St. Francis Institute of Technology
Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII Subject: Distributed Computing** **Class / Branch / Division: BE/CMPN/A**
Name: Rebecca Dias Roll Number:18

Experiment No: 08

Aim: Case study: Distributed File System

Theory:

CASE STUDY 1:

ANDREW FILE SYSTEM Andrew is a distributed computing environment being developed in ajoint project by Carnegie Mellon University and IBM. One of the major components of Andrew is a distributed file system. The goal of the Andrew File System is to support growth up to at least 7000 workstations (one for each student, faculty member, and staff at Carnegie Mellon) while providing users, application programs, and system administrators with the amenities of a shared file system.

Architecture:

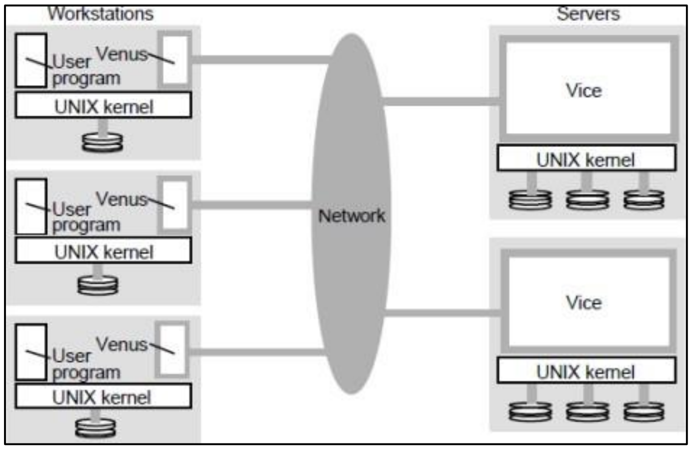


Figure 8.1 Andrew File System Architecture

File System:

The general goal of widespread accessibility of computational and informational facilities, coupled with the choice of UNIX, led to the decision to provide an integrated, campus-wide filesystem with functional characteristics as close to that of UNIX as possible.
The first design choice was to make the file system compatible with UNIX at the system call level. The second design decision was to use whole files as the basic unit of data movement andstorage, rather than some smaller unit such as physical or logical records.
This is undoubtedly the most controversial and interesting aspect of the Andrew File System. It means that before a workstation can use a file, it must copy the entire file to its local disk, and itmust write modified files back to the file system in their entirety. This in turn requires using a local disk to hold recently-used files. On the other hand, it provides significant benefits in performance and to some degree in availability.
Once a workstation has a copy of a file it can use it independently of the central file system. This

dramatically reduces network traffic and file server loads as compared to record-based distributed file systems. Furthermore, it is possible to cache and reuse files on the local disk, resulting in further reductions in server - loads and in additional workstation autonomy. Two functional issues with the whole file strategy are often raised. The first concerns file sizes: only files small enough to fit in the local

CASE STUDY 2: SUN NETWORK FILE SYSTEM NFS views a set of interconnected workstations as a set of independent machines with independent file systems. The goal is to allow some degree of sharing among these file systems in a transparent manner. Sharing is basedon server-client relationship. A machine may be, and often is, both a client and a server. Sharing is allowed between any pair of machines, not only with dedicated server machines. Consistent with the independence of a machine is the critical observation that NFS sharing of a remote file system affects only the client machine and no other machine. Therefore, there is no notion of a globally shared file system as in Locus, Sprite, UNIX United, and Andrew. To make a remote directory accessible in a transparent manner from a client machine, a user of that machine first has to carry out a mount operation. Actually, only a super user can invoke the mount operation. Specifying the remote directory as an argument for the mount operation is done in a nontransparent manner; the location (i.e., hostname) of the remote directory has to be provided. From then on, users on the client machine can access files in the remote directory in a totally transparent manner, as if the directory were local. Since each machine is free to configure its ownname space, it is not guaranteed that all machines have a common view of the shared space. The convention is to configure the system to have a uniform name space. By mounting a shared file system over user home directories on all the machines, a user can log in to any workstation and get his or her home environment. Thus, user mobility can be provided, although again by convention. Subject to access rights accreditation, potentially any file system or a directory within a file system can be remotely mounted on top of any local directory. In the latest NFS version, diskless workstations can even mount their own roots from servers (Version 4.0, May 1988 described in Sun Microsystems Inc. . In previous NFS versions, a diskless workstation depends on the Network Disk (ND) protocol that provides raw block I/O service from remote disks; the server disk was partitioned and no sharing of root file systems was allowed. One of thedesign goals of NFS is to provide file services in a heterogeneous environment of different machines, operating systems, and network architecture. The NFS specification is independent of these media and thus encourages other implementations. This independence is achieved through the use of RPC primitives built on top of an External Date Representation (XDR) protocol-two implementation independent interfaces [Sun Microsystems Inc. 19881. Hence, if the system consists of heterogeneous machines and file systems that are properly interfaced to NFS, file systems of different types can be mounted both locally and remotely.
Architecture In general, Sun’s implementation of NFS is integrated with the SunOS kernel forreasons of efficiency (although such integration is not strictly necessary).

