Study of Network File System(NFS) And Its Variations

Omkar Kulkarni[1], Deepali Gawali[2], Shweta Bagul[3], Dr. B. B. Meshram[4]

***Abstract*:**
A file system is the way in which files are named and where they are placed logically for storage and retrieval. The different operating systems have file systems in which files are placed somewhere in a hierarchical (tree) structure. Network file system is any file system that allows access to files from multiple hosts sharing via a computer network. This makes it possible for multiple users on multiple machines to share files and storage resources. NFS provides transparent and remote access to file systems. It is propitious to go for NFS as it can be machine and operating system independent, provides transparent access, provides high crash recovery mechanisms and it is highly Scalable. A Designer can fabricate adaptations in basic NFS and can use the new variation according to his need. Some of the variations of NFS are Replicated NFS, Parallel NFS, Web service based NFS, Mobile agent based NFS and gVault- Gmail based NFS etc. These variations can be used in different architectures and applications according to the users need.
In this paper, we have mentioned the basic Network file system (NFS) and a comparative study of different variations of it. On the basis of different features, we have made comparison between different variations of NFS and proposed a new NFS for a banking application.

Keywords: NFS, DFS, pNFS, variations, gVault

1. INTRODUCTION:

Although disks are becoming so inexpensive that workstations are seldom set up to be diskless, file servers are still becoming more and more crucial in modern network environments acting as centralized data sharing and storage stations. Computers use particular kinds of file systems to store and organize data on media. Any place that a PC stores data is employing the use of some type of file system. A file system can be thought of as an index or database containing the physical location of every piece of data on a hard drive. Since its inception in the mid-80's, the Network File System (NFS) protocol has been ubiquitously accepted as a means for file sharing across a network of compute nodes and across many operating systems simultaneously. Network File System (NFS) is a file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network in a manner similar to how local storage is accessed. Data storage is the most important component in a distributed or network system. It is critical that the file system is made to support data replication and can tolerate faults. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call(ONC RPC) system. The Network File System is an open standard defined in RFCs, allowing anyone to implement the protocol. In computing, a distributed file system or network file system is any file system that allows access to files from multiple hosts sharing via a computer network. This makes it possible for multiple users on multiple machines to share files and storage resources. The client nodes do not have direct access to the underlying block storage but interact over the network using a protocol. This makes it possible to restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed. A user can make some changes in the Basic NFS and can use it as per his requirements. There are different variations of NFS. Some of them are Replicated NFS, Parallel NFS, Web service based NFS, Mobile agent based NFS and gVault- Gmail based NFS etc. In Replicated NFS, replication and failure transparency are emphasized in the design of it with the help of primary backup model and distributed algorithms. Parallel NFS(pNFS) is now assumed as an emergent standard protocol for parallel I/O access in various storage environments purposefully designed for aggregating I/O bandwidth from many storage servers. Mobilele based NFS demonstrate that Web services can be used to construct a distributed file system, called the Web-services-based network file system (WSNFS). One of the goals of the WSNFS is to provide a platform for file sharing among heterogeneous distributed file systems. Mobile Agent based DFS(MADFS) is designed to gain a good performance on WAN. The objective of MADFS is to reduce the overhead of network transfer and cache management inherent to the distribution of a distributed files system in WAN. The MADFS organizes hosts into a hierarchical structure, and uses mobile agents as the underlying facility for transmission, communication and synchronization. gVault, a crypto-graphic network file system that utilizes

the data storage provided by Gmail's web-based email service.

2.LITERATURE SURVEY:

2.1.Basic NFS:

The NFS provides the transparent, remote access to file system[1]. Unlike many other file system implementation, the NFS is designed to be easily portable to other operating systems and machine architectures. It uses External data representation (XDR) specification to describe protocols in a machine and in a system independent way. The NFS is implemented on the top of the Remote procedure call (RPC) package to help simplify protocol definition, implementation, and maintenance. The NFS protocol was intended to be as stateless as possible. That is, a server should not need to maintain any protocol state information about any of its clients in order to function correctly. Stateless servers have a distinct advantage over stateful servers in the event of a failure. With stateless servers, a client need only retry a request until the server responds; it does not even need to know that the server has crashed, or the network temporarily went down. The client of a stateful server, on the other hand, needs to either detect a server failure and rebuild the server's state when it comes back up, or cause client operations to fail. This may not sound like an important issue, but it affects the protocol in some unexpected ways. We feel that it may be worth a bit of extra complexity in the protocol to be able to write very simple servers that do not require fancy crash recovery. Note that even if a so-called "reliable" transport protocol such as TCP is used, the client must still be able to handle interruptions of service by reopening [1]

