UNIX International

Distributed Computing Work Group

Distributed File System

White Paper

December 1991

Published by

UNIX International
Waterview Corporate Center
20 Waterview Boulevard
Parsippany, NJ 07054

For further information, contact:
Vice President of Marketing

Phone: (1) 201-263-8400
Fax: (1) 201-263-8401

International Offices:

UNIX International	UNIX International	UNIX International	UNIX International
Asian/Pacific Office	Australian Office	European Office	Pacific Basin Office
Shinei Bldg. 1F	Suite 1	25, Avenue de Beaulieu	Cintech II
Kameido	80 Berry Street	1160 Brussels	75 Science Park Drive
Koto-ku, Tokyo 136	North Sydney, NSW 2060	Belgium	Singapore Science Park
Japan	Australia		Singapore 0511
			Singapore
Phone: (81) 3-3636-1122	Phone: (61) 2-922-5341	Phone: (32) 2-672-3700	Phone: (65) 776-0313
Fax: (81) 3-3636-1121	Fax: (61) 2-929-0635	Fax: (32) 2-672-4415	Fax: (65) 776-0421

NOTICE:
UNIX International is making this documentation available as a reference point for the industry. While UNIX International believes that these requirements are well defined in this first release of the document, minor changes may be made prior to conforming products meeting these requirements being made available.

Table of Contents

1. EXECUTIVE SUMMARY

This document summarizes the UNIX International Working Group on Distributed Computing conclusions on future directions for Distributed File Systems (DFS). The group defined a 2-5 year phased plan of future functionality desired in a Distributed File System.

The DCWG studied the current and anticipated Distributed File System requirements from the following perspectives:

a) **Size of network** - It was determined that the DFS must operate efficiently in a Wide Area Network environment with as many as 100,000 nodes. This means that the DFS must be both scalable and provide availability in partitioned network conditions.

b) **Types of Consistency**- It was determined that the DFS must support environments in which multiple clients shared files in a read only, single writer/multiple readers, sequential use, private use or in a concurrent writer mode where writes occurred as often as 1msec.

c) **Type of Semantics** - It was determined that the DFS should evolve to full single site semantics.

d) **Interoperability** - Implementation of the DFS must be possible across a wide range of platforms, both UNIX Systems and non-UNIX Systems. Of particular importance is operation in PC/MAC/Desktop environments.

e) **Availability** - It was determined that the DFS must be highly available. The distributed computing environment should be able to switch automatically to a replica if the current instance of the file becomes inaccessible, if the current instance becomes overly loaded, or if the network becomes partitioned.

f) **Data Integrity** - It was determined that the DFS must provide high integrity by maintaining tight consistency. If multiple copies of a file exist, the file's integrity must be reconcilable following a network failure which resulted in copies of the file residing on different partitions.

g) **Security** - It was determined that the DFS must provide authentication, authorization, assurance and auditing to inhibit violation of the DFS.

h) **Administration** - It was determined that the DFS must provide easy to use administration.

i) **Extensibility** - It was determined that it was not possible to ship a DFS capable of meeting the needs of every possible distributed environment. Rather, SVR4 should be shipped with a set of basic functionality and provisions made such that the DFS protocol will not inhibit third parties from adding extended functionality.

These requirements are refined for internal consistency and described as a set of desired Distributed File System features in the DFS phased plan shown in Figure 1, page 2.

	Requirements	Features
Phase I	Provide Support for WAN's Large Number of Nodes Easier to Use	NFS w/ Caching NFS w/ Replication Better Administration Enhanced Naming System
Phase II	Huge Number of Nodes Full Consistency High Availability	Security Single Site Semantics R/W Caching w/ full consistency R/W Replication w/ full consistency
Phase III	Excellent Resource Use Integrity after Failure	Transparent File Migration Reconcilation upon Reconnection Partial File Caching and Replication Continued Operation after Partition Fully Mirrored R/W Replication

Figure 1 - DFS Phased Plan

2. INTRODUCTION

2.1. Background

The DCWG was created by the UNIX International Steering Committee. The Steering Committee provided the DCWG group with a charter to "Establish requirements for future Distributed File Systems." With this charter, the UNIX International members assigned leading industry experts in Distributed File Systems as representatives to the DCWG.

2.2. Participants

Companies whose representative actively participated in the DCWG activities and discussions are:

Aggregate Computing	CDC	Intel
KMart	Locus	NCR
NEC	Retix	SPARC Intl
Sun Microsystems	Texas Instruments	Unisys
Veritas Software		

Figure 2 - List of Attendees

The DCWG is chaired by Joan Wrabetz of Aggregate Computing.

2.3. Contents

This paper contains the following sections:

Section 1 Provides an Executive Summary of the document.

Section 2 Contains an introduction to the document.

Section 3 Summarizes the principle decisions of the DCWG on DFS.

3. PRINCIPLE DECISIONS

The DCWG studied the current and anticipated Distributed File System requirements from the following perspectives:

a) Size of network
b) Types of Consistency
d) Type of Semantics
e) Interoperability
f) Availability
g) Data Integrity
h) Security
i) Administration
j) Extensibility

From these perspectives the DCWG identified the following principle issues which need to be addressed:

a) Short Term vs Long Term Requirements of Distributed File Systems
b) DFS in a Large Distributed Environments Issues
c) Architectural Issues
d) Availability and Integrity Issues
e) Caching and Replication Issues
f) Single Site Semantics Issues
g) Data Security Issues

These were refined for internal consistency and described as a set of desired Distributed File System features in the DFS Roadmap shown in Figure 3, page 4.

	Requirements	Features
Phase I	Provide Support for WAN's Large Number of Nodes Easier to Use	NFS w/ Caching NFS w/ Replication Better Administration Enhanced Naming System
Phase II	Huge Number of Nodes Full Consistency High Availability	Security Single Site Semantics R/W Caching w/ full consistency R/W Replication w/ full consistency
Phase III	Excellent Resource Use Integrity after Failure	Transparent File Migration Reconcilation upon Reconnection Partial File Caching and Replication Continued Operation after Partition Fully Mirrored R/W Replication

Figure 3 - DFS Roadmap

3.1. Short Term-Long Term Requirements for Distributed File Systems

A Distributed File System is a protocol (e.g. NFS protocol) and a set of functions (e.g. read, write). It depends on the physical filesystem to implement those functions locally, and it depends on the transport protocol to distribute the DFS protocol. Thus, the capabilities of the DFS are limited by the underlying physical filesystem and the capabilities of the transport protocol. For example, unless the physical filesystem supports multiple versions of the same file, it is hard for the DFS to do so.

As Distributed Computing becomes more and more a part of everyday computing, new functionality will be required of the Distributed File System. As we start to add new functionality, NFS will start to run up against limitations which cannot be overcome without changes to the physical file system, the type of transport used or the DFS protocol.

The DCWG has identified a set of basic functionality that can be added easily to NFS with no or minor additions to the DFS protocol (Short Term), and a second set that would involve extensive changes to the existing NFS protocol (Long Term).

3.1.1. Short Term Requirements for Distributed File Systems

NFS is currently shipped with SVR4. NFS has installed over 1.5 million copies with over 300,000 additional copies being purchased each year. Since NFS will be the most widely used DFS for the foreseeable future, it will be necessary to understand the use of NFS and to establish a strategy for its continued support.

The DCWG noted that NFS has been successful for applications requiring loose consistency between multiple clients in which write coordination is controlled by some administrative (procedural, manual, or by some programmatic handshake) mechanism. Typical files which fall into this class are executables, read-only files, private files, and files used sequentially (if someone writes to the file, other users of the file wait or are locked out until the current write competes his use of the file).

Figure 4 on page 6 shows a graphical representation of the different manner in which files are accessed and highlights those which NFS are used for. The proportions vary from environment to environment. However, typically NFS can be used to access 75%-95% of all files used in a distributed environment.

