Cache Coherency in Distributed File System

Anagha Kulkarni anagha.kulkarni@cumminscollege.in
Department of Information Technology,
Cummins College of Engineering for Women,
Pune, 411052, India.

Radhika Bhagwat radhika.bhagwat@cumminscollege.in
Department of Information Technology,
Cummins College of Engineering for Women,
Pune, 411052, India.

Abstract

Principle of locality states that most memory references are made to a small number of memory locations. Not only that, memory locations near most recently referenced locations are more likely to be referenced than one further away. To take advantage of this, cache memory is inserted between memory and CPU [1]. Better utilization of cache is crucial for good performance of distributed file system; even in case of remote file accesses.

Not caching a file during writes prolongs the session, thereby increasing write-sharing time, leading to slow performance especially on WANs. This paper introduces a technique to reduce miss penalty during remote file writes and allows write sharing in LAN. It uses the principle of divide-and-rule and arranges the system into hierarchical domains and then gives ownerships to the writers.

Keywords: Cache Coherency, Distributed file system, Performance, WAN.

1. INTRODUCTION

Caches are used extensively in practice. It is a store of recently used data objects that is closer than the objects themselves. When a client needs an object it first checks its cache for the object. If not found or if it does not have a valid copy, it is fetched from the target computer and it is added or replaced in cache. This is done to increase the availability and performance of the service by reducing the latency in fetching the object [2].

Sharing data is fundamental to distributed systems. Therefore, distributed file systems form a very important part of distributed systems. They allow processes to access data stored at a server in secure way similar to data on local disk [3]. Due to this, existence of same data or collocation of a file on multiple client caches is normal in distributed systems.

This sharing of files comes at price. Shared files must remain consistent. When a file is being written and read simultaneously by different clients, there is a potential for different versions of same file in every clients' cache. This is when we say caches are not consistent. Changes made by a client have to be propagated to the appropriate file server, but it involves a finite amount of delay. In addition, if there are replicas of files, maintaining stronger degree of consistency is more time consuming. Therefore, managing cache consistency in distributed file systems is very challenging.

Synchronization is the solution to ensure consistency of files.

The remainder of this paper discusses related work in section 2, proposed cache coherency model in section 3 and conclusion of the paper in section 4.

2. RELATED WORK

Sun Network File System (NFS) [4] does not support remote file locking. That means locks are maintained locally by the file server. When a client wants to obtain a write/read lock on a file, it has to request lock manager on the file server. This prevents writing of one file by different clients simultaneously. It uses UNIX semantics for maintaining file consistency. All reads and writes go to the file server, which are processed sequentially. So maintaining consistency is easier. Drawback of this mechanism is that it does not scale well.

Whenever a client needs a file, it sends a request to the Andrew File System (AFS) [5] server (file may be replicated). The file is sent to the client. The client then operates on the file locally. If the file is updated, the changes are sent back to the server when the file is closed. The server then sets the ‘valid flag’ to ‘cancelled’ of all the clients which have the file in their cache. This prompts the client to re-fetch the latest version of the file. This makes AFS scalable. But if two or more clients are writing into the same file simultaneously, the one who finishes early wins and ‘valid flag’ for that file of other clients is set to ‘cancelled’.

Cache consistency is maintained in both the file systems, but altogether in a different way. NFS server is stateless and does not transfer the file to a client machine. AFS server is stateful and transfers the file to each and every client.

CODA [6] file system is a descendant of AFS that is more resilient to failures. It introduces a concept called Volume Storage Group (VSG) which contains set of replicas of a volume. Ideally, a client is connected to VSG and modifications are propagated to all replicas in VSG. In case one or more replicas are not up or if there is a network partition, then a client sends the modifications to Accessible Volume Storage Group (AVSG). If the client gets disconnected from all the servers (i.e. AVSG is a null set), it continues to use locally available file and when the connection is established, it propagates the changes to its AVSG, if any.