The overall design goals of NFS were[1]:

1. Machine and Operating System Independence –The protocol used should be independent of UNIX so that an NFS server can supply files to many different types of clients. It should be designed to be easily portable to other OS and machine architecture.
2. Crash Recovery – when clients can mount remote filesystems from many different servers it is very important that clients be able to recover easily from server crashes
3. Transparent Access- NFS should allow programs to access remote files in exactly the same way as local files. No pathname parsing, no special libraries, no recompiling should be required. Program should not be able to tell whether a file is remote or local.
4. Reasonable Performance -The design goal is to make NFS as fast as or about 80% as a local disk. User will not want to use NFS if it is no faster than the existing networking utilities.

Architecture of NFS is shown in figure 1[1].

2.2.Basic Design:

The NFS design consists of three major pieces:

1.The protocol 2.The server side 3.The client side,[1]

2.2.1.The protocol:

The NFS protocol uses the Remote procedure call mechanism (RPC). RPC helps simplify the definition, organization and implementation of remote services. NFS protocol is defined in terms of a set of procedures, their arguments and results, and their effects. RPCs are synchronous i.e. client blocks until the server has completed the call and returned the results .

NFS uses a stateless protocol. The parameter to each procedure call contains all of the information necessary to complete the call, and server does not need to keep track of any past requests. This makes crash recovery very easy i.e. when a server crashes, the client resends NFS requests until a response is received. When client is crashed no recovery is necessary for either the client or server. Using a stateless protocol allows to avoid complex crash recovery and simplifies the protocol. New transport protocols can be plugged in to RPC implementation without affecting the higher level protocol code. The NFS uses User Datagram Protocol (UDP) and internet protocol (IP) for its transport level. UDP is unreliable, the packet can get lost, but as the NFS protocol is stateless, client can recover by trying the call until the packet gets through. The most common NFS procedure parameter is a structure called a file handler(fh or fhandle).It is provided by the server and used by the client to reference file[1].

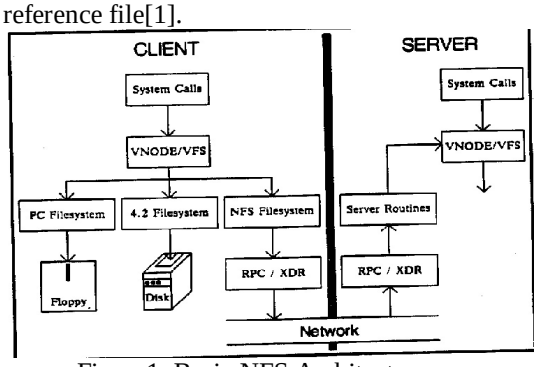

Figure1: Basic NFS Architecture

2.2.2. Server side:

As the NFS server is stateless, when servicing an NFS request it must commit any modified data to stable storage before returning results e.g. if a write request, then not only the data blocks, but

also any indirect block must be flushed if they have been modified[1].

To make the server work, the generation number in inode, and a filesystem id in the super block must be added. These extra numbers make the server to use inode number, inode generation number and filesystem id together as fhandle for a file. Server may handout the fhandle with inode number of file that is later removed and inode is reused. When original fhandle comes back, the server must be able to recognize that inode is reffering to different file or not[1].

2.2.3. Client Side:

Client side provides the transparent interface to the NFS. To make transparent access to remote files, a method to locate files that does not change the structure of path names should be used. Some remote file access schemes use host:path to name remote files. In NFS, hostname lookup and file address binding is done once per filesystem by allowing the client to attach a remote filesystem to a directory using mount program. So, client only has to deal with hostname once, at mount time. Transparent access to different types of filesystems mounted on a single machine is provided by a new file system interface in kernel. Each 'filesystem type' supports two sets of operations :The virtual filesystem (VFS) interface defines the procedure that operate on filesystem as a whole;and the Virtual Node (vnode) interface defines the procedure that operate on an individual file within that file system type

Figure 1 given is a schematic diagram of file system interface and how NFS uses it[1].