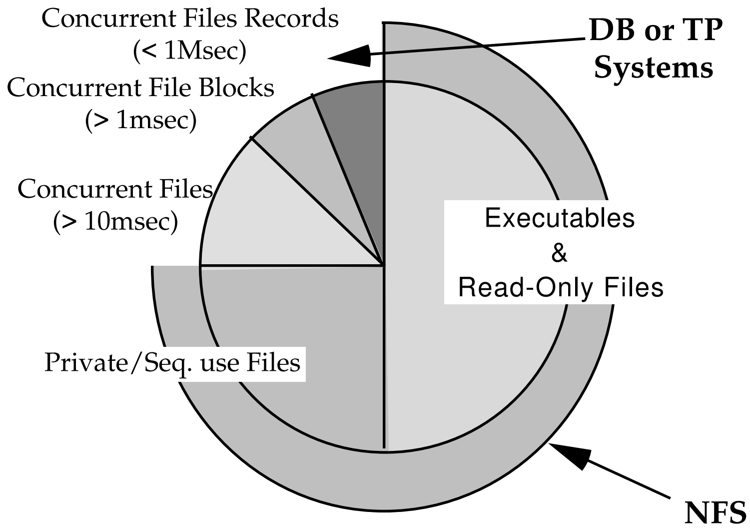

Figure 4 - Types of File Accesses

Because the large installed base and because NFS can be used for such a large percentage of the distributed files in the system, it is the DCWG's belief that NFS will continue to be a major DFS in the market place. Because of this the DCWG concluded that continued extensions to NFS must be made to meet the changing needs of the users of NFS.

The group concluded, that if NFS were given the following extensions, NFS would continue to provide excellent functionality for files requiring loose consistency between multiple clients.

1. Increase the number of nodes that NFS server will support by providing file caching on the client and support for replicating the file on multiple servers. Figure 5, page 7 shows a number of techniques for increasing the number of nodes that NFS can support and the expected increase in nodes supported for each technique.

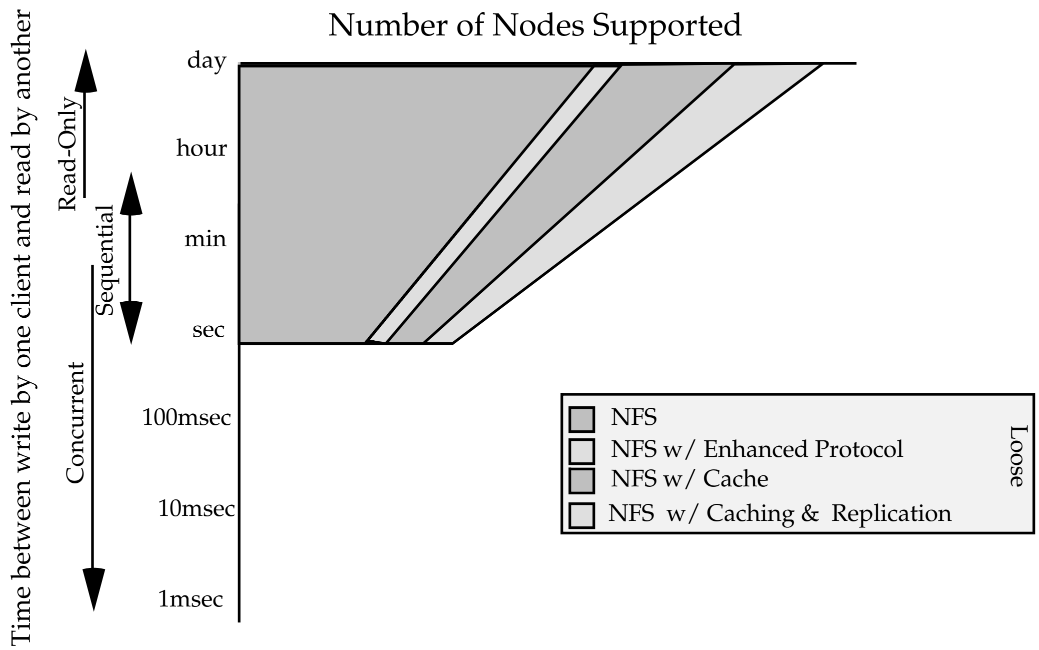

Figure 5 - NFS with Caching and Replication

2. Provide support for additional security authentication techniques besides DES, e.g., Kerberos.

3. Provide easier to use administration and installation of NFS, e.g., automounter enable by default, name service with default maps, etc.

4. Make NFS transport independent so it can be run over transports like OSI, Internet, TCP/IP etc.

5. Make NFS more robust by enhancing NFS to support larger files, handles, etc.

These enhancements to NFS may result in minor additions to the protocol. These enhancements must be forward compatible with current NFS. However, currently installed versions of NFS may not be able to fully utilize all of the new capabilities of the new DFS.

3.1.2. Long Term Requirements for Distributed File System

A future generation DFS should provide the following:

A. Single Site semantics -- Provide the same semantics for distributed files as for local files. This also implies the need to support remote special devices, such as is provided currently for local special devices.

B. Data Consistency -- Provide tight read-write consistency so that applications can update a file concurrently and guarantee that updates made by one application on one client will be seen by a second application on another client as soon as the second application performs a read.

C. Integrated with other Distributed Services -- The Distributed File System should be integrated with other distributed services such as remote processing at least for homogeneous environments.

D. Support for Large Distributed Networks -- This implies scalability, availability, administration, and naming capable of supporting 10 to 100,000 nodes. The following are required to achieve Large Distributed Network capability:

1) File caching on clients to increase the number of clients a server can support with acceptable response time.
2) File replication across multiple servers to increase the size of the network.
3) Administration features to allow administration of very large networks.
4) Integrated and federated naming viable for large distributed networks.
5) Support WAN transports like Internet and OSI.
6) Tightly mirrored replication for high availability.
7) Reconciliation of all copies of a file following reestablishment of a lost partition back into the network for increased data integrity.
8) Authentication, authorization, and auditing for assurance that the net’s security is not being violated.
9) Local file replicas, to gain faster response times then connecting through gateways, WAN’s and routers to a remote file.

Figure 6, page 9 shows the expected range of use of the future DFS with caching and the DFS with caching and replication. If an excessive number of clients try to write to the file, a DFS with full consistency may cause excessive network traffic to occur seriously limiting the number of nodes supported. For such situations a DB or a TP system will provide a better solution then a DFS.

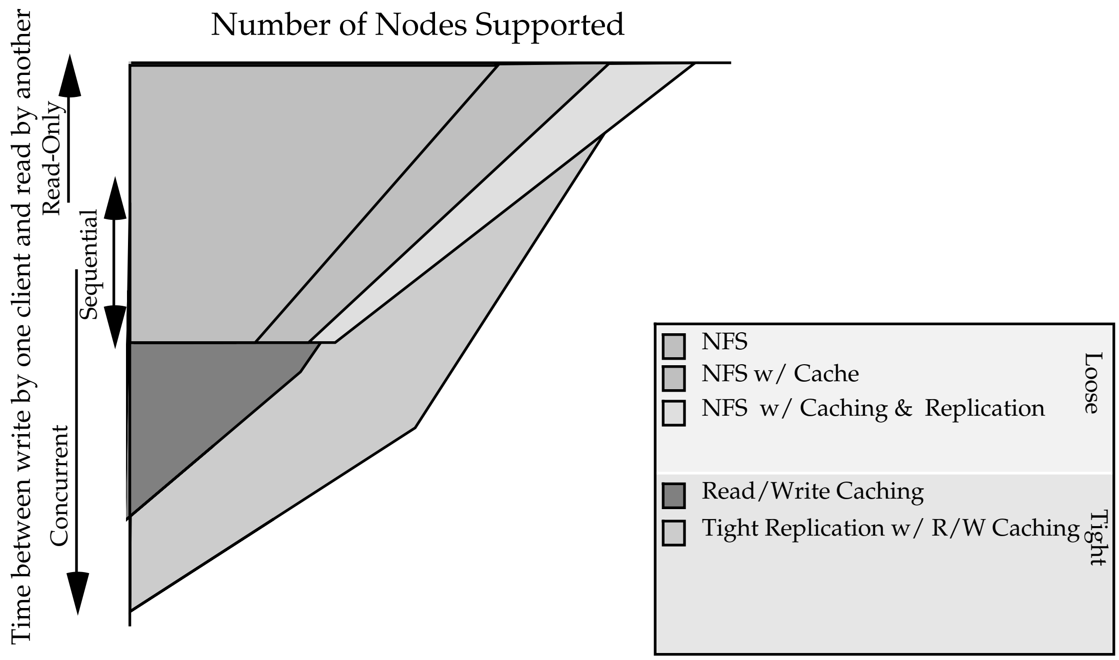

Figure 6 - Caching and Replication on a DFS with Full Consistency

This future generation DFS should add auxiliary layered protocols to the current DFS protocol so that the new DFS will continue to interoperate with older versions of the DFS protocol. If the new DFS supports tight consistency, interoperating with current versions of NFS may require that the future generation DFS access with loose consistency files shared with NFS. Therefore, if a user has an environment which requires strong consistency guarantees, he may wish to not interoperate with older versions of the DFS protocol.