Mobile Agent-based Distributed File System (MADFS) [7] is conceptually a new Distributed File System suitable even in Wide Area Network (WAN). It improves the performance of distributed system by reducing the network transfer and the overhead of cache management.

MADFS divides the whole system into number of domains, each managed and controlled by domain server (DS). All the hosts within a domain are connected by high-speed Local Area Network (LAN). All the DSes are connected to each other by low-speed WAN. The DSes are managed and controlled by a main server (MS). Thus the whole system is hierarchically managed which reduces load on a single server. Every host, DS and MS has a mobile agent capable of accepting request from client process, moving the order to target server to execute and also responsible for name, cache, access management (in case of domain management agent).

Advantages of MADFS are: It works well in WAN by reducing traffic in WAN and reduces overhead in cache coherency management by using Hierarchical and Convergent Cache Coherency Mechanism (HCCM). The disadvantages are: It does not support write sharing.

Sprite File System [8] lets each host cache the file blocks for reading by multiple readers or by single writer at a time. It uses system memory for caching the blocks and not the disk. During write sharing Sprite File system disables caching altogether whenever there is a writer with any other readers or writers.

Ownership-based Cache Consistency Model in a Distributed File System [9], based on Sprite File System, improves the performance of the file system during write sharing. Advantages are: It supports write sharing and works well only in LAN. The disadvantage is: It does not have good performance in WAN.

3. THE PROPOSED CACHE COHERENCY MODEL

The proposed distributed file system is based on applying the technique of ownership based file system to MADFS. It has been observed that when a host wants to read or write a file, almost 2/3rd of the times it has been accessed by another host within the same domain [10]. This paper presents a model for getting better performance during write sharing within a domain.

As in MADFS, the whole distributed system should be divided into domains; all hosts within a domain should be connected by high-speed link and controlled by DS. All DSes should be connected to each other by low-speed links and controlled by MS.

Every file in the proposed file system should have read and write locks as in the case of MADFS. All the servers using this technique should be stateful. MS should maintain a repository to record the details of file accesses. It should have following details:

File name, version no, DS id, type of lock (read/write), current owner (indicating DS id)

DS, too, has to maintain a repository similar to the one maintained by MS. It must have following details:

File name, version no, host id, type of lock (read/write), current owner (indicating host id)

When a host needs to read a file, it should send request to its DS. One of the two circumstances may arise.

4. DS may already have a valid read lock on that file - it forwards the cached file to the host.

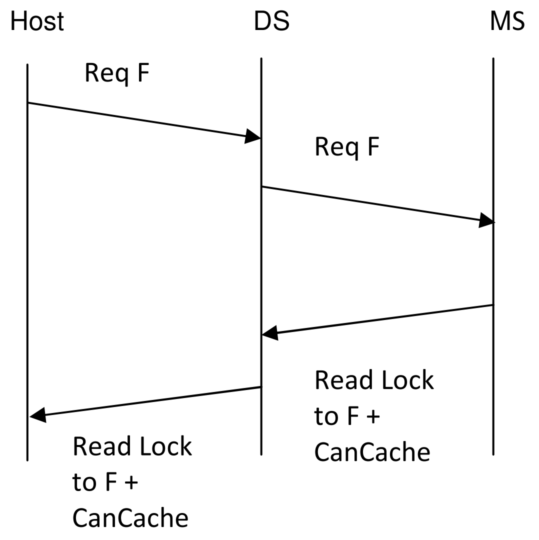

FIGURE 1: Exchange of messages between host, DS and MS for obtaining Read Lock

5. DS may not have a valid read lock on that file – As shown in figure 1, DS forwards the request to MS. MS makes entry into its repository and assigns a read lock to requesting DS. It also sends a 'CanCache' to DS. Upon acquiring the read lock on the requested file and 'CanCache', it grants read lock and sends 'CanCache' to requesting host. DS makes entry into its repository. Valid 'CanCache' allows the requesting entity to cache the file.