2.2.4.The Filesystem Interface:

[1]The VFS interface is implemented using a structure that contains the operations that can be done on a whole filesystem. Vnode interface is a structure that contains the operations that can be done on a node(file or directory) within a filesystem. There is one VFS structure per mounted filesystem in the kernel and one vnode structure for each active node. Using this abstract data type implementation allows the kernel to treat all filesystems and nodes in the same way without knowing which underlying filesystem implementation it is using. Each vnode contains a pointer to its parent VFS and a pointer to a mounted-on VFS. Thus, any node in a filesystem tree can be a mount point for another filesystem. A root operation is provided in the VFS to return the root vnode of a mounted filesystem. The root operation is used instead of just keeping a pointer so that root vnode for each mounted filesystem can be released. The operations defined for the filesystem interface are:

Filesystem operations:
--mount(): system call to mount filesystem
--mount_root(): mount filesystem as root
VFS Operations:
--unmount(vfs): Unmount filesystem
-- root(vfs) returns(vnode): Returns the vnode of filesystem root
-- sync(vfs):Flush delayed write blocks
Vnode Operations:
--open(vnode,flags): Make file open
--close(vnode,flags): Make file closed
--rdwr(vnode,uio,rwflag,flags): Read or write a file
--getattr(vnode) returns(attr): return file attributes
--access(vnode, mode):Check access permitions
--create(vnode,name,attr,mode): create a file
--remove(vnode,name):Remove a filename from directory
--rename(vnode,name,tovnode,toname): Rename a file
--link(vnode, todvnode, name):link to a file
--fsync(vnode):flush dirty blocks of file
--mkdir(vnode,name,attr): create a directory
--rmdir(vnode,name):Remove a directory
--strategy():Read and write filesystem blocks
--bread(vnode,blockno) returns(buf):read a block [1]

2.3.Implementation:

The first step in implentation was modification of the kernel to include filesystem interface. After making changes and appropriate running tests, kernel slows down upto 2% only, which is neglisible. Some of the filesystem routines must be rewritten completely to build NFS. Namei, the routine that does path name lookup, was changed to use the vnode lookup operation. The direnter routine, which adds new directory entries, also had to be modified to do directory locking during directory rename operations because inode locking is no longer available at this level, and vnodes are never locked. To avoid having a fixed upper limit on the number of active nodes and VFS structures, a memory allocater is added to kernel so that structures must be allocated and freed dynamically. A new system call, getdirentries, was added to read directory entries from different types of filesystems. Kernel readdir library routine was modified to use the new system call so programs would not have to be rewritten.After porting user level RPC and XDR libraries, kernel to kernel or kernel to user RPC calls can be done successfully. Once RPC and vnode kernel were

in place the, implementation of NFS is simply writing routines to do the NFS protocol, implementing an RPC server for NFS procedures in the kernel, and implementing a filesystem interface which translates vnode operations into NFS remote procedure call.The mount protocol was built into the NFS protocol. On the client side, mount command was modified to take additional arguments including filesystem type and option string. Filesystem type allows one mount command to mount any type of filesystem. Option string is used to pass optional flags to the different filesystem mount system calls. For example NFS allow two flavors of mount i.e. hard and soft. A hard mounted filesystem will retry NFS calls forever if server goes down, while a soft mount gives up after a while and returns an error[1].

2.4. Variations Of NFS:

2.4.1. Replicated NFS and transparency:

2.4.1.1. Introduction:

File servers acting as centralized data sharing and storage stations are very crucial in modern network environments. Failure and inefficiency of a file server would be unacceptable and thus data replication is necessary to provide fault-tolerant and high-performance services. In a network file service with replication, a cluster of two or more servers work together to provide the service. File information is replicated in whole or in stripes on the servers. Service will not stop with at least one server alive. The performance may degrade and the risk of losing data may rise as the number of servers decreases due to individual failures, but since repaired servers may join back into the service in time, the level of replication can be restored. Transparency in a replicated network file service consists of several major issues. In addition to location transparency and name transparency that most articles in designing distributed or network file services focus on, replication transparency, failure transparency, and performance transparency are also vital in a replicated file service. Location and name transparency are major considerations when designing a remote file access mechanism. Replication and failure transparency are required when the file service consists replicated servers in order to improve reliability. The client program may or may not have to know the identification of each server, while providing; transparent user access. In the presence of failures, the client program may or may not have to explicitly change its access to another server. It is crucial that the client programs do not have to know anything about replication and failures. Replication transparency and failure transparency are emphasized in the design. Any NFS client implemented on existing systems can use our fault-tolerant service without any modifications. In the suggested prototype, the simple primary-backup model is used to ensure data consistency between servers. Two distributed algorithms were designed to monitor and maintain the server modes. TOFF (Transparent Operation Fault-tolerant Filesystem) is presented, which is a replicated NFS that follows the Sun protocol[2].

