Replication in Distributed File Systems

By

Smita Hegde

Department of Computer Science, University of Texas at Arlington

Abstract

This term paper focuses on replication in distributed file systems and replicating Web documents. Replication is a key strategy for improving reliability, fault tolerance and availability in distributed systems. This paper gives a brief introduction to replication and the basic principles used for replication in DFS. The next section discusses the enormous load on the web servers and inability of the caching techniques lead to the demand for replicating web documents for a better, speedy, and efficient way of delivering information. This has lead to a lot of research in the areas of replication in web documents. This paper introduces the latest research conducted by Guillaume Pierre and others and introduces Globule, a platform which automates all aspects of replication.

1. Introduction:

The basic function of s file system is to provide a long time reliable storage. Similar to a local based file system, distributed file system (DFS) stores files on one or more computers called servers, and makes them accessible to other computers called as clients, on the intranet. A recent study by IBM researchers [15] emphasizes the advantages of DFSs which are briefly discussed below.

Reliability – DFS results in the elimination of the single point of failure that has dogged all timesharing systems. It also supports replication for all of its network services. If one of the servers becomes unavailable, a client automatically switches over to one of the replicated servers.

Availability – The wining feature of DFS is that its components are available to users at all times. Use of replication enables an administrator to do file system backups while the system is up and running. The replicated copy remains stable even while the user is changing the original file.

Fault Transparency or Tolerance - Components in a distributed system can fail independently. A file may be made more available in the face of failures of a file server if it appears on more than one file server. Availability is one of the main measures of the advantage of a replication algorithm.

Load Balancing - Replication of files in a DFS allows better load balancing.

Performance - Another advantage of a distributed file system is the ability to share information with many diverse users. DFS is an efficient, extensible system. In addition, the server tracks which clients have cached copies of files, reducing the need for the client to constantly query the server, as well as reducing the network and server load.

2.1 Replication and Replication Techniques:
The main idea in replication is to keep several copies or replicas of the same resources at various different servers. A client can contact any of these available servers, preferably the closest, for the same document. This helps in reducing server load, access latency and network congestion [18]. Some of the important factors for replication are to maintain consistency among the replicas, to find the best replica server for each client and also keep it transparent to the users.

There are number of technique for file replication that are used to maintain data consistency. Replication services maintain all the copies or replicas having the same versions of updates. This is known as maintaining consistency or synchronization [3]. Replication techniques to provide consistency can be divided into two main classes: [10]

- Optimistic- These schemes assume faults are rare and implement recovery schemes to deal with inconsistency
- Pessimistic- These schemes assume faults are more common, and attempt to ensure consistency of every access.

Schemes that allow access when all copies are not available use voting protocols to decide if enough copies are available to proceed.

2.1.1 Pessimistic Replication
This is a more conservative type scheme using prime site techniques, locking or voting for consistent data update. As this approach assumes that failure is more common it guards against all concurrent updates. An update cannot be written if a lock cannot be obtained or if majority of other sites cannot be queried. In doing so you sacrifice data availability. The pessimistic model is a bad choice where frequent disconnections network and network partitions are common occurrence[3]. It is used for more traditional DFS.

2.1.2 Optimistic Replication
This approach assumes that concurrent updates or conflicts are rare [3]. This scheme allows concurrent update, updates can be done at any replica or copy. This increases the data availability. However, when conflicts do occur, special action must be taken to resolve the conflict and merge the concurrent updates into a single data object. The merging is referred to as conflict resolution. When conflicts do occur, many can be resolved transparently and automatically without user involvement [3]. This approach is used for mobile computing.

2.2.1 Replication Reconciliation –Updates and modifications must be propagated to all replicas. This can be done immediately when the update occurs or it can be done at a scheduled interval later. Immediate propagation to all the replicas is fast but it is expensive to do so if it is not important. Alternatively updates can be done later, more like a batch process. This is a periodic reconciliation, which allows propagation to occur when it is cheap or convenient. [Rumor] In systems which have disconnected operations, periodic reconciliation must be supported, as the immediate reconciliation will fail when the systems is disconnected.

2.3 Replication Models

There are three basic replication models the master-slave, client-server and peer-to-peer models.

2.3.1 Master-slave model

In this model one of the copy is the master replica and all the other copies are slaves. The slaves should always be identical to the master. In this model the functionality of the slaves are very limited, thus the configuration is very simple. The slaves essentially are read-only. Most of the master-slaves services ignore all the updates or modifications performed at the slave, and "undo" the update during synchronization, making the slave identical to the master [3]. The modifications or the updates can be reliably performed at the master and the slaves must synchronize directly with the master.