When a host needs to write into a file, it sends request to its DS. One of the two circumstances may arise.

1. DS may not have a valid write lock on that file – As shown in figure 2, DS forwards the request for write lock to MS. MS makes entry into its repository and assigns a write lock to requesting DS. It also sends a 'CanCache' as before and 'TempOwner' to DS. 'TempOwner' indicates that the requesting DS holds all ownership rights to the file temporarily (i.e. as long as it is writing the file). Upon acquiring the write lock on the requested file, 'CanCache' and 'TempOwner', it grants write lock and sends 'CanCache' and 'TempOwner' to requesting host. DS makes entry into its repository.

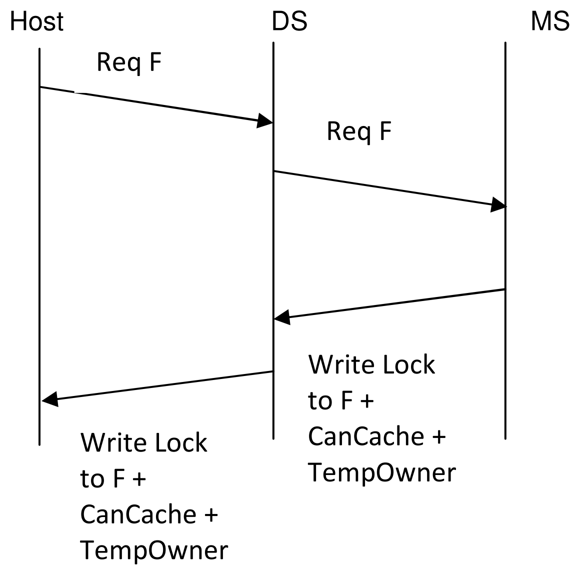
Host
DS
MS
Req F
Req F
Write Lock to F + CanCache + TempOwner
Write Lock to F + CanCache + TempOwner

FIGURE 2: Exchange of messages between host, DS and MS for obtaining Write Lock

2. DS may have a valid write lock on that file – This means some other host in the same domain is writing the file. Hence DS should tell requesting host about the 'NewOwner'. If the requesting host receives 'NewOwner', it means it cannot cache the file into its system memory but has to access it remotely.

After writing is completed by the writer, it flushes all the dirty data to the server and releases lock on the file, informs DS which in turn informs MS. DS and MS should remove the respective records from the repository.

Write sharing is said to occur when there is a writer with other readers or writers. It occurs in two cases:

1. Writer and its DS have obtained a write lock and other readers or writers in the same domain want to open the file.

Assume h1 in DS1 has obtained a write lock on file F. h4 wants to obtain a read (or write) lock.

h4 will request DS1. DS1 knows h1 is owner of F, so it tells h4 about 'NewOwner' h1 and maintains h4's record into its repository. h4 now cannot cache F, but should read (or write) F from h1's cache ('CanCache' is not valid for h4), as shown in figure 3.