1.1) The goals of the development of TOFF are:

1.fault-tolerance: supporting a file service that has high data availability and reliability

2.transparency: providing replication transparent and failure transparent service to users and client programs

3. standardized: using an industrial standard network protocol and file service protocol.

4. compatibility: allowing any client stations that support the protocol to use TOFF without any modifications

5.low cost and high performance: utilizing low cost personal computers as dedicated servers to decrease replication cost but with comparable or better performance as commercial workstation implementations.

2.4.1.2.File Consistency:

[2]In the earlier stage of the development of TOFF, they chose to use the simple primary backup model for server configuration. With considerations on performance, non modification requests are served only by the primary server and are not redirected to backup servers. Since the file images on all servers are totally consistent, no voting is necessary to obtain the up-to-date data. Under this model, several servers within the same subnet form a fault-tolerant file service group, with one of the servers being the primary server. The other servers are the backup servers. Normally, the primary server is the very first server that was started and already providing service. All servers started after the primary will try to join the group by registering to the primary. Once acknowledged, the new server enters a rebuilding stage, when it compares its file storage hierarchy and file contents with the primary server, updates or downloads the new files, and removes the files that no more exist. After the rebuilding process, the file system of the new server is consistent with the primary server, and the new server can become an official backup server, receiving redirected-requests from the primary server. On the other hand, the backup servers

monitor the state of the primary server and when they find that the primary server has failed, the backup servers can elect a new primary server with the "primary server election algorithm".

2.4.1.3.Transparency:
Replication transparency and failure transparency are the major goals of TOFF
With replication transparency, the clients do not have knowledge that the data is being replicated, and do not have to maintain the list of replicated servers.[2]
With failure transparency, the clients do not have knowledge that any server in the service group may have failed, even if it is the primary server
After the primary server fails, a new primary server is elected among the backup servers
How do the clients know where to send their requests when a new primary server appears?
Answer: by modifying RPC client program or by setting identical ethernet addresses and IP addresses on all servers in the same service groupBecause the machines used as servers are low-cost dedicated systems, and only the primary server replies to TCP/IP packets

The first approach is not desirable since the goal of TOFF is to be totally transparent to existing clients. IP multicasting can work, but in many currently existing systems, IP multicasting is not yet supported. Some NFS client designs use the replying source IP information for timeout control, and thus would not work with multicast servers. In TOFF, a special technique is used to solve this problem, by setting identical ethemet addresses and IP addresses on all servers in the same service group. Because the machines used as servers are low-cost dedicated systems, and only the primary server replies to TCPIIP packets, no compatibility and consistency problems would occur. The backup servers can all receive the client requests in order to monitor the primary server action, but they do not reply to the clients[2].

2.4.2. Parallel NFS:
[3]Since its inception in the mid-80's, the Network File System protocol has been universally accepted as a means for file sharing across a network of compute nodes and across many operating systems simultaneously Parallel NFS (pNFS) was further introduced as an extension of NFS to meet the bandwidth demands of these applications[3]. It aims for this goal by decoupling the data path of NFS from its control path, thereby enabling concurrent I/O streams from many client processes to storage servers. Compared to existing parallel file systems such as GPFS, Panasas, Lustre, and PVFS, pNFS stands out as the only open standard that is currently being developed and approved by many commercial and non-profit organizations[3].
Parallel NFS (pNFS) is touted as an emergent standard protocol for parallel I/O access in various storage environments. Several pNFS prototypes have been implemented for initial validation and protocol examination. Previous efforts have focused on realizing the pNFS protocol to expose the best bandwidth potential from underlying file and storage systems. We discuss its architecture and propose a direct I/O request flow protocol to improve its performance. Figure shows a typical pNFS.