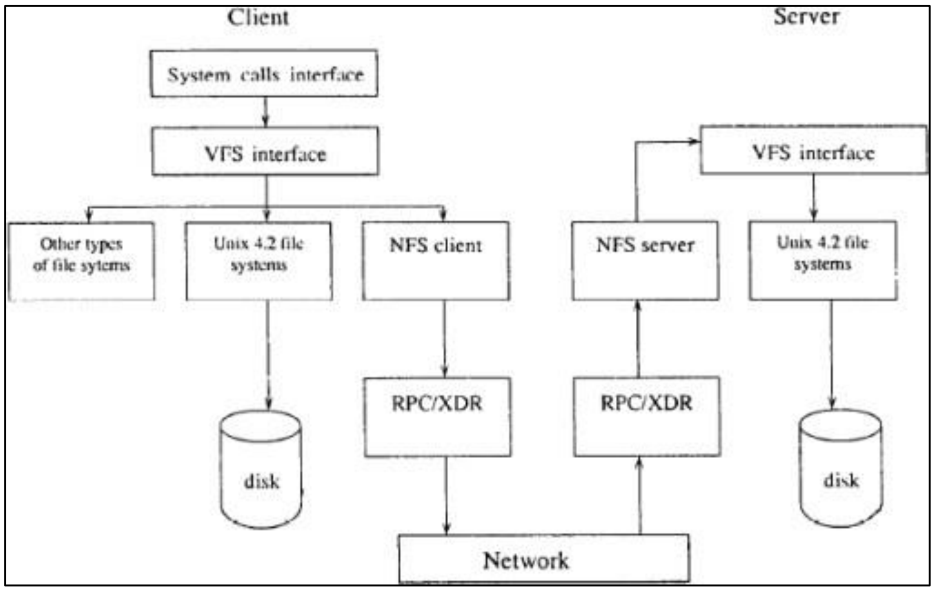


Figure 8.2: Schematic View of NFS architecture

The NFS architecture is schematically depicted in Figure 6. The user interface is the UNIX system calls interface based on the Open, Read, Write, Close calls, and file descriptors. This interface is on top of a middle layer called the Virtual File System (VFS) layer. The bottom layeris the one that implements the NFS protocol and is called the NFS layer. These layers comprise the NFS software architecture. The figure also shows the RPC/XDR software layer, local file systems, and the network and thus can serve to illustrate the integration of a DFS with all these components. The VFS serves two important functions: It separates file system generic operationsfrom their implementation by defining a clean interface. Several implementations for the VFS interface may coexist on the same machine, allowing transparent access to a variety of types of file systems mounted locally (e.g., 4.2 BSD or MSDOS). The VFS is based on a file representation structure called a unode, which contains a numerical designator for a file that is networkwide unique. (Recall that UNIXi- nodes are unique only within a single file system.) Thekernel maintains one vnode structure for each active node (file or directory). Essentially, for every file the vnode structures complemented by the mount table provide a pointer to its parent file system, as well as to the file system over which it is mounted. Thus, the VFS distinguishes local files from remote ones, and local files are further distinguished according to their file system types. The VFS activates file system specific operations to handle local requests according to their file system types and calls the NFS protocol procedures for remote requests.

File handles are constructed from the relevant vnodes and passed as arguments to these procedures. As an illustration of the architecture, let us trace how an operation on an already open remote file is handled (follow the example in Figure 6). The client initiates the operation bya regular system call. The operating system layer maps this call to a VFS operation on the appropriate vnode.