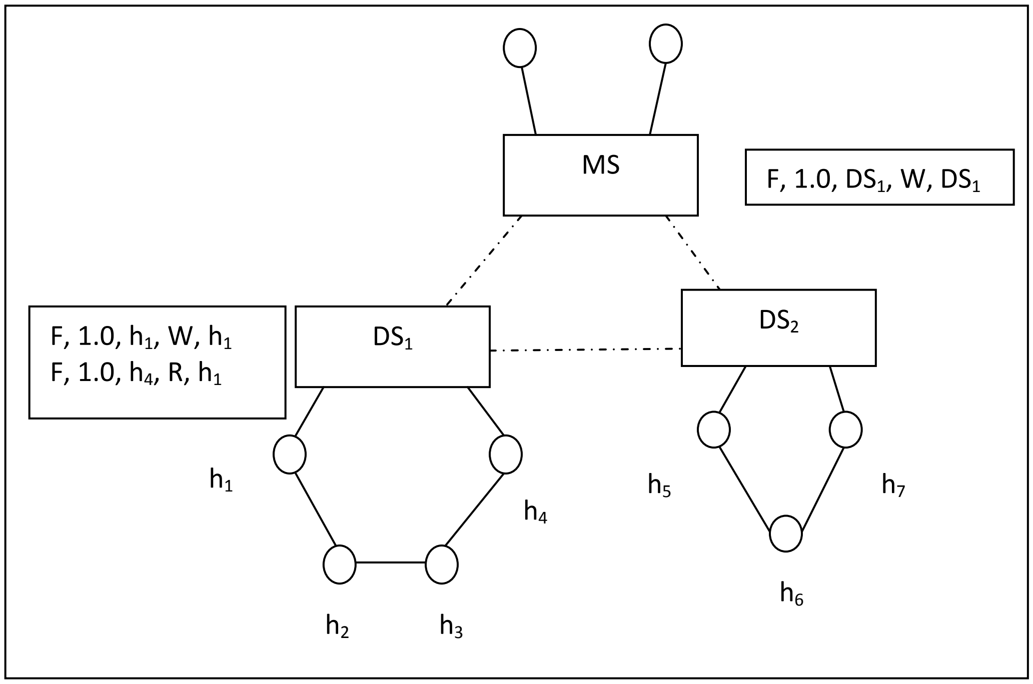

FIGURE 3: Repositories of MS and DS1

2. Writer and its DS have obtained a write lock and other readers or writers from other domain(s) want to open the file.

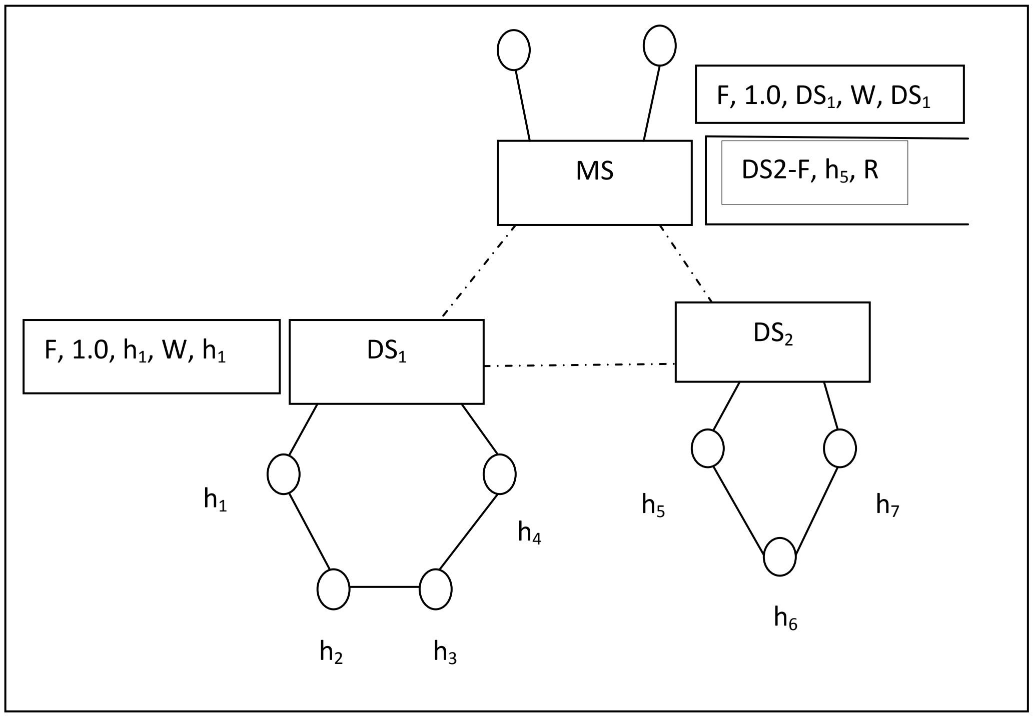

FIGURE 4: Repositories of MS and DS1 and PendingRequestQueue of MS

Assume h1 in DS1 has obtained a write lock on file F. h5 in DS2 wants to obtain a read (or write) lock on F.