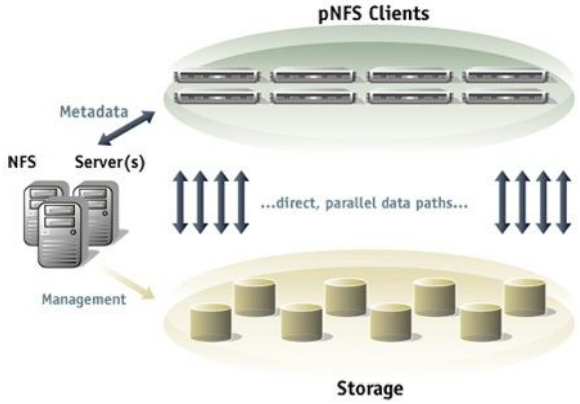

Figure 2:pNFS

2.4.3.Mobile Agent Based NFS:
2.4.3.1.Introduction[4]:
The conventional distributed file system is designed for LAN environment. They always play poor performance in WAN. Here, a novel distributed file system: The Mobile Agent-based Distributed File System (MADFS) is presented. The objective of MADFS is to reduce the overhead of network transfer and cache management inherent to the distribution of a distributed files system in WAN. The MADFS organizes hosts into a hierarchical structure, and uses mobile agents as the underlying facility for transmission, communication and synchronization instead of RPC call. Also a novel cache coherency mechanism for MADFS: Hierarchical and Convergent Cache Coherency Mechanism (HCCM) is presented. HCCM can effectively reduce the overhead of cache coherency management in distributed file system. The comparing results show that HCMM has better performance in WAN. In conclusion,

MADFS can achieve better performance and availability than conventional distributed file system in WAN.

2.4.3.2.The system structure of MADFS:

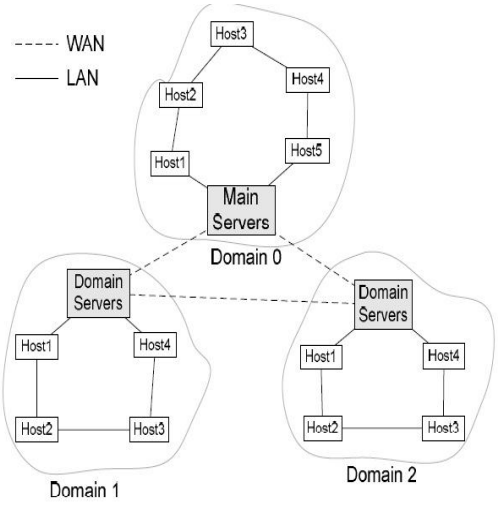

Figure The architecture of MADFS.

figure 3:MADFS[4]

Every server MADFS run the environment for mobile agent and the whole MADFS is a large platform for mobile agent

In MADFS, agent is in charge of transfer, communication and management

In MADFS, all agents can be classified as following[3]:

IA (Interface Agent). IA runs on the client host and accepts the file system calls sent by client. Then, IA processes these calls by dispatching, controlling or coordinating with other agents

WA (Working Agent). WA accepts orders or applications from IA, moves to the target server to execute file operation, and then return the result of execution

DMA (Domain Manage Agent). DMA is responsible for name, cache, property and space management and access control management in a domain. DMA can duplicate itself and actively move to target server in order to be close to the data to gain higher processing performance

MMA (Main Management Agent). MMA is responsible for the management and coordination of all DMAs in MADFS. MMA and DMA can cooperate with each other to accomplish the management work in MADFS

The advantages in the hierarchical architecture of MADFS are as following:

1)DMA is responsible for the management of a domain and MMA is responsible for the management of all the DMAs. This architecture can share all the overloads of communication and cache management over all DMA, and avoid the single central server to be the bottleneck of system. 2) Hosts in a Domain are connected with each other through LAN which has wide bandwidth and short transfer delay. Therefore, the communication in domain can gain a better performance by using the protocol designed for LAN. Meanwhile, the security operation can be properly reduced because that the servers in a domain are always inter-trusted, and then the more performance improvement can be obtained. When communication is across domains, MADFS could use the special protocols and mechanisms that are designed for WAN to gain better performance.

2.4.4. Web Service Based NFS:

[5]Web services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data between calling clients and servers. Web services can be used to construct a distributed file system, called the Web-services-based network file system (WSNFS). One of the goals of the WSNFS is to provide a platform for file sharing among heterogeneous distributed file systems.

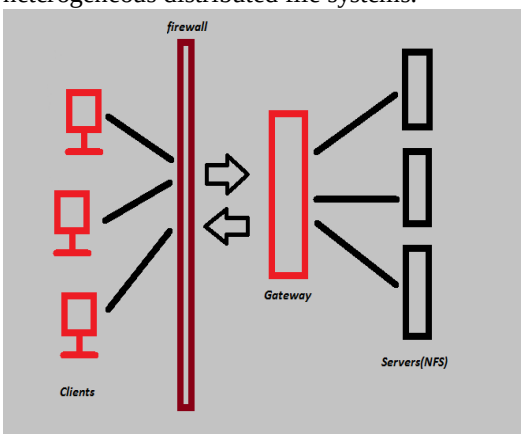

Figure 4:WSNFS[5]

Figure shows the network file system proposed which is called the Web-services-based network file system (WSNFS). A Web services file gateway is installed to overcome the problems associated with file sharing among heterogeneous file servers. All of the file directories of the file servers that are to be shared by the client are mounted on the client. The client sends file-operation requests to the gateway, which is responsible for forwarding each request to the appropriate file server and returning the result to the client.