2.3.2 Client-server model

The client-server model like the master-slave designates one server, which serves multiple clients. The functionality of the clients in this model is more complex than that of the slave in the master-slave model. It allows multiple inter-communicating servers, all types of data modifications and updates can be generated at the client. One of the replication systems in which this model is successfully implemented is Coda. Coda is a distributed file system with its origin in AFS2. It has many features that are very desirable for network file systems [9]. Optimistic replication can use a client-server model. In Client- server replication all the updates must be propagated first to the server, which then updates all the other clients. In the client-server model, one replica of the data is designated as the special server replica. All updates created at other replicas must be registered with the server before they can be propagated further. This approach simplifies replication system and limits cost, but partially imposes a bottleneck at the server [11]. Since all updates must go through the server, the server acts as a physical synchronization point [13]. In this model the conflicts which occur are always be detected only at the server and only the server needs to handle them. However, if the single server machine fails or is unavailable, no updates can be propagated to other replicas. This leads to un-consistency as individual machines can accept their local updates, but they cannot learn of the updates applied at other machines.
In a mobile environment where connectivity is limited and changing, the server may be difficult or impossible to contact, while other client replicas are simple and cheap to contact. The peer model of optimistic replication can work better in these conditions [13].

2.3.3 Peer-to-peer model

The Peer-to-peer model is very different from both the master-slave and the client-server models. Here all the replicas or the copies are of equal importance or they are all peers. In this model any replica can synchronize with any other replica, and any file system modification or update can be applied at any replica. Optimistic replication can use a peer-to-peer model. Peer-to-peer systems allow any replica to propagate updates to any other replicas [11]. The peer-to-peer model has been implemented in Locus, Rumor and in other distributed environments such as xFS in the NOW project. Peer-to-peer systems can propagate updates faster by making use of any available connectivity. They provide a very rich and robust communication framework. But they are more complex in implementation and in the states they can achieve [11]. One more problem with this model is scalability. Peer models are implemented by storing all necessary replication knowledge at every site thus each replica has full knowledge about everyone else. As synchronization and communication is allowed between any replicas, this results in exceedingly large replicated data structures and clearly does not scale well. Additionally, distributed algorithms that determine global state must, by definition, communicate with or hear about (via gossiping) each replica at least once and often twice. Since all replicas are peers, any single machine could potentially affect the outcome of such distributed algorithms; therefore each must participate before the algorithm can complete, again leading to potential scaling problems [3]. Simulation studies in the file system arena have demonstrated that the peer model increases the speed of update propagation among a set of replicas, decreasing the frequency of using an outdated version of the data [14].

3.1 Some examples of DFS:

3.1.1 Ficus:

University of California Los Angeles, USA Gerald J. Popek, Richard G. Guy, Thomas W. Page, Jr. and John S. Heidemann developed Ficus. Ficus is a replicated general filing environment for Unix intended to scale to very large networks. The Ficus system employs an Optimistic, Peer-to peer, one copy availability model in which conflicting updates to the file system's directory information are automatically reconciled, while conflicting file updates are reliably detected and reported. The system architecture is based on a stackable layers methodology, which permits a high degree of modularity and extensibility of file system services [2].

3.1.2 Rumor:

The File Mobility Group from University of California Los Angeles developed Rumor. It is a software package that allows users to keep multiple copies of their files synchronized on different machines automatically. Rumor is an optimistically replicated file system designed for use in mobile computers. Rumor uses a peer model that allows opportunistic update propagation among any sites replicating files. It detects changes made to any files under its control and propagates the changes made at any replica to all other replicas. Rumor permits users to update any of their replicas freely, while guaranteeing that the updates will be properly propagated to all replicas.

3.1.3 Coda:

Coda is a distributed file system with its origin in AFS2. It is an advanced networked file system. The systems group of M. Satyanarayan in the SCS department has developed it at CMU since 1987. Coda is suitable for disconnected operation for mobile computing and for continued operation during partial network failures in server network. This system follows the Optimistic replication approach with a client server model. Coda provides replication flexibility akin to selective replication at the clients, but not at the replicated servers, which are traditionally peers [3]. The Coda clients cannot inter communicate directly as it follows the client-server model. Thus it is expensive if all the clients contact the server at one time.