It requests DS2. DS2 requests MS. Since the ownership of the file is transferred to DS1, DS2's request is kept pending in the 'PendingRequestQueue' maintained by MS, as shown in figure 4.

3. Readers across different domains and their respective DSes have obtained a read lock and a host wants to obtain a write lock.

Assume hosts h4 from DS1, h5 and h7 from DS2 have obtained read lock on F, as shown in figure 5. h1 in DS1 wants to obtain write lock on F.

h1 requests DS1 for write lock. DS1 requests MS. MS sets 'CanCache' of DS2 to invalid. DS2 sets 'CanCache' of h5 and h7 to invalid. It then grants write lock to DS1. MS's repository changes to:

F, 1.0, DS1, W, DS1

DS1 now grants write lock to h1 along with 'CanCache' and 'TempOwner'. Repository of DS1 changes to:

F, 1.0, h4, R, h1
F, 1.0, h1, W, h1

All the requests to MS from all other DSes are kept pending into 'PendingRequestQueue' as explained before.

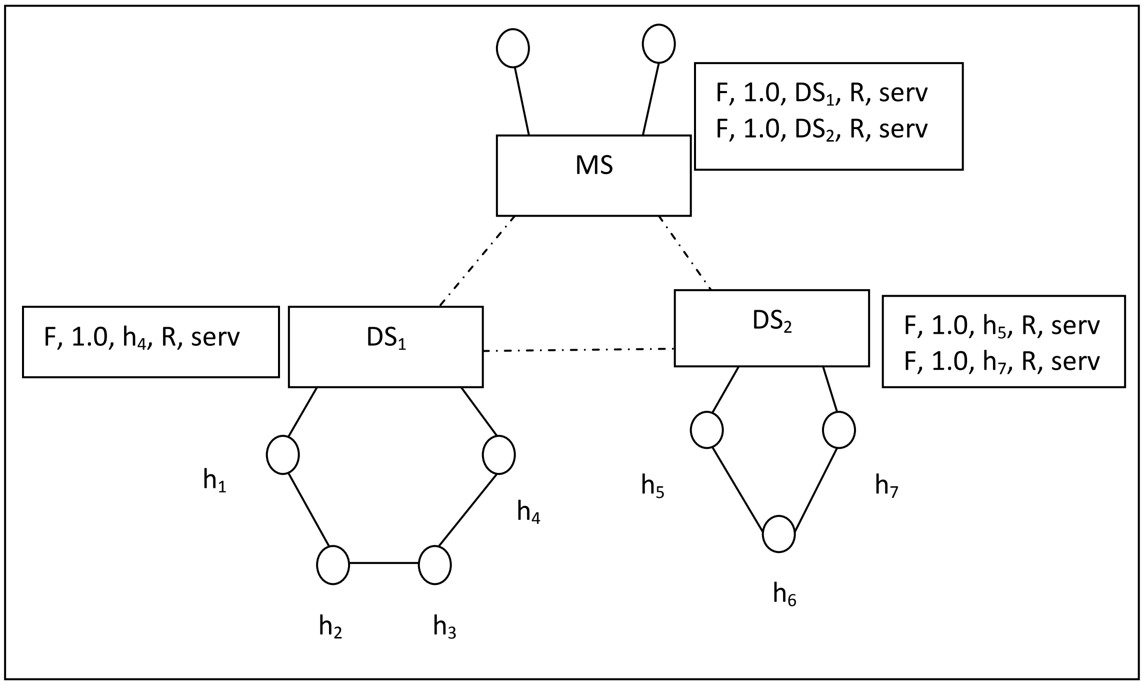

FIGURE 5: Repositories of MS, DS1 and DS2

In all the cases above, dirty data should be flushed to the server and lock on the file should be released by informing DS and MS. DS and MS should remove the respective records from the repository.