2.4.5. gVault:Gmail based cryptographic NFS:

[8]gVault is a crypto-graphic network file system that utilizes the data storage provided by Gmail's web-based email service. Such a file system effectively provides users with an easily accessible free network drive on the Internet. gVault provides numerous bene¯ts to the users, including: a) Secure remote access: Users can access their data securely from any machine connected to the Internet; b) Availability: The data is available 24/7; and c) Storage capacity: Gmail provides a large amount of storage space to each user.

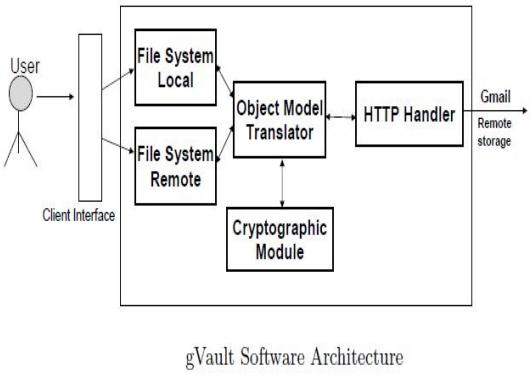

Figure 5: gVault[8]

The figure shows the overall architecture of gVault. The clients interact with gVault through the File System Local and the File System Remote components. The File System Local component provides a GUI interface to the local file system where the application is running. The File System Remote component provides a GUI interface to the file system stored at the remote server. The interface of both these components is similar to the interfaces that exist to any modern file system. The Object Model Translator maps the file system the user is outsourcing into data objects. The Cryptographic Module supports the object model translator in the cryptographic operations. The HTTP Handler translates file operations into HTTP operations that Gmail servers support.

2.4.6. IncFS An Integrated High-Performance Distributed File System Based On NFS

Scientific computing applications running in the cluster environment require high performance distributed file system to store and share data. A new approach, the IncFS , of building a high performance distributed file system by integrating many NFS servers is presented in this paper. The IncFS is aimed at providing a simple and convenient way to achieve high aggregate I/O bandwidth for scientific computing applications that require intensive concurrent file access. The IncFS uses a hyper structure to integrate multiple NFS file systems. And it provides multiple data layouts to effectively distribute file data among those NFS servers. Performance evaluations demonstrate that the IncFS has very good data access bandwidth with near perfect scalability, while still maintains an acceptable meta data throughput. The global view of IncFS is as shown in figure 6.

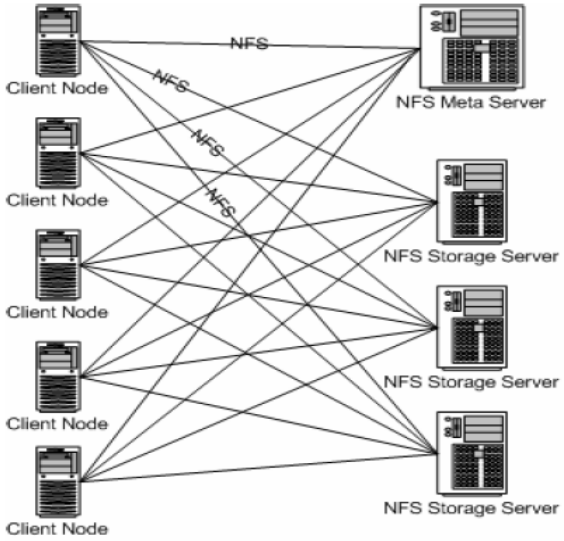

Figure 6: Global view of IncFS

3.Comparison Of All variations Of NFS studied above::

After studying different variations of NFS, their features and architecture, we can summarize the study and comparison as shown in the table 3.1

.

Table 3.1: Comparison of NFS variations

Variation / Property	Replicated NFS	Parallel NFS	WSNFS	MADFS	gVault
Transparency	High	Low	Medium	High	Mrdium
Scalability	Low	Medium	Medium	High	Low
Fault Tolerence	High	Low	Low	Medium	Low
Security	Low	Low	Medium	Medium	High
Efficiency	Medium	High	High	Medium	Low

4.Proposed System: Design of NFS for a Banking Application with the help of above studied variations :

4.1.Basic Information:

In our modern bank application, all the information about the transactions, customers, account details etc. is stored on a file server which acts as a centralized data sharing. These storage stations are very crucial in modern network environment. The information and data stored on remote server is accessed by remote users using NFS.