3.2. Support Large Distributed Environments

3.2.1. Locality of the Distributed File System

Distributed File Systems are used in a number of environments. Some of these environments are composed of 10-100,000 machines dispersed geographically all over the globe. Others may consist of 10-100,000 machines all existing in a single office complex. Because of the diversity of these environments different connection mechanisms will be used. The cost of reliable data exchange and the cost of obtaining acceptable response times for file accesses will result in the following needs that the Distributed File System must fulfill:

1. Ability to keep a local copy of a file via replication and caching.

2. Ability to continue local operation, using a replicated or cached copy, should the network become partitioned/disconnected because of a failure or planned action.

3. Efficient mechanism for establishing sub-domains within the total net.

4. Intra-domain and inter-domain authentication and secure file access and data exchange.

5. Inter-domain naming and mapping capabilities.

6. Administration of distributed computing environments regardless of size or dispersity.

3.2.2. Size of Distributed Computing Network

The Distributed File System will be used in networks of 10-100,000 nodes. Large environments require additional functionality. The DCWG believes that a DFS must provide the following functionality for operation in large environments.

1. High availability.

2. Increased data locality.

3. Support 100,000 nodes.

4. Ability to perform most maintenance activity without stopping the system's use of the file.

5. Ability to administrate the entire system from a single site.

3.2.3. Heterogeneous Systems

SVR4 will be connected to non-UNIX Systems. It is necessary that SVR4 interoperate with these systems. The following capabilities will be needed.

1. A limited support of remote devices.

2. Support of large file handles.

3.3. Architectural Issues

3.3.1. Consistency during Concurrent Write Sharing

The DCWG concluded that some applications in a distributed computing environment will want to support multiple clients writing concurrently to distributed files. This will allow file semantics (read/write) to be used by applications which have a modest amount of concurrency. A typical application might be collaborative writing where two or more individuals are writing a document together.

Figure 7, page 12 shows that a DFS with tight consistency will allow applications to access data concurrently. This allows the commonly understood read/write file access mechanism to be used by programmers writing such applications. It also removes the need to know how to use or pay for a database for such applications.

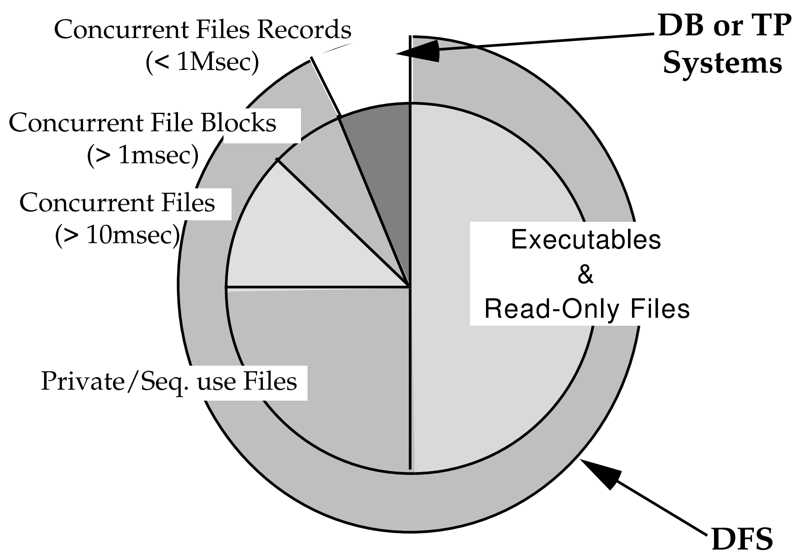

Figure 7 - Types of File Accesses

Files on a single system ensure that all writes to a file occur in the time order that they were issued, regardless whether or not the writes were issued by different applications or by the same application. Thus, tight consistency is maintained over the file. However in a distributed system maintaining the consistency of a distributed file is not an easy task.

NFS provides tight consistency between multiple writers which reside on the same client and loose consistency between clients. For a DFS to maintain tight consistency between clients, the DFS must know who issued a write and when. This may result in the DFS architecture maintaining state information.

3.3.2. Unified Caching and Replication Architecture

The key motivation for caching is to increase the number of nodes that a server can handle with an acceptable response time for file accesses.

The key motivation for replication is to increase the number of nodes that can exist in the network, to provide an alternate copy for availability purposes, to provide a nearby copy in WAN environments, to obtain faster a response time and to off load a heavily loaded server to improve the response time.

To support large distributed networks, the DFS must be provide caching and replication. Figure 8, page 13 shows graphically how caching and replication with different types of consistency increases the number of nodes supported over a DFS which does not provide caching and replication.

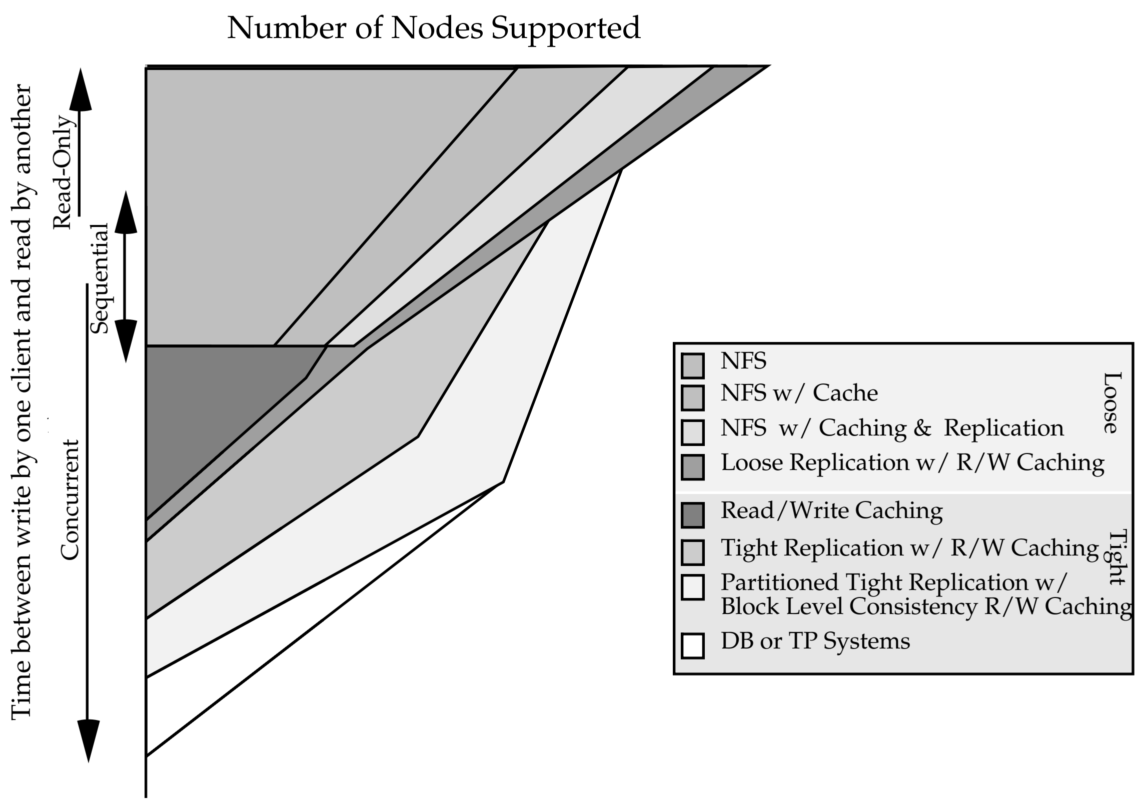

Figure 8 - DFS with Loose and Tight Caching and Replication

Because of the close interrelationship of caching and replication a coherent and consistent architecture must be provided. The caching and replication architecture must map the file caching and file replication into a unified form to reduce duplicated DFS code segments and to unify the administration (Phase III).

Both replication and caching have some common operations [particularly if it is determined that disconnected operation is achievable with caching]. In particular, the problem of consistency between replicas resembles the problem of consistency of a cache with its backing store. Such common operations should be handled by a unified DFS code segment.

The DCWG believes that the architecture to support file caching must be able to support caching with tight consistency with consistency possible at the entire file level, and the cache block level. However, SVR4 need only supply a DFS which provides cache consistency at the file level.

Similarly the DCWG believes that the architecture to support file replication must be able to support file replication at any of the following levels entire file, partial file or cache block. However, SVR4 need only supply a DFS which supports entire file replication and partial file replication.

3.3.3. Extensible DFS Architecture

A Distributed File System is a protocol (e.g. NFS protocol) and a set of functions (e.g. read, write). It depends on the physical filesystem, the VFS interface, a transport mechanism, a DFS protocol, out of band DFS messages, code providing DFS functionality, auxiliary decision and control code and administrative function.

The DCWG believes that the DFS in SVR4 should be able to be used to provide a wide range of functionality. However the SVR4 supplied DFS should only provide a basic set of functionality and be extensible by third parties or upgradable in later releases by simply popping on a stackable module in a manner similar to a STREAMS module.

To provide such extensibility, the DFS architecture must provide a general purpose mechanism which allows third parties to add on new functionality. This architecture should provide initially for static addition of new functionality, and then latter support more dynamic stacking of functionality. By providing such modularity and layering of functionality, a strong architectural foundation will be laid which will allow addition of simple functionality as compression or complex functionality as caching with equal ease and simplicity, Figure 9, page 16.

To provide such extensibility the following features should be provided:

1. Static addition of new stacks to the DFS

2. Dynamic addition of new stacks to the DFS

3. Provide a interposition mechanism to user resident code modules

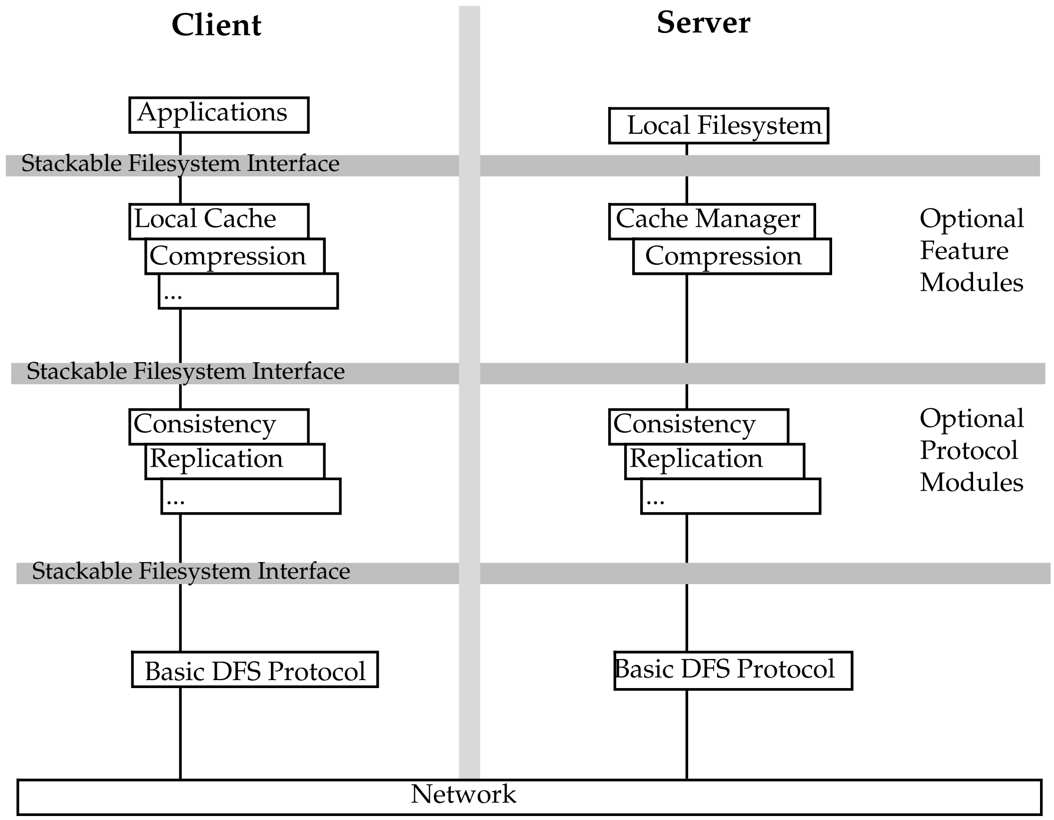

Figure 9 - Stackable DFS Architecture

3.4. Availability and Integrity

3.4.1. Availability of the DFS after Failure

As distributed computing becomes a basic mode of operation of most enterprises, it becomes imperative that simple malfunctions do not prevent the enterprise from operating. In addition, as distributed computing environments grow in size, the likelihood and cost of a malfunction will increase. It is therefore imperative that Distributed File Systems able to be configured to meet high availability.

To meet the needs of high availabilities, a DFS should have replication capability. File replication provides and alternative copy of the file if the current copy becomes inaccessible for some reason. This allows switching to the replica upon failure (failover) and allow the operation to continue.

The DCWG believes that the SVR4 Distributed File system must provide the following types of replication services to provide good system availability.

1. Loose/Tight Replication for environments which will accept inconsistencies when failures occur, or write so infrequently that it is a high probability that all replicates are up-to-date.

2. Mirrored replication for environments that must be assured of the availability and integrity of a consistent alternate copy.

3. Automatic failover for the current version of NFS in SVR4.0 (could be nothing more than failover to a statically defined alternate location).

4. Full automatic failover to any available replica of a file replicated by the new generation SVR4 DFS.

5. Location broker to locate an alternate copy.

3.4.2. Integrity of the DFS after Failure

Availability of a file does not necessarily imply that the file is correct or has good integrity. As networks become larger, so will the likelihood that a network failure will partition the network into two separate partitions, each containing a replica of the file. This quickly raises the question of what to do if an application writes to one of the replicas and how the integrity of the two replicas is reestablished when the two partitions are reconnected.

Transaction and database systems have built basic recovery mechanism which provide integrity checks and rollbacks/roll-forwards. For distributed files to be used in situations which require that the file maintain its integrity after failure requires handling of partitioning and failure situations.

The following capabilities should be provided in a future generation DFS:

1. Provide the ability to continue after a failure by simply failing over to a replica. This is the simplest level of integrity. It simply assumes that no other partition is altering the file.

2. Provide versioning notification. Thus, if a file is updated then upon reconnection of the two partitions applications using the file will continue to use the proper version of the file.

3. Provide the ability to keep a write journal of all changes to the file. This will allow changes to be tracked and an audit function performed at a later time.

4. Provide automatic rollback and reconciliation of the replicas upon reconnection of the two partitions.

5. Provide a transactional file system. This provides TP completeness and recovery capabilities as an inherent part of the file system.

3.5. Caching and Replication Issues

3.5.1. Caching in a Distributed File System

Caching is used to increase the number of clients for which a server can provide acceptable performance. Client resident file caching is much like the caching used by hardware multiprocessors to reduce the memory and bus loads. Like hardware caches the type of caching used depends on the manner in which the file is used.

Figure 10, page 19 shows a breakout of the different way files are used. It should be noted that as we go around the circle that the real difference is in the frequency that a new client obtains permission to update the file. The proportions vary from environment to environment. However, typically NFS can be used to access 75%-95% of all files used in a distributed environment.

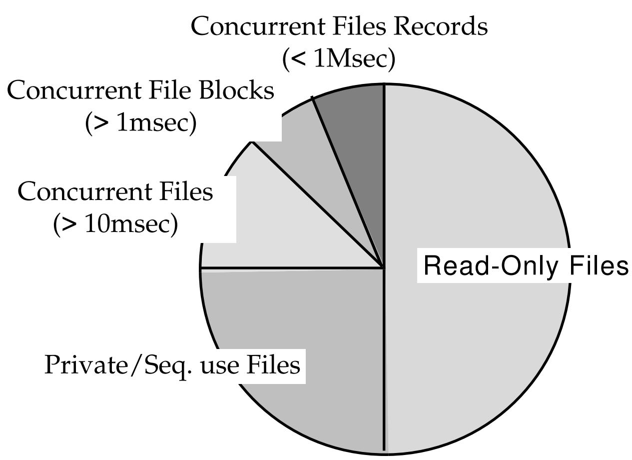

Figure 10 - Types of File Accesses

Read-Only files are typically binaries and executables. Since most accesses in any system are the execution of code it is not surprising that a large majority of all file accesses are to Read-Only Files. Read-Only Files are easy to cache since there is, in general, no need to worry about write traffic. There is however a need to provide a mechanism which will allow the Read-Only File to be updated to a new release or version. Updating requires that all clients which have cached the file invalidate their cache and begin using the new version of the file.

The second largest set of files are private files accessed by only a single user or client. This is a specific instance of a more general class where a file may be accessed by multiple users, however the use is sequential, i.e., dbedit. Thus, each user waits for the prior user to complete before using the file. Because of this sequentiality, it is not necessary to worry about other clients obtaining a prior version of the file, since no client is reading the file while another is writing. Again the only concern is to provide a mechanism which will allow the file to be updated to a new release or version.

Files that are used concurrently by multiple users, typically use different techniques to establish order on the writes to the file. The techniques depend on how fast new clients request the ability to write to the file. The shorter the time between concurrent use of a file by multiple clients the finer granularity of the size of date over which consistency is maintained, entire file, partial file, or cache block. However because of the extra overhead in maintaining such fine gained consistency, as the average time between writing by one client and the need to read the correct data by a second client becomes shorter the number of nodes supported becomes less and less.

For files that are accessed as read-only, as private files or sequentially, a DFS with loose consistency is sufficient. For files accessed in a concurrent manner, a DFS with tight consistency is required.

NFS is a DFS with loose consistency. One of the basic requirements is to increase the number of nodes that NFS supports. This can be done by simply enlarging the current NFS buffer cache. By increasing the NFS cache to encompass the working set of the files being accessed and using a good caching algorithm, we can increase the number of nodes that NFS supports by 2x-3x, see Figure 11, page 20.

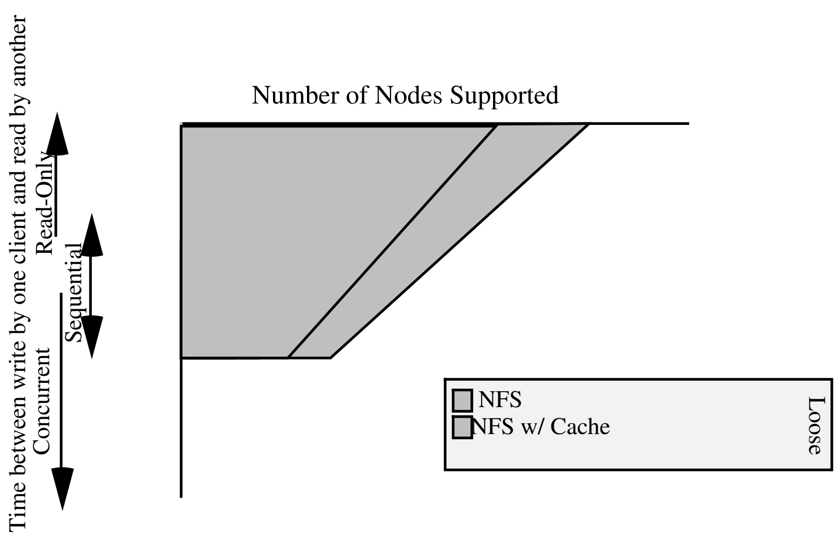

Figure 11 - NFS with Caching

For files which are used concurrently it becomes necessary to provide a Distributed File System that will support tight consistency. Tight consistency uses a technique similar to that used in hardware caching. Thus, as many clients can read and cache the file as wish to until some client wishes to write. Then all clients are told to invalidate their caches and a write permission is granted to the client who requested to write. Should another client request to write, then write permission is revoked from the first client and granted to the second. Similarly if some client wishes to read the file the write permission is revoked and the file becomes available for reading by any client.

A DFS with tight consistency makes the data available to applications on other clients sooner then does a DFS with loose consistency. Thus, allowing a higher level of collaboration between clients which use the same file, see Figure 12, page 21

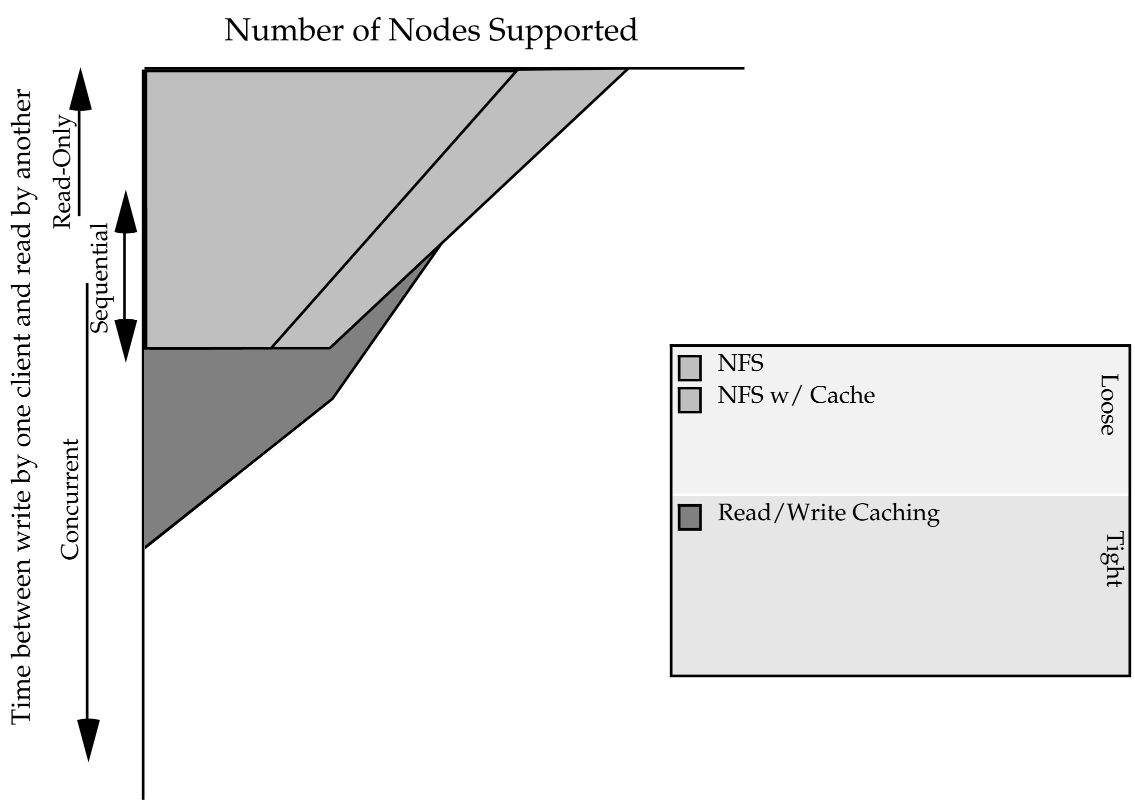

Figure 12 - DFS with Tight Consistency

This is great until clients begin to revoke/grant the write permission so fast that no real work gets done. Then it becomes necessary to break the file down to block or to the record level and revoke/grant write permissions at a finer level of granularity. This provides an even higher level of concurrency, see Figure 13, page 22

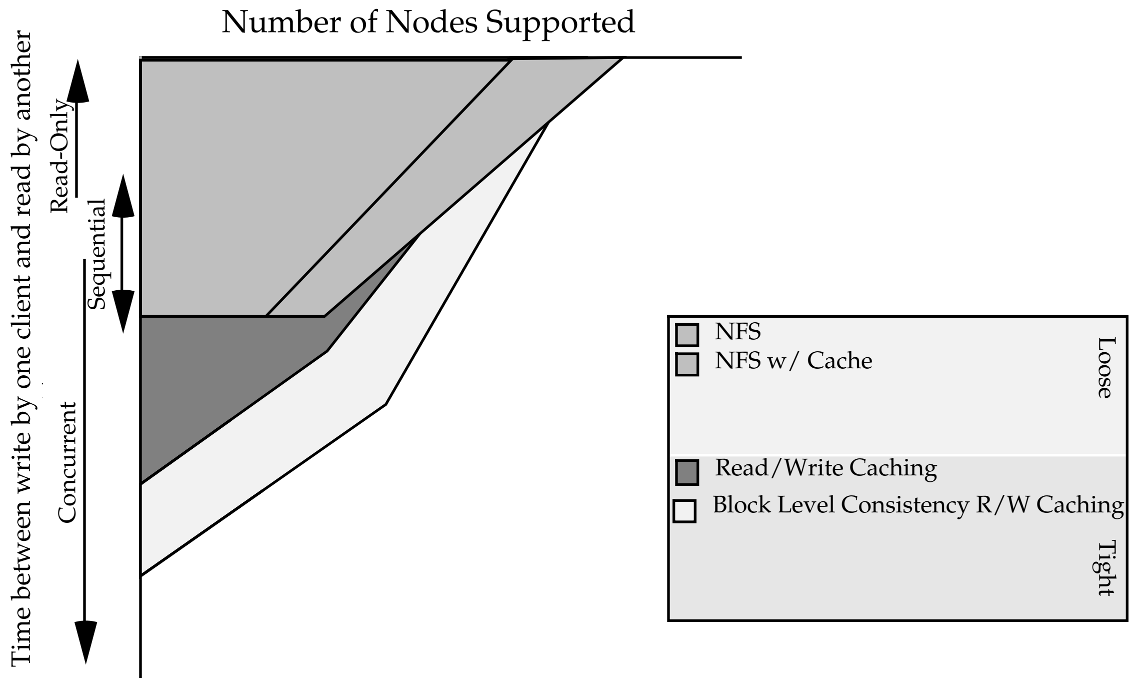

Figure 13 - DFS with Block Level Consistency

All of the tightly consistent caching techniques, assume that there is some server who is arbitrating and managing the revoking/granting of write permission. Eventually the overhead of moving the write permission becomes excessive to a single server. Then a distributed Database or distributed TP system, which maintains record or byte consistency and broadcasts the write to all clients using a two phase commit to update all of the client caches, becomes faster, see Figure 14, page 22.

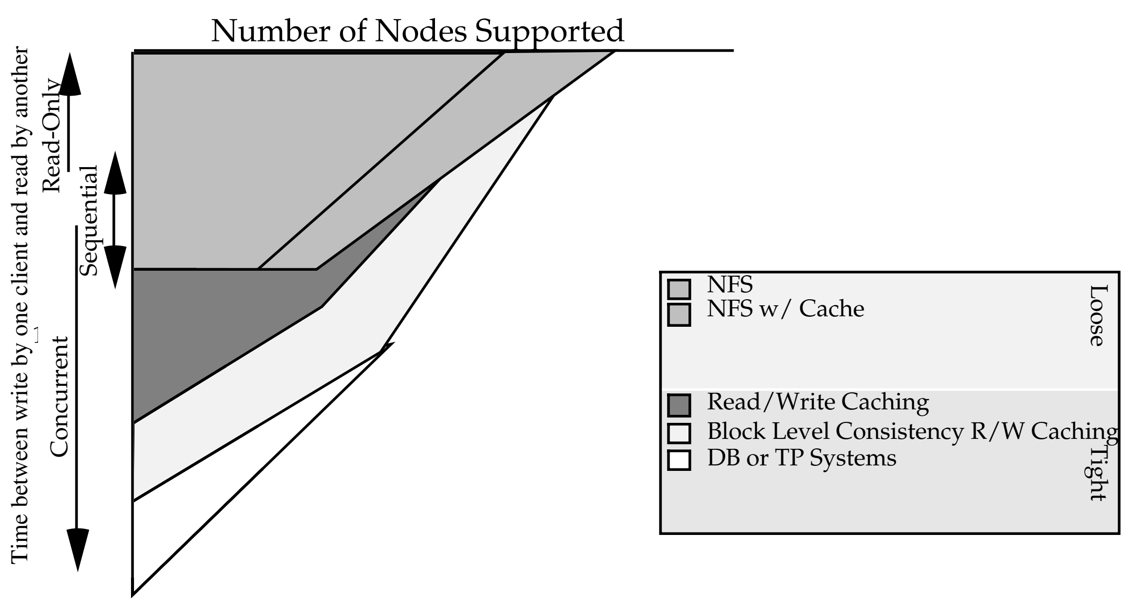

Figure 14 - DB or TP System Consistency

The DCWG believes that the architecture of the DFS should be able to provide caching with tight consistency by providing consistency at the block level. However, the DFS supplied with SVR4 need only provide caching with tight consistency by providing consistency at the file level.

The DCWG also recommends that NFS be extended to provide caching with loose consistency. The reason for this that caching should be added to NFS as a mid-life kicker for currently installed NFS sites.

Pinning a file in the cache prevents the file from ever being moved out of the cache once the file is placed into the cache. Pinning is desired for environments to prevent the file from being moved out of the cache during critical operations. For environments where continued operation after disconnection is desired, there needs to be an option which will allow the entire file to be cached and pinned so that the client does not cache it out.

The DCWG believes that caching will increase the number of nodes supported by any single server. However, in most environments a small percentage of nodes will be responsible for most of the load. Only nodes which are responsible for the network/server load should have to bear the memory and disk cost necessary to support caching. Thus, it is desirable that caching not be mandatory on all nodes which wish to access a file, only on nodes where caching will provide a benefit.

3.5.2. Replication on Distributed File Systems

Replication is used for a number of reasons. The key reasons why the DCWG believe that replication is an important part of any Distributed File System are:

1. Increase number of nodes beyond that provided by caching
2. Increase response time for clients in a WAN by providing a local copy
3. Increase availability by providing an alternate copy of the file

Depending on the environment and the manner in which the file is used, the priority of these three motivations change. For this reason, the DCWG believes there are market requirements for three types of consistency loose, tight and mirrored.

The three different types of replications (loose, tight, and mirrored) work in completely different environments and have different motivations for existing. However, a server may be used in all three environments at the same time or a file may be moved from one type of replication to another under administrative controls. So all three types of replications must be able to coexist without interfering with the others. It is highly desired that a single interface for replication exist. The replication technique itself should be a pluggable under VFS or a stackable item. The strength of the single interface is the ability to add on mechanisms for failovers, disconnection and reconciliation resolution, installation of new data files, etc. independently of the replication mechanism. It also allows new replication mechanisms to use existing replication services like failover, etc.

Replication with loose consistency will allow a large number of replicas to exist because the write traffic between replicas only occurs at fixed periodic times. This substantially reduces the server load that must be dedicated to maintaining consistency, leaving more for the clients to access files. Loose replication works well in environments that have read-only files, files used in a sequential manner, and files updated in a background mode by a single writer. This type of replication is also ideal for use in providing local copies to reduce the response time in WANs. NFS is sometimes used as a replication method with loose consistency, however it does not coexist with any form of caching today nor does it support failover.

Replication with tight consistency will allow a medium number of replicas to exist. When a write occurs, the write must be applied to all replicas before clients can read from their local replica. If write permission is requested by one of the clients, read permission must be revoked from all other clients. In WAN environments, it may take a while for all clients in the network to have their read permissions revoked. When the write session is ended, all writes by the writing client are propagated to all replicas. Once all replicas have the changes, reads can once again be dispersed among all of the replicas. A performance improvement can be obtained by allowing all writes to write through to a master replica and have the writes propagate to the other replicas in a background mode. Since there is a single consistent replica, all reads can be serviced by the master replica until the write session is over. This prevents the thrashing which would occur since the writing client will keep seizing write permission until it has finished its writing session and the reading nodes will keep taking write permission away. If combined with a caching mechanism using block level consistency checking, only clients reading from the same block that is being written to will need to reference the master copy. Even with block level consistency on caching, this will result in a high number of messages going to a single server. So the single server is likely to become swamped during the write period. Further performance improvement can be obtained by dedicating specific replicas to handle all write accesses for a specific section of the file. Thus, as long as the client is not reading from a section of the file that is being written to, the client's reads will continue to be serviced by the local replica that it normally works with. Again if combined with a caching mechanism using block level consistency checking, only clients reading from the same block of the section of the file being written to will need to reference the master copy.

Replication with mirrored consistency may only allow a small number of replicas to exist. When a write occurs, the write must be committed to by all replicas before clients can read from their local replica. Once all replicas have the changes, reads can once again be dispersed between all of the replicas. This may in some environment be the source of a major performance degradation. The reason for the performance degradation is that no write will complete until all replicas are written to and no read may take place until the write completes. A performance improvement can be obtained by allowing all writes to write to a single master replica which is mirrored. Reads then can be serviced by either the master-copy or its mirror until such time as the read is propagated to all replicas of the file, at which time reads can once again be serviced by any replica of the file. However for high frequency write environments, this may result in only two replicas ever existing. This may restrict the size of the network which can utilize a replication mechanism for high availability. Further performance improvement can be obtained by dedicating specific mirrored replicas to handle all accesses for a specific section of the file when a write occurs to that section. Thus, as long as the client is not reading from a section of the file that is being written to, the client's reads will continue to be serviced by the local replica that he normally works with. If combined with a caching mechanism using block level consistency checking, only clients reading from the same block of the section of the file being written to will need to reference the mirrored replica.

NFS is currently used to access replicated files. However, NFS provides no automatic consistency mechanism between replicas nor does it provides automatic failover to an alternate copy.

If a failover mechanism was provided on NFS, NFS could be used as a replicatable Distributed File System for read-only files. If automatic consistency between replicas were added to NFS, NFS would provide an acceptable DFS supporting loose consistency replication. This would allow NFS to support more nodes then the current implementation of NFS or NFS with caching, see Figure 15, page 26.

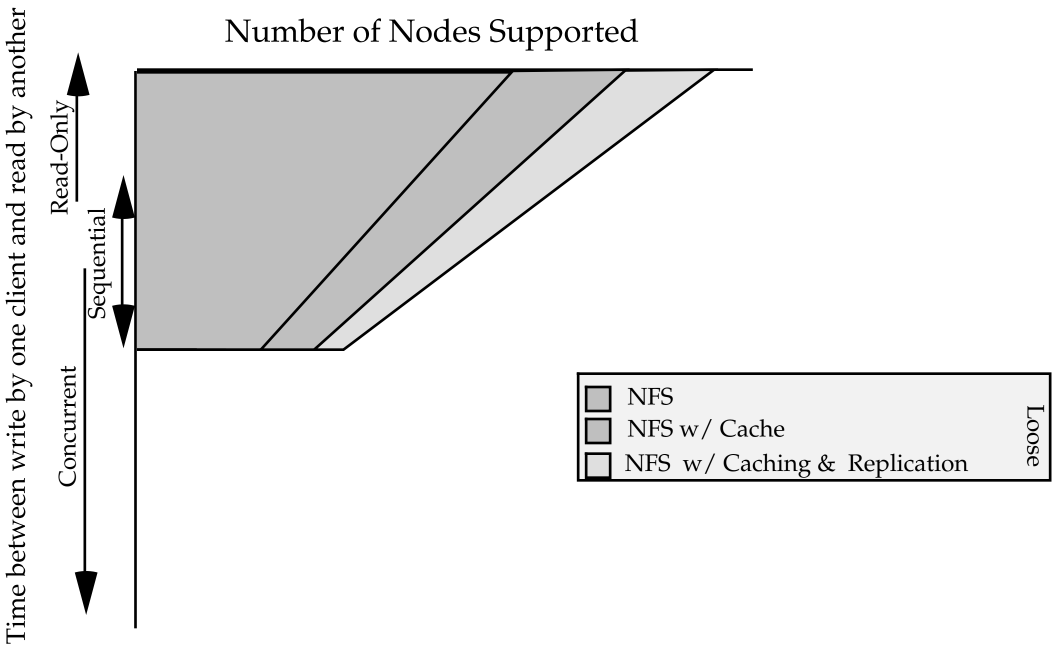

Figure 15 - NFS with Caching and Replication

Similarly, providing replication with tight consistency in a DFS which supported caching with tight consistency would allow more nodes to be supported, see Figure 16, page 26.

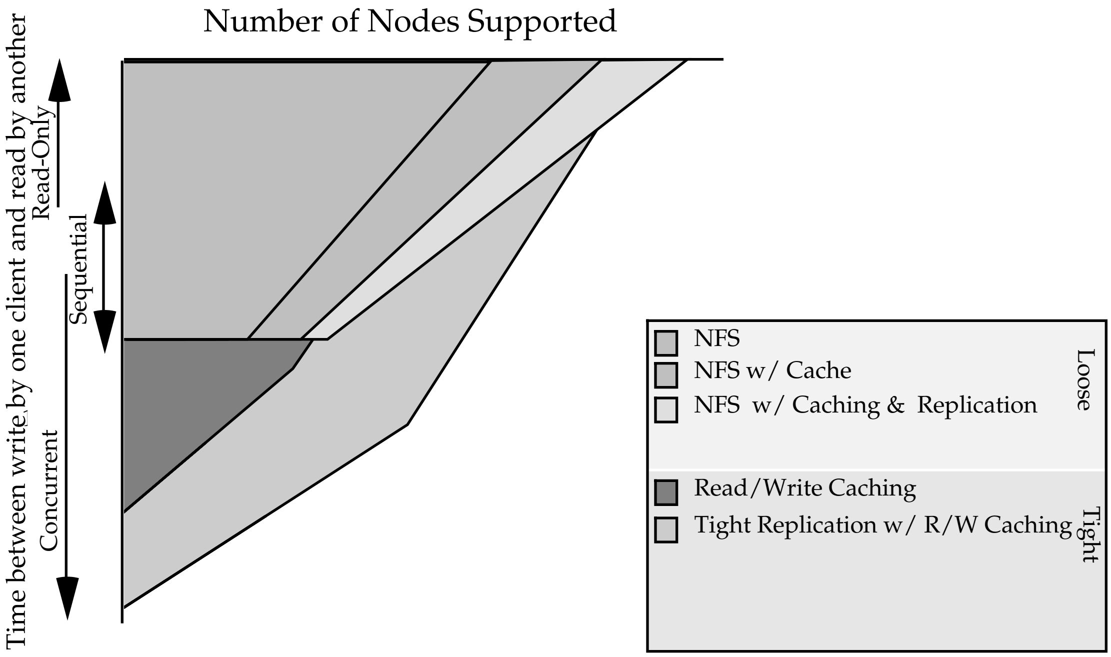

Figure 16 - DFS supporting Caching and Replication with Tight Consistency

A further increase in the number of nodes supported can be obtained by partitioning the file across multiple servers and directing all write traffic on a per partition basis, see Figure 17, page 27. This allows reads to other partitions to continue without any impediment.

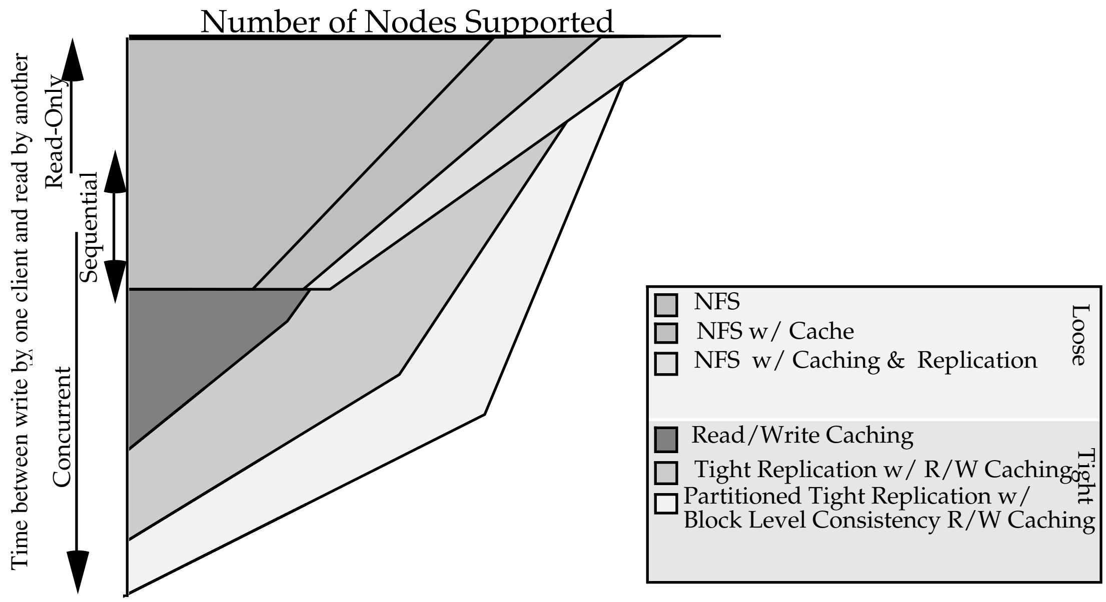

Figure 17 - DFS with Fine Grain Consistency

Further improvement requires going to a Database or a TP type system. Figure 18, below shows a complete mapping of all of the caching and replication mechanism that have been discussed in this section. Figure 18 also shows that a number of the mechanisms overlap or may supersede others depending on the phasing.

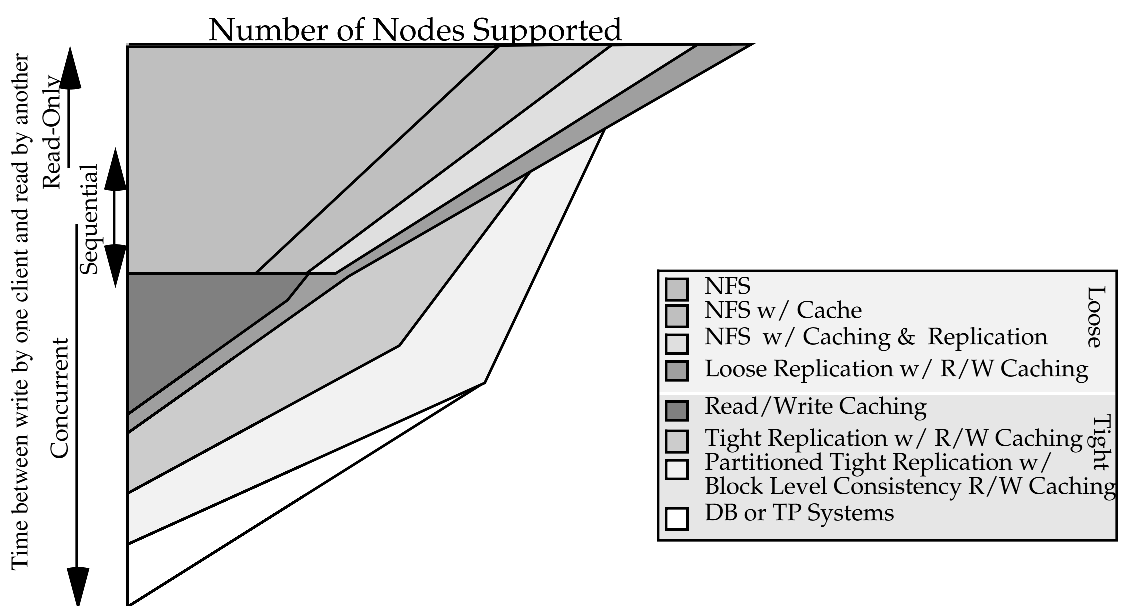

Figure 18 - Composite of Caching and Replication Mechanisms

The DCWG believes that only replication at the file system level need be supplied in SVR4. However the DFS architecture should be able to support replication consistency at the file or file partition level resulting in a further increase in the number of nodes that can be supported in the network.

Three types of replication must be supplied, replication with loose consistency, replication with tight consistency, and replication with mirrored tight consistency.

The DCWG believes that any Distributed File System with replication must also support failover because of failure, load balancing, etc. Such failovers must be performed automatically as well as under administrative control. Such failover needs to have a location broker which can provide the location of an alternative copy or a location where an alternative copy can be established.

Other modes of failure necessitate the need for continued operation after the network becomes partitioned. So the DCWG believes that to achieve the requirement of providing a highly available distributed computing environment, there is a need for continued operation after a planned disconnection or an unplanned network partitioning.

3.6. Single Site Semantics

Though NFS supports most file system semantics, the original design struck a fine balance between function and efficiency/cost in implementation. NFS does not support full semantics on the network, or more appropriately Single Site Semantics (also known sometimes as concurrent write sharing). The implications of such semantics touch on several key areas, which are not supported by the current NFS protocol and implementation. The following known areas of divergence within NFS inhibit NFS from providing full Single Site Semantics:

1. Concurrent write sharing with local file semantics is not supported (interleaved writes at the write() system call level, serialization of multiple readers and writers).

2. An interesting subset of concurrent write sharing is append mode writes, which is not supported for processes appending to the same file from different machines.

3. Delete on last close semantics for a file deleted while open is not generally supported. [Currently, NFS supports last close only for processes executing on a single node.]

4. Idempotency (exclusivity) for create, remove, rename, and link (see section 5.6.2).

The original design of NFS restricted the scope of the above semantics to processes executing on a single machine for efficiency and ease of implementation. Some of the "stateless" design of NFS involves the restriction of these features, reducing the ability to achieve Single Site Semantics in the current implementation.

3.7. Data Security

There has been a been a great deal of interest in security in the last few years. Over time new and better technology will be developed that provides an increased level of security. It is not the DCWG's intent that the requirements described in the section be the ultimate answer to security in the Distributed File Systems. Rather the DCWG recommends that the security in Distributed File Systems be constructed in such a manner that is it easily upgradable to whatever new technology becomes available without having to replace the entire Distributed File System.

The DCWG also recognizes that the Distributed File System is only one component in the entire system which needs security and that the real question which needs to be addressed is how does the Distributed File System's security integrate with the rest of the system's security capabilities. However, such a question is beyond the scope of the DCWG and should be addressed by a set of security experts. So the DCWG has established a set of recommendations from the viewpoint of the Distributed File System as input into such a group.

The DCWG believes that security for a Distributed File System should address the areas listed below, and has established requirement for each in the following pages.

1. Authentication and Identification that anyone/service/client requesting access to a file is who he says he is.

2. Verification that anyone/service/client requesting access to a file is authorized to access the file.

3. Assurance that anyone/service/client requesting access to a file is not being spied on, copies of his data are being made by looking at the transmission line, or someone has stolen his data and substituted something else.

4. Auditing and Reporting of sufficient events in such a manner that should some one violate security that we can detect the violation and then take steps to prevent further violation.