operating System Principles, pages 155-162, Austin, Texas, November 1987.

[15] Andy Hisgen, Andrew Birrell, Charles Jerian, Timothy Mann, and Garret Swart. New-value logging in the echo replicated file system. Technical Report SRC Research Report 104, Digital Systems Research Center, June 1993.

[16] A. Sweeney, D.Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the xfs file system. In proceedings of the USENIX 1996 Annual Technical Conference, pages 1-14, San Diego, CA, January 1996.

[17] B. J. Walker, G. J. Popek, R. English, C. S. Kline, and G. Thiel. The locus distributed operating system. In proceedings of Ninth ACM Symposium on Operating System Principles, pages 49-70, Bretton Woods, New Hampshire, October 1983.

[18] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the harp file system. In Proceedings of the Thirteenth ACM Symposium on Operating System Principles, pages 226-238, Pacific Grove, CA, October 1991.

[19] Jeffrey C. Mogul. Recovery in spritely nfs. ACM Transactions on Computer Systems, 7(2):201- 262, 1994.

[20] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, Mattew T. O'Keefe. A 64-bit, Shared Disk File System for Linux. In proceedings of Seventh NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, pages 22-41,San Diego, California, March 1999.