4.2. Steps to build Proposed NFS:

Step 1: The failure and inefficiency of file server would be unacceptable, thus data replication is necessary to provide fault tolerance and high performance service. So, first we used a primary server, which is the very first server, after starting of which, remaining backup servers can join the group. This provides high data availability and fault tolerance. For this the TOFF(Transparent Operation fault tolerant File system) is used. If any non modification request is coming from client side, the primary server can reply it without redirection or it may redirect the request to the backup server. But, if any modification request is coming, it will redirect it to backup servers. The primary server will be decided by selection algorithm. By this step we are providing the high transparency.

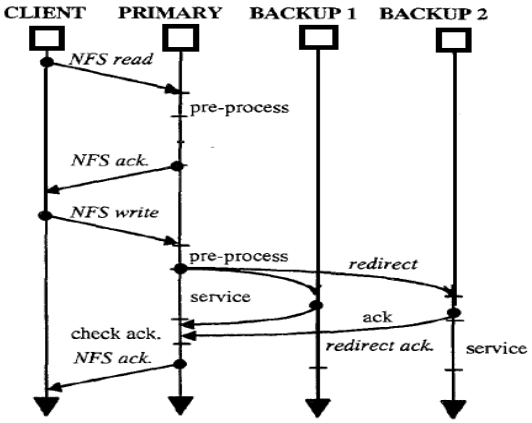

Figure7: step1(replicated NFS)

Step 2: In the implementation of NFS for this application, features of pNFS are added, where clients can access the storage devices directly and parallely. pNFS supports the 3 storage levels i.e. object level, file level and block level. It will improve the scalability.

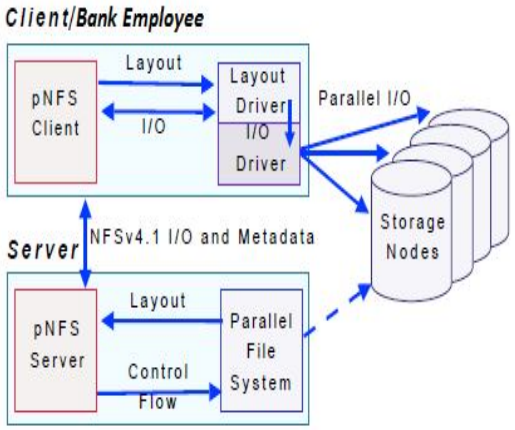

Figure 8: step 2(pNFS features)

Step 3: A web service file gateway is installed in the application, which is helpful to overcome the problems associated with the file sharing among the heterogeneous file servers. All the requests will be forwarded to the appropriate file server by gateway coming from the clients.

Step 4: Now, the most important issue is taken in the consideration i. e. security. The data is very crucial and important, so care should be taken to avoid the outside and inside attacks on data. In our application, a interface is provide for the user. User need to enter master password, a user name and a password for access of the files. In our designed NFS, a session is created after successful login of the user and it will work as HTTP client. All the file operations that the user performs at the client side is mapped to their equivalent HTTP requests. Our NFS implements the necessary cryptographic techniques to ensure the security of user data in the translation of file operations to HTTP requests.

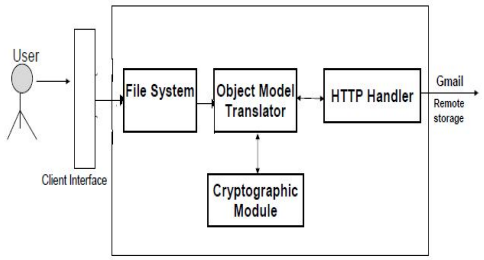

Figure 9 :step4(cryptographic features)

Step 5: NFS is implemented in such a way that it is supported in WAN. The hosts are organized into the hierarchical structure i.e. main server, backup server, domain server, clients etc. The mobile agents are used as underlying technology for transferring and communication of data.