Before sending the dirty data to the server and releasing the write lock, no operation should be kept pending by the writer. If a request is received from another reader or writer during flushing of dirty data to the server, the writer needs to send a 'Negative Acknowledge' to the requesting client. Client understands that it has to delay its operation till it receives 'CanCache' or 'TempOwner' from the server.

4. DISCUSSION

In MADFS, if a host has obtained write lock on a file, no other host (within or outside the domain) can obtain read or write lock on same file. But if there is no write sharing this file system gives good performance.

Ownership-based cache consistency model is suitable for LAN. It allows write sharing. But it does not handle file accesses in WAN.

The proposed file system allows write sharing within a domain. The performance of this file system will be the same for simple reads and writes as in case of MADFS because no change is proposed in the working from this point of view. In case of write sharing, however, the performance will be slightly poorer than MADFS as the file will now be shared for reading/writing by other clients in the domain while a client has held it for writing. During write sharing load of the writer who is 'TempOwner' increases slightly because it has to satisfy the requests of other readers or writers within the same domain. Reader/writer clients (not 'TempOwner') will not have the copy locally available in cache, but will be accessed remotely in LAN, thereby degrading the performance slightly. However, the performance will be the same as ownership based file.

When a client within a domain asks for a file, only for the first time DS has to contact MS and then fetch it from the file server. Any further requests for the same file within that domain are handled by DS. Thus communication overhead is reduced. Also, as the file is available in local cache memory, while being read by multiple clients across different domains simultaneously, the communication overhead is reduced. Thus the protocol works very efficiently [7].

During write-sharing, only one client ('TempOwner') holds the file and rest of the clients reading (or writing) the file, access it from 'TempOwner'. In this case, there is some amount of communication overhead but consistency is maintained.

5. CONCLUSION AND FUTURE WORK

This proposed technique, as discussed above, divides the network into domains. It allows write-sharing within a domain. It does not degrade the performance of MADFS in case of only writing or reading. In case of occasional write sharing, there is a slight degradation of performance.

If the write sharing should be allowed outside domain, then 'TempOwner' rights should not be given to the individual host. They should only be maintained at DS level. Also an attempt could be made to maintain distributed Main Server, so that there is no bottleneck with the Main Server.

6. ACKNOWLEDGEMENT

We would like to thank Dr. Sadashiv Bhide for his constant encouragement. His comments were very helpful.

7. REFERENCES

1. Hennessy and Patterson. *"Computer Architecture a Quantitative Approach*". 3rd Edition.

2. George Coulouris, Jean Dollimore and Tim Kindberg. *“Distributed Systems Concepts and Design”.* Third Edition, Pearson Edition, 2006.

3. Andrew Tanenbaum and Maarten van Steen. *“Distributed Systems Principals and Paradigms”*. Prentice-Hall of India Pvt Ltd. 2007.

4. Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh and Bob Lyon. *“Design and Implementation of Sun Network Filesystem”*. Summer Techn Conf USENIX, 1985.

5. John Howard. *“An overview of the Andrew file-system”.* USENIX Winter Conference, 1988.

6. M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel and D. Steere. *“Coda: A Highly Available File System for a Distributed Workstation Environment*”. IEEE Transactions on Computer, 1990.

7. Jun Lu, Bin Du, Yi Zhu and DaiWei Li. *“MADFS: The Mobile Agent-based Distributed Network File System”.* 2009.

8. M. Nelson, B. Welch and J. K. Ousterhout. *“Caching in the Sprite Network File System.* ACM TOCS 1988.

9. ByungGi Hong and TaeMu Chang. *“Ownership based Cache Consistency Model in Distributed File System”.* IEEE Tencon 1993.

10. M Blaze and R Alonso. *“Towards Massive Distributed Systems”.* Proceedings of 3rd Workshop on Workstation Operating Systems, 92.