3.1.4 Roam:

Roam is a Scalable Replication System for Mobile and Distributed Computing by David Howard Ratner Doctor of Philosophy in Computer Science University of California, Los Angeles, 1998 Professor Gerald J. Popek, Co-chair Professor W. W. Chu, Co-chair Mobile computing is rapidly becoming a way of life [3]. Roam is an optimistically replicated file system. It uses a peer-to-peer model that allows opportunistic update propagation among any sites replicating files.

4.1 Recent Developments in Replication for World Wide Web Access

The ever-increasing demand for the World Wide Web had not come with out its share of problems. The primary amongst them being the inability of resources to cope up with the ever increasing demands. Nowadays more and more applications use the web's infrastructure further aggravating the problem. All this has resulted in high latencies, low throughputs and huge amounts of network traffic. Hence a key research area in today's world is replication, which is been promoted as an elixir for the problems. The basic idea behind replication is to keep several copies of the same resource at different servers. The benefits associated with replication are reduced access times, balanced load among servers and prevention of repetitive transfers of documents over the same network links.

4.2.1 Problems with the Web

The success of world wide web has also given rise to some serious problems.The number of clients or users who access web servers have increases. This leads to the substantial requirements of network bandwidth to connect clients and servers. This increasing demand of bandwidth causes network congestion [18].

4.2.2 Caching

Caching of documents is an obvious solution to reduce network communication, server load and access latencies. The browser can cache recently used documents in memory or on disk. But a major issue with caching is to keep the cache up to-date, which is called cache coherence. There are various solutions to keep a cache coherent is the validation check mechanism, Data Replication Protocol (DRP) etc. Several enhancements to the basic caching scheme have been proposed. In the following subsections we discuss some of these proposals such as Stream based caching, Pre-fetching and Hierarchical Caching. The other option is to this is replication.

4.2.2 Replication

The basic replication strategies can be classified as replicas or caches. A replica site (also known as a mirror) always holds a copy of the document where as a cache site may or may not hold the document.

The paper "Differentiated strategies for replicating web documents" by Guillaume Pierre and others based on their results proposed that the idea of a "universal policy" is not as effective as the solution of having several specialized policies used together. Using some of the figures and graphs for their paper this can be further explained. The policy of replicating each document should be based on its individual characteristic such as the size of the document, its popularity, the geographical location of its clients and the frequency of updates. This will ensure that each document is replicated using the best-suited policy for that particular document. In this paper it is proposed that the web pages (or groups of pages) are encapsulated into distributed objects. This ensures that a replication policy is associated with every object. To take this a step further they also suggested creating web documents that are capable of selecting its own optimal policy or in other words creating adaptive web documents.

4.3.1 Autonomous System

In their study the clients were grouped based on the autonomous system (AS) that hosted them. ASes are basically used to achieve efficient worldwide routing of IP packets. An intermediate server is placed per AS, this may be either a replica or a cache and this was used to ensure that all the user in that particular AS are binded to the intermediate server of that AS. Clients that were in the same AS as the master server, accessed the master server directly as did the clients whose AS could not be determined.

4.3.2 Different Replication and Cache Techniques

For the sake of comparison the baseline configuration are given below.
NoRepl – No caching or replication. All clients access the master server directly.
Caching configurations use proxy caches in place of intermediate servers. The various configurations using the caching policy are

4.3.2.1 Caching Configuration

These use proxy caches in place of intermediate servers. The different polices used are
Check – In this technique the cache sends an If-Modified-Since request to the master whenever it gets a hit. By doing this it ensures that it checks the consistency of the document before answering the client's request.

Alex – In this technique whenever a copy it created a TTL value (Time to live) is associated with it. This value is proportional to the time elapsed since it was last modified. Until the expiration of the TTL the copy is sent to the clients without having to check for consistencies. After the expiration of the TTL the copy is simply deleted.

AlexCheck – This techniques is similar to the Alex technique expect for the fact that after the expiration of the TTL instead of just deleting the file the copy is still kept in the cache and a flag called "possible stale" is associated with it. Incase a hit occurs the cache sends an If-Modified-Since request to the master to check for the consistency of the document before answering the client's request.

CacheInv – In this technique the cache registers the copy at the server and this ensures that whenever the master updates the document, it send an invalidation to the registered caches asking them to delete the stale copies.

4.3.2.2 Replication Configuration

The replica servers on the other hand create permanent copies. Replica server allows stronger consistency policies that are not possible with caches. The replica servers are placed based on the fact that the maximum numbers of requests actually originate from a small number of ASes. As result the replica servers are places in ASes where the maximum requests come from. Hence the various replica configurations are

Repl10
Repl25
Repl50

These are decided by placing the replica servers in the top 10, 25 and 50 ASes respectively. The consistency is maintained by pushing updates.