III. CASE STUDY 3: GOOGLE FILE SYSTEM The Google File System (GFS) is a proprietary DFS developed by Google. It is designed to provide efficient, reliable access to data using large clusters of commodity hardware. The files are huge and divided into chunks of 64 megabytes. Most files are mutated by appending new data rather than overwriting existing data: once written, the files are only read and often only sequentially. This DFS is best suited for scenarios in which many large files are created once but read many times. The GFS is optimized to run on computing clusters where the nodes are cheap computers. Hence, there is a need for precautions against the high failure rate of individual nodes and data loss. Motivation for the GFS Design: The GFS was developed based on the following assumptions: a. Systems are prone to failure. Hence there is a need for self monitoring and self recovery from failure. b. The file system storesa modest number of large files, where the file size is greater than 100 MB. c. There are two typesof reads: Large streaming reads of 1MB or more. These types of reads are from a contiguous region of a file by the same client. The other is a set of small random reads of a few KBs. d. Many large sequential writes are performed. The writes are performed by appending data to files and once written the files are seldom modified. e. Need to support multiple clients concurrently appending the same file. Architecture Master – Chunk Servers – Client A GFS cluster consists ofa single master and multiple chunkservers and is accessed by multiple clients. Files are divided into fixed-size chunks. Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation. Chunkservers store chunks onlocal disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers. By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace. The master maintains all file system metadata. It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunkservers. The GFS architecturediagram is shown below:

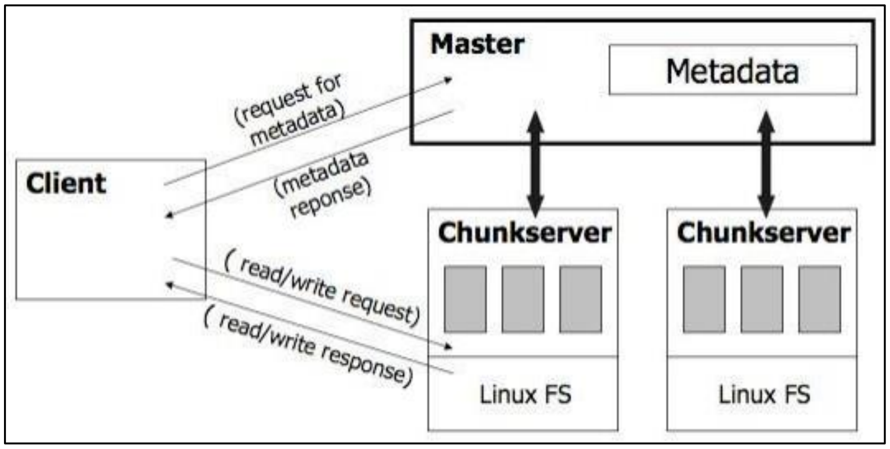


Figure 8.3: Google File System Architecture

Chunk Size The GFS uses a large chunk size of 64MB. This has the following advantages: a. Reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information. b. Reduce networkoverhead by keeping a persistent TCP connection to the chunkserver over an extended period oftime. c. Reduces the size of the metadata stored on the master. This allows keeping the metadatain memory of master. File Access Method File Read A simple file read is performed as follows:
a. Client translates the file name and byte offset specified by the application into a chunk index within the file using the fixed chunk size. b. It sends the master a request containing the file name and chunk index.
c. The master replies with the corresponding chunk handle and locations of the replicas. The client caches this information using the file name and chunk index as the key.
d. The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk. e. Further reads of the same chunkrequire no more client-master interaction until the cached information expires or the file is reopened. This file read sequence is illustrated below:

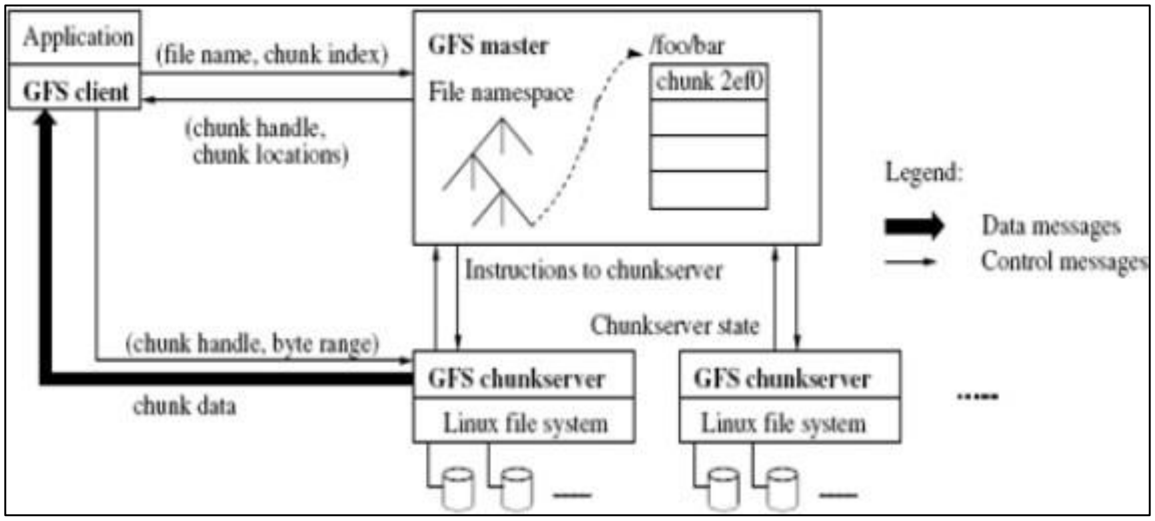


Figure 8.4 File Read System

File Write The control flow of a write is given below as numbered steps: 1. Client translates the file name and byte offset specified by the application into a chunk index within the file using thefixed chunk size. It sends the master a request containing the file name and chunk index. 2. The master replies with the corresponding chunk handle and locations of the replicas 3. The client pushes the data to all the replicas. Data stored in internal buffer of chunkserver. 4. Client sends awrite request to the primary. The primary assigns serial numbers to all the write requests it receives. Perform write on data it stores in the serial number order 5. The primary forwards the write request to all secondary replicas 6. The secondaries all reply to the primary on completion of write 7. The primary replies to the client.

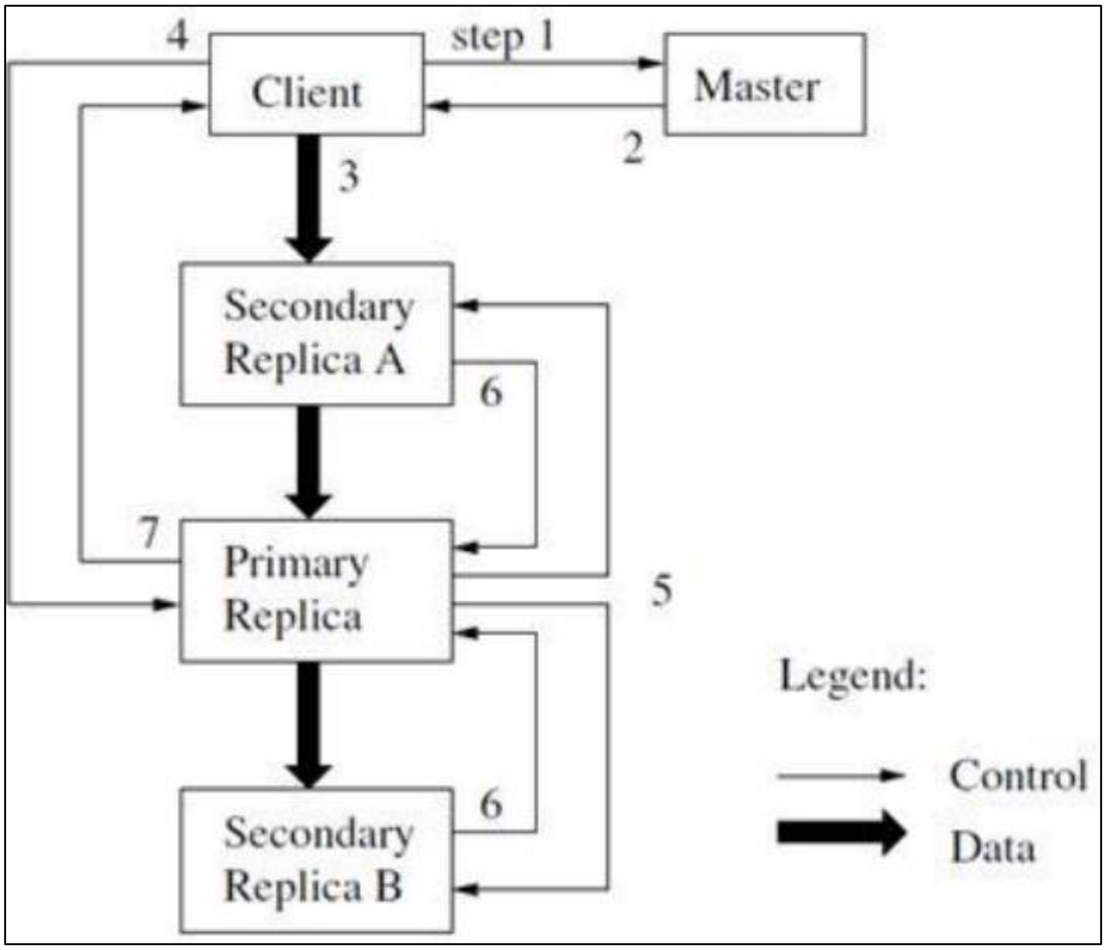


Figure 8.5: A File Write Sequence

Replication and Consistency Consistency The GFS applies mutations to a chunk in the same order on all its replicas. A mutation is an operation that changes the contents or metadata of a chunk such as a write or an append operation. It uses the chunk version numbers to detect any replica that has become stale due to missed mutations while its chunkserver was down. The chance of a client reading from a stale replica stored in its cache is small. This is because the cache entry uses a timeout mechanism. Also, it purges all chunk information for that file on thenext file open. Replication The GFS employs both chunk replication and master replication foradded reliability. System Availability The GFS supports Fast Recovery to ensure availability.

Both the master and the chunkserver are designed to restore their state and start in seconds no matter how they terminated. The normal and abnormal termination are not distinguished. Hence,any fault will result in the same recovery process as a successful termination. Data Integrity TheGFS employs a checksum mechanism to ensure integrity of data being read / written. A 32 bit checksum is included for every 64KB block of chunk. For reads, the chunkserver verifies the checksum of data blocks that overlap the read range before returning any data to the requester.

For writes (append to end of a chunk) incrementally update the checksum for the last partial

checksum block, and compute new checksums for any brand new checksum blocks filled by theappend. Limitations of the GFS 1. No standard API such as POSIX for programming. 2. The Application / client have opportunity to get a stale chunk replica, though this probability is low.
3. Some of the performance issues depend on the client and application implementation. 4. If a write by the application is large or straddles a chunk boundary, it may be added fragments fromother clients.

IV. CASE STUDY 4: HADOOP FILE SYSTEM

The Hadoop is a distributed parallel fault tolerant file system inspired by the Google File System.It was designed to reliably store very large files across machines in a large cluster. Each file is stored as a sequence of blocks; all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. The files are "write once" and have strictly one writer at any time. This DFS has been used by Facebook and Yahoo.

Architecture

The file system metadata and application data stored separately. The Metadata stored on a dedicated server called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-basedprotocols. File content is split into large blocks (typically 128MB) and each block of the file isindependently replicated at multiple DataNodes for reliability.

Name Node

The files and directories represented by inodes, which record attributes like permissions, modification and access times, namespace and disk space quotas. The metadata comprising the inode data and the list of blocks belonging to each file is called the Image. Checkpoints are the persistent record of the image stored in the local host's native files system. The modification logof the image stored in the local host's native file system is referred to as the Journal. During restarts the NameNode restores the namespace by reading the namespace and replaying the journal.

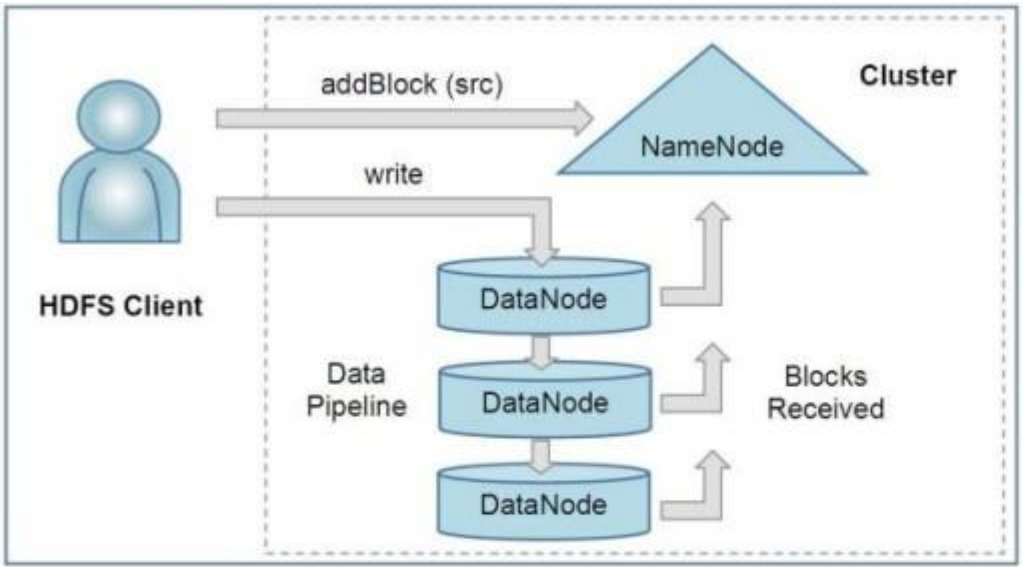
Data Node

Each block replica on a DataNode is represented by two files in the local host's native file system. The first file contains the data itself and the second file is block's metadata including checksums for the block data and the block's generation stamp. The size of the data file equals the actual length of the block and does not require extra space to round it up to the nominal blocksize as in traditional file systems. Thus, if a block is half full it needs only half of the space of thefull block on the local drive. During startup each DataNode connects to the NameNode and performs a handshake. The purpose of the handshake is to verify the namespace ID and the software version of the DataNode. If either does not match that of the NameNode the DataNode automatically shuts down. A DataNode identifies block replicas in its possession to the

NameNode by sending a block report. A block report contains the block id, the generation stampand the length for each block replica the server hosts. The first block report is sent immediately after the DataNode registration. Subsequent block reports are sent every hour and provide the NameNode with an up-to date view of where block replicas are located on the cluster.
File Access Method File Read When an application reads a file, the HDFS client first asks the NameNode for the list of DataNodes that host replicas of the blocks of the file. It then contacts a DataNode directly and requests the transfer of the desired block. File Write When a client writes,it first asks the NameNode to choose DataNodes to host replicas of the first block of the file. Theclient organizes a pipeline from node-to-node and sends the data. When the first block is filled, the client requests new DataNodes to be chosen to host replicas of the next block. A new pipelineis organized, and the client sends the further bytes of the file. Each choice of DataNodes is likelyto be different. Synchronization The Hadoop DFS implements a singlewriter, multiple-reader model. The Hadoop client that opens a file for writing is granted a lease for the file; no other client can write to the file. The writing client periodically renews the lease. When the file is closed, the lease is revoked. The writer's lease does not prevent other clients from reading the file; a file may have many concurrent readers. The interactions involved are shown in the figure below:

Figure 8.6 File Access in Hadoop

Replication Management



The blocks are replicated for reliability. Hadoop has a method to identify and overcome the issues of under-replication and over- replication. The default number of replicas for each block is
3. NameNode detects that a block has become underor over-replicated based on DataNode’s block report. If over replicated, the NameNode chooses a replica to remove. Preference is given to remove from the DataNode with the least amount of available disk space. If under-replicated, it is put in the replication priority queue. A block with only one replica has the highest priority, while a block with a number of replicas that is greater than two thirds of its replication factor has

the lowest priority. It also ensures that not all replicas of a block are located on same physicallocation.

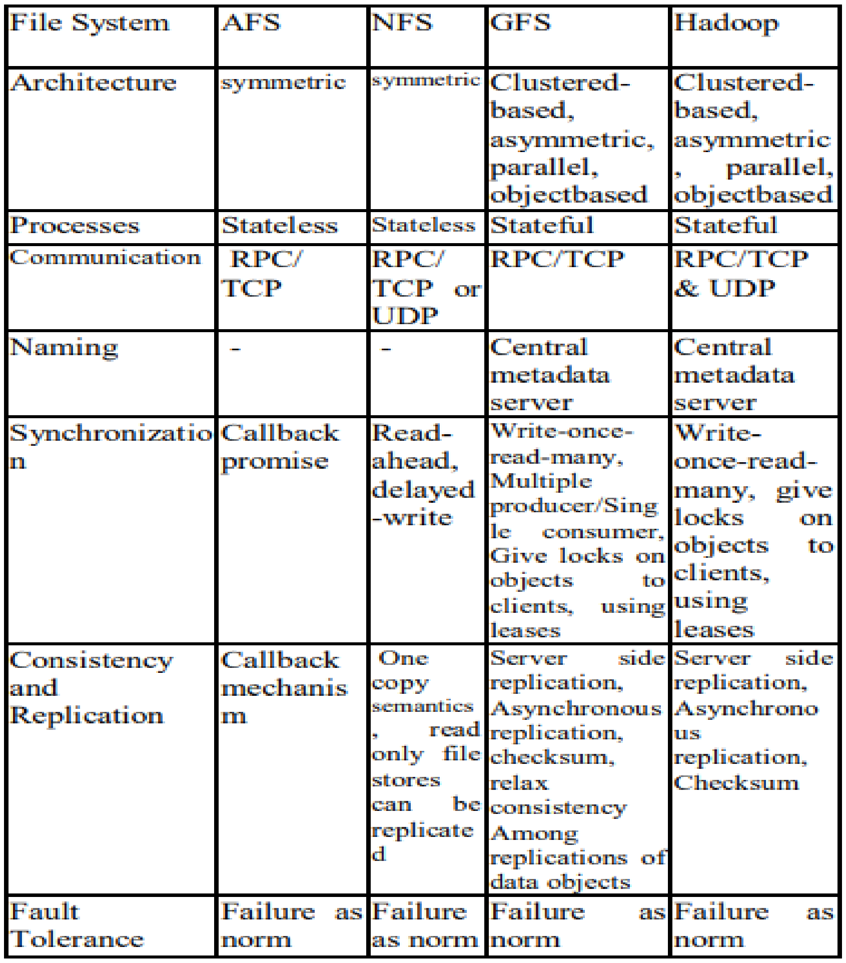
Consistency

The Hadoop DFS use checksums with each data block to maintain data consistency. The checksums are verified by the client while reading to help detect corruption caused either by theclient, the DataNodes, or network. When a client creates an HDFS file, it computes the checksum sequence for each block and sends it to a DataNode along with the data. DataNode stores checksums in a metadata file separate from the block's data file. When HDFS reads a file,each block's data and checksums are returned to the client.

Limitations of Hadoop DFS

1. Centralization: The Hadoop system uses a centralized master server. So, the Hadoop cluster is effectively unavailable when its NameNode is down. Restarting the NameNode has been a satisfactory recovery method so far and steps being taken towards automated recovery.

2. Scalability: Since the NameNode keeps all the namespace and block locations in memory, thesize of the NameNode heap limits number of files and blocks addressable. One solution is to allow multiple namespaces (and NameNodes) to share the physical storage within a cluster.



File System	AFS	NFS	GFS	Hadoop
Architecture	symmetric	symmetric	Clustered-based, asymmetric, parallel, objectbased	Clustered-based, asymmetric, parallel, objectbased
Processes	Stateless	Stateless	Stateful	Stateful
Communication	RPC/ TCP	RPC/ TCP or UDP	RPC/TCP	RPC/TCP & UDP
Naming	-	-	Central metadata server	Central metadata server
Synchronization	Callback promise	Read-ahead, delayed -write	Write-once-read-many, Multiple producer/Single consumer, Give locks on objects to clients, using leases	Write-once-read-many, give locks on objects to clients, using leases
Consistency and Replication	Callback mechanism	One copy semantics, read only file stores can be replicated	Server side replication, Asynchronous replication, checksum, relax consistency Among replications of data objects	Server side replication, Asynchronous replication, Checksum
Fault Tolerance	Failure as norm	Failure as norm	Failure as norm	Failure as norm

1. What is DFS (Distributed File System)?

A **Distributed File System (DFS)** as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access orstore isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

The main purpose of the Distributed File System (DFS) is to allows users of physically distributed systems to share their data and resources by using a Common File System. A collection of workstations and mainframes connected by a Local Area Network (LAN) isa configuration on Distributed File System. A DFS is executed as a part of the operating system. In DFS, a namespace is created and this process is transparent for the clients.

DFS has two components:

- **Location Transparency –**
  Location Transparency achieves through the namespace component.
- **Redundancy –**
  Redundancy is done through a file replication component.

In the case of failure and heavy load, these components together improve data availabilityby allowing the sharing of data in different locations to be logically grouped under one folder, which is known as the "DFS root".

It is not necessary to use both the two components of DFS together, it is possible to use the namespace component without using the file replication component and it is perfectly possible to use the file replication component without using the namespace component between servers.

2. What are the Features of DFS ?

- **Transparency :**
  - **Structure transparency –**
    There is no need for the client to know about the number or locations offile servers and the storage devices. Multiple file servers should be provided for performance, adaptability, and dependability.
  - **Access transparency –**
    Both local and remote files should be accessible in the same manner. The file system should be automatically located on the accessed file and send itto the client's side.
  - **Naming transparency –**
    There should not be any hint in the name of the file to the location of thefile. Once a name is given to the file, it should not be changed during transferring from one node to another.
  - **Replication transparency –**
    If a file is copied on multiple nodes, both the copies of the file and their locations should be hidden from one node to another.

- **User mobility :**
  It will automatically bring the user's home directory to the node where the userlogs in.
- **Performance :**
  Performance is based on the average amount of time needed to convince the client requests. This time covers the CPU time + time taken to access secondary storage + network access time. It is advisable that the performance of the Distributed FileSystem be similar to that of a centralized file system.
- **Simplicity and ease of use :**
  The user interface of a file system should be simple and the number of commandsin the file should be small.
- **High availability :**
  A Distributed File System should be able to continue in case of any partialfailures like a link failure, a node failure, or a storage drive crash.
  A high authentic and adaptable distributed file system should have different and independent file servers for controlling different and independent storage devices.
- **Scalability :**
  Since growing the network by adding new machines or joining two networks together is routine, the distributed system will inevitably grow over time. As aresult, a good distributed file system should be built to scale quickly as the number of nodes and users in the system grows. Service should not be substantially disrupted as the number of nodes and users grows.
- **High reliability :**
  The likelihood of data loss should be minimized as much as feasible in a suitable distributed file system. That is, because of the system's unreliability, users shouldnot feel forced to make backup copies of their files. Rather, a file system should create backup copies of key files that can be used if the originals are lost. Many file systems employ stable storage as a high-reliability strategy.

3. What is the history of distributed file system?

The server component of the Distributed File System was initially introduced as an add-on feature. It was added to Windows NT 4.0 Server and was known as "DFS 4.1".
Then later on it was included as a standard component for all editions of Windows 2000Server. Client-side support has been included in Windows NT 4.0 and also in later on version of Windows.

Linux kernels 2.6.14 and versions after it come with an SMB client VFS known as "cifs"which supports DFS. Mac OS X 10.7 (lion) and onwards supports Mac OS X DFS.

4. What are the applications of the Distributed file system?

- **NFS –**
  NFS stands for Network File System. It is a client-server architecture that allows a computer user to view, store, and update files remotely. The protocol of NFS is one of the several distributed file system standards for Network-Attached Storage (NAS).
- **CIFS –**
  CIFS stands for Common Internet File System. CIFS is an accent of SMB. Thatis, CIFS is an application of SIMB protocol, designed by Microsoft.
- **SMB –**
  SMB stands for Server Message Block. It is a protocol for sharing a file and was

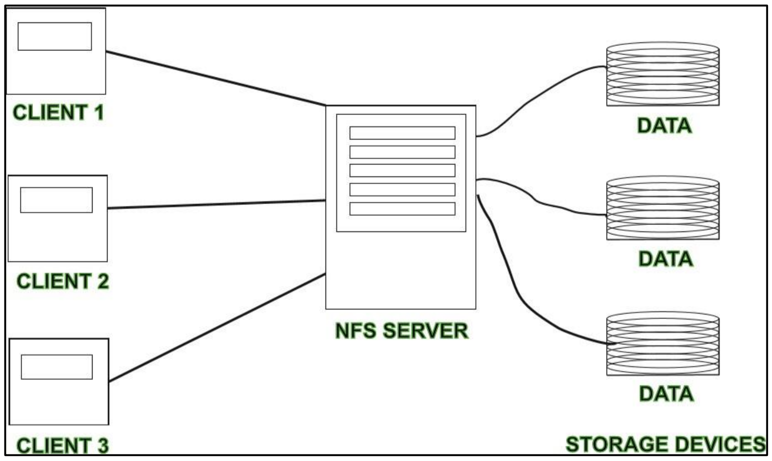
invented by IMB. The SMB protocol was created to allow computers to performread and write operations on files to a remote host over a Local Area Network (LAN). The directories present in the remote host can be accessed via SMB and are called as "shares".

- **Hadoop –**
  Hadoop is a group of open-source software services. It gives a software framework for distributed storage and operating of big data using the MapReduceprogramming model. The core of Hadoop contains a storage part, known as Hadoop Distributed File System (HDFS), and an operating part which is a MapReduce programming model.
- **NetWare –**
  NetWare is an abandon computer network operating system developed by Novell,Inc. It primarily used combined multitasking to run different services on a personal computer, using the IPX network protocol.

5. Explain the working of DFS.

There are two ways in which DFS can be implemented:

- **Standalone DFS namespace –**
  It allows only for those DFS roots that exist on the local computer and are not using Active Directory. A Standalone DFS can only be acquired on those computers on which it is created. It does not provide any fault liberation and cannot be linked to any other DFS. Standalone DFS roots are rarely come acrossbecause of their limited advantage.
- **Domain-based DFS namespace –**
  It stores the configuration of DFS in Active Directory, creating the DFS namespace root accessible at \\**<domainname>****<dfsroot>** or **\\<FQDN>\<dfsroot**>



6. What are the Advantages and disadvantages of DFS?

Advantages :

- DFS allows multiple user to access or store the data.
- It allows the data to be share remotely.
- It improved the availability of file, access time, and network efficiency.
- Improved the capacity to change the size of the data and also improves the abilityto exchange the data.
- Distributed File System provides transparency of data even if server or disk fails.

Disadvantages :

- In Distributed File System nodes and connections needs to be secured thereforewe can say that security is at stake.
- There is a possibility of lose of messages and data in the network while movementfrom one node to another.
- Database connection in case of Distributed File System is complicated.
- Also handling of the database is not easy in Distributed File System as comparedto a single user system.
- There are chances that overloading will take place if all nodes tries to send data atonce.

Conclusion:

In this Case study we learned about the Distributed File System (DFS) in detail, about its history, architecture , uses , advantages and disadvantages.

Reference:

https://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/wdm/wdm_ch14b.pdf
https://www.ijera.com/papers/Vol3_issue1/GT3112931298.pdf
https://www.techopedia.com/definition/1825/distributed-file-system-dfs#:~:text=A%20distributed%20file%20system%20(DFS,a%20controlled%20and%20authorized%20way.
https://www.geeksforgeeks.