5.Proposed Architecture:

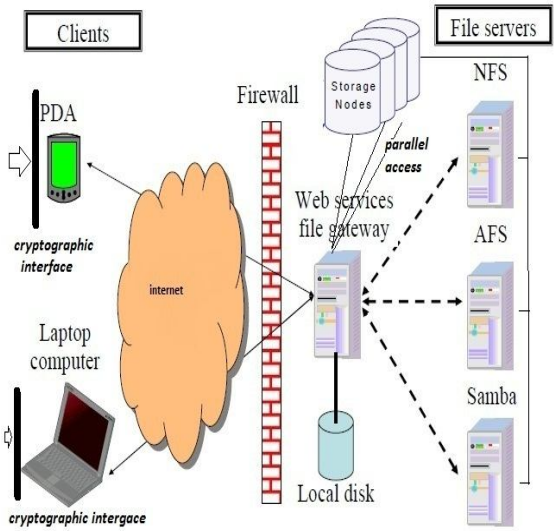

Figure 10: Proposed Architecture

As shown in the figure 9, the architecture diagram for proposed NFS is shown. The Employee/ users are provided with a cryptographic interface, where they can perform different operations. A gateway is installed to support different web services. It will support different DFS. The users can have parallel access to the storage nodes as in pNFS.

6.Conclusion:

We can conclude that NFS protocol provides the efficient way to access the remote data. A user can't understand whether he is accessing remote data or local data. NFS along with different underlying technologies for communication provide most flexible method of remote file access available today. Here we studied different implementations of NFS in different situations

7.References:

[1]Russel Sandberg, David Goldberg "Design and implementation of Sun network file system", Sun Microsystems inc.
[2]Charles Changli Chin, Shang-Rong Tsai "Transparency in a Replicated Network File System" 1996 IEEE
[3]Weikuan Yu, Jeffrey S. Vetter, "Initial Characterization of Parallel NFS implementation" 2010 IEEE
[4]Jun Lu,Bin Du, Yi Zhu,"MADFS: The Mobile Agent-based Distributed Network File system" 2009 IEEE
[5]Gwan-Hwan Hwang, Chih-Hao Yu, "Design and Implementation of a Web-Services-Based Network File System" 2006 IEEE
[6]Thomas E. Andersone, Michael D. Dahlin "Serverless Network File System", ACM Symposium, ACM transactions on Computer Sys
[7]Michael N. Nelson, Brent B. Welch,"Caching in Sprite Network file system" ACM Transaction on Computer Sys.. 1988
[8]Ravi Chandra, Roberto Gamboni "gVault:A Gmail based Cryptographic NFS"
[9]Dean Hildebrand, Lee Ward "Large Files, Small Writes, And pNFS" May 2010. CITI Technical Report
[10]Dean Hildebrand, Peter Honeyman "Direct-pNFS: Scalable, Transparent, and versatile access to parallel file systems" June 2007 ACM
[11] Tatsuya Igarashi, Koichi Hayakawa, Takuya Nishimura "Home Network File System For Home Network Based On IEEE-1394 Technology" IEEE Transactions on Consumer Electronics, Vol. 45, No.3, August 1999
[12] Yi Zhao, Rongfeng Tang, Jin Xiong, Jie Ma "IncFS: An Integrated High-Performance Distributed File System Based on NFS"
[13] Rohit Dube, Cynthia D. Rais, and Satish K. Tripathi "Improving NFS Performance Over Wireless Links" IEEE Transactions On Computers, VOL. 46, NO. 3, March 1997
[14] Song Jiang, Xuechen Zhang, Shuang Liang, and Kei Davis "Improving Networked File System Performance Using a Locality-Aware Cooperative Cache Protocol" IEEE Transactions On Computers, VOL. 59, NO. 11, November 2010
[15] Weikuan Yu, Oleg Drokin, Jeffrey S. Vetter, "Design, Implementation, and Evaluation of Transparent pNFS on Lustre" 2009 IEEE
[16] Alexandros Batsakis and Randal Burns, "NFS-CD: Write-Enabled Cooperative Caching in NFS" IEEE Transactions On Parallel And Distributed Systems, VOL. 19, NO. 3, March 2008
[17] F. Garcia, A. Calderon, J. Carretero, J. M. Perez, and J. Fernandez "A Parallel and Fault Tolerant File System Based on NFS Servers" 2003 IEEE
[18]A. Calderon, F. Garcia, J. Carretero, J. Perez, and J. Fernandez. "An Implementation of MPI-IO on Expand: A Parallel File System Based on NFS Servers" Lecture Notes in Computer Science, 2474:306–313, 2002.
[19] P. Carns, W. L. III, R. Ross, and R. Takhur. "PVFS: A Parallel File System for Linux Clusters" Technical Report ANL/MCS-P804-04
[20]www.google.com