4.3.2.3 Hybrid configuration

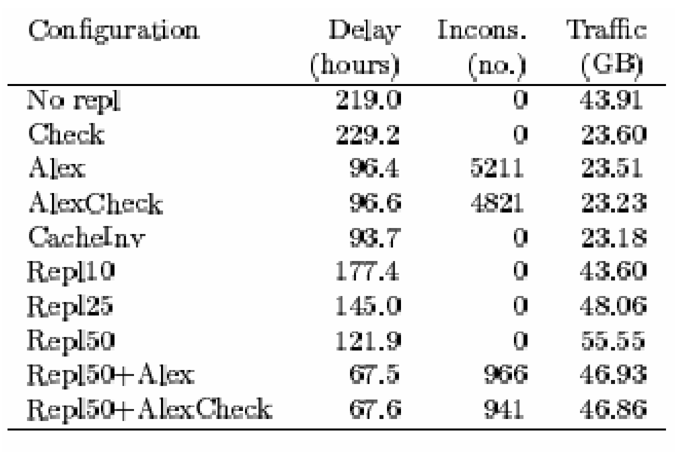
The third and final configuration for replication is Hybrid configuration. This is very similar to a replica configuration except for the fact that all ASes that do not have a replica have a cache instead. Based on the consistency policy of these caches there are two-hybrid configurations

Repl50 + Alex
Repl50+AlexCheck

4.4.1 Experiments Setup

The experiment consisted of simulating the replication of each document with each of the different configurations discussed above. One simulation per document and per strategy was used to measure the delay at the clients, number of clients who got stale copies, and the network bandwidth consumed. It was noticed that choosing a replication policy requires making tradeoffs in terms of access times, consistency of copies delivered to the clients, master server loads, the overall network traffic etc. It is impossible to optimize all these at the same time hence the metrics that were taken into consideration were

Total Delay
Inconsistency
Server Traffic



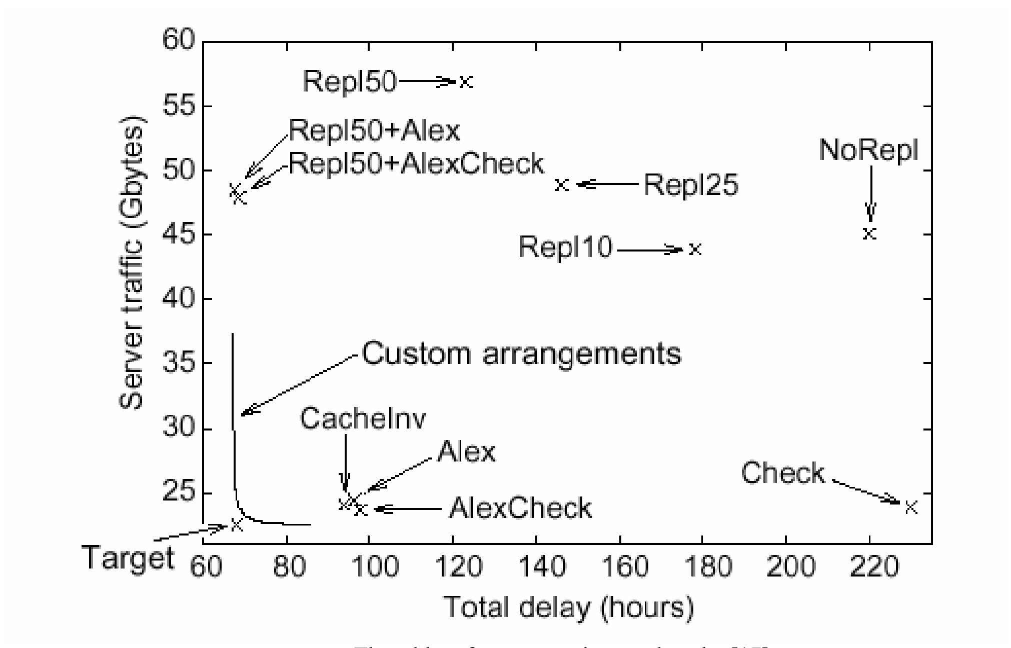
Configuration	Delay (hours)	Incons. (no.)	Traffic (GB)
No repl	219.0	0	43.91
Check	229.2	0	23.60
Alex	96.4	5211	23.51
AlexCheck	96.6	4821	23.23
CacheInv	93.7	0	23.18
Repl10	177.4	0	43.60
Repl25	145.0	0	48.06
Repl50	121.9	0	55.55
Repl50+Alex	67.5	966	46.93
Repl50+AlexCheck	67.6	941	46.86

The table refers to experiments done by [17].

The results of this paper suggested that most policies were good with respect to one or two metrics but none of the configurations actually managed to optimize all the three of them. For example Repl50+ Alex and Repl50+AlexCheck provide excellent delays but are not so good with respect to inconsistency and traffic.

4.5.1 Adaptive Strategy

So the best strategy for replicating all documents is finding the optimal strategy, which is the best compromise between the various metrics. On this basis it was concluded that if we define the target point as an ideal point based on two metrics (for the sake of convince the target point corresponds to the best achievable traffic and best achievable delay) and then plot the values of the various strategies based on a weighted sum of the evaluation metrics we observe that among the one-size-fits-all arrangements, some have good performance with respect to traffic but have poor performance with respect to delay. None of them get very close to the target point. But if the custom arrangements are mapped then it is observed that it is very close to the target if we compare them to the one-size-fits-all configuration. This implies that selecting replication strategies on a per-document basis provides a significant performance improvement over any one-size-fit-all configuration. This can be observed in the figure given ahead



The table refers to experiments done by [17].

Finally an adaptive strategy is proposed which uses the knowledge of the past accesses is used to determine which replication policy will be optimal in the near future.

5 Globule

An example an adaptive strategy architecture is discussed below from The paper "Globule: A platform for Self-replicating web documents" by Guillaume Pierre and Maarten van Steen discusses Globule a platform (designed Globule as a module for the Apache Web) server which is able to automate all aspects of such replication: server-to-server peering negotiation, creation and destruction of replicas, selection of the most appropriate replication strategies on a per-document basis, consistency management and transparent redirection of clients to replicas. To achieve its goal two things are taken care of first. First, Web servers need to be adapted so that they can support adaptive per-document replication policies. Second, servers need to cooperate to allow replicas to be dynamically installed and removed, and to redirect clients to the nearest replica. The document is considered as a physically distributed object whose state is replicated across the Internet. The 'best' policy is chosen from simulation outputs of performance metrics such as client retrieval time, network traffic and consistency using a cost function. The other concept involved is that of servers automatically negotiate for resource peering. The result of such a negotiation is for a "hosting server" to agree to allocate a given amount of its local resources to host replicas from a "hosted server." The hosted server keeps control on the resources it has acquired: it controls which of its clients are redirected to the hosting server, which documents are replicated there and which replication policies are being used. Even though resource limitation is enforced by the hosting server, the hosted server remains in control of its allocated resources. It does so by attaching priority flags to its documents, indicating how important each replica is. When making replacement decisions, the hosting server takes these priorities into account in addition to standard parameters such as the frequency of requests and replica size. The more important a replica, the less likely it is to be removed.

In terms of document replication a replica is made of two separate local objects: a document's content, which is available in the form of delivery components capable of producing documents, and a replication meta-object which is responsible for enforcing the document's replication policy. The issue of client redirection can be taken care by two methods which are

HTTP redirection: Browsers display the URL of the mirror site instead of the home site.

DNS redirection: The authoritative server to sends customized responses depending on the location of the client.

Currently like Akamai even Globule uses DNS redirection but this is flexible.

6 Streaming Video

It is suggested that the above framework can be used even for replicating streaming documents. In these cases consistency is often not a major issue, since these documents are hardly ever updated. On the other hand, since they are quite big (typically between 1 MB and 100 MB), partial replication often offers better cost-benefit ratio than total replication. Prefix caching (i.e., replicating only the beginning of each file) can help reducing the startup delay and smooth the subsequent bandwidth requirement between the server and the cache. The utility of such prefix caching is reinforced by the fact that many users stop movie playback after only a few seconds. Another possibility for partial replication is video staging, where only the parts of the variable-bit-rate video stream that exceed a certain cut-off bandwidth are replicated. Replication of layered encoded video can also be realized by replicating only the first N layers. This technique is of interest only if at least some clients use the ability of viewing a low-quality version of the video. Redirecting clients to appropriate replicas is more difficult with streaming documents than with Web documents. Streaming sessions last longer and use more resources at the replica site, in particular in terms of network bandwidth and disk I/O. This leads to complex scheduling and resource reservation problems at the server or at the replicas. However, many solutions have been proposed that range from clever scheduling policies to mechanisms for redirecting a client to another server during a streaming session. Clearly, streaming documents place different requirements on wide-area content delivery platforms than Web documents. Specific mechanisms are being developed to handle them efficiently. Moreover, these mechanisms do not apply to all situations, depending on the interactive or non-interactive nature of the documents, their compression formats, and the usage pattern of users. Research is still going on in this area.

7 Conclusions

Between the two approaches, Optimistic replication approach is generally followed in DFS. This is a trade off with consistency against availability, as it allows updates to take place even in the presence of communication failures at a cost of sometimes violating single-copy serializability. Although optimistic replication systems have been in use for some time, it is seen that there is still a lack of adequate metrics.

Client-Server model is generally used but when dealing with mobile users Peer-to-peer model is used. This allows direct communication and synchronization between all the peers but has scalability problems. Thus depending upon the system the replication approach and model is selected. As both have some trade off.

Due to the increase in the web, the single server approach is not enough to service all the requests. Thus replication or caching techniques are two solutions for the network congestion and latency time problems caused by single server. Similarly, even with web replication no single replication or caching policy is optimal for all documents. The Adaptive strategy with based-trace performance gives better results. The presented platform, Globule for Web documents replication is purposed to have dynamic creation and removal of replicas, consistency management and automatic client redirection. It is expected to meet the ever-increasing quality of service expectations that users have.

References

[1] Giacomo Cabri, Antonio Corradi, Giacomo Cabri, Antonio Corradi, Franco Zambonelli "Experience of Adaptive Replication in Distributed File Systems"- Copyright IEEE. Published in the Proceedings of EUROMICRO '96, September 1996 at Praha, Chzech Republic.

[2] Gerald Popek, Richard Guy, Thomas Page, John Heidemann "Replication in Ficus Distributed File Systems"- Proceedings of the Workshop on Management of Replicated Data, November 1990.

[3] David Ratner "Roam: A Scalable Replication System for Mobile and Distributed Computing"- UCLA Computer Science Department Technical Report UCLA-CSD-97044, January 1998.

[4] James Jay Kistler "Disconnected Operation in a Distributed File System"- Carnegie Mellon University Pittsburgh, PA 15213, May 1993

[5] Carl Downing Tait "A File System for Mobile Computing "-Columbia University 1993.

[6] Qi Lu "Improving Data Consistency for Mobile File Access Using Isolation-Only Transactions "- School of Computer Science Carnegie Mellon University
Pittsburgh, PA 15213, May 1996

[7] Eliezer Levy, Abraham Silberschatz "Distributed File Systems: Concepts and Examples "- ACM Computing Survey, Vol. 22, No 4, December 1990.

[8] APGen Online Documentation -
http://www.webgecko.com/products/apgen/docs.asp?page=replication.htm

[9] The Coda Distributed File System-
http://www.coda.cs.cmu.edu/ljpaper/lj.html

[10] Open Distributed Systems, Jon Crowcroft
http://www.cs.ucl.ac.uk/staff/J.Crowcroft/ods/ods.html

[11] Richard Guy, Peter Reiher, David Ratner, Michial Gunter, Wilkie Ma, Gerald Popek "Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication" –University Of California, Los Angeles

[12] An Overview of Projects onDistributed Operating Systems - **Alfred J. Lupper**
http://www-vs.informatik.uni-ulm.de/DOSinWWW/DistribSys.html

[13] T. Ekenstam, C. Matheny, P. Reiher, G.J. Popek "The Bengal Database Replication System" - the Journal of Distributed and Parallel Databases, Vol. 9, No. 3, May 2001

[14] A. Wang; P.L. Reiher; R. Bagrodia, "A simulation evaluation of optimistic replicated filing in mobile environments," Proceedings of International Performance, Computing and Communications Scottsdale, AZ, 1999

[15] White Papers File Systems in a Distributed Computing Environment - http://www.transarc.ibm.com/Library/whitepapers/OSF_Whitepapers/dfs.html

[16] Guillaume Pierre and Maarten van Steen "Globule : a Platform for Self-Replicating Web Documents," Internet Report IR-483, January 2001.

[17] Guillaume Pierre, Ihor Kuz, Maarten van Steen and Andrew S.Tanenbaum , "Differential Strategies for Replicating Web Documents".

[18] Kapil Rastogi , "Replication of Documents in the World Web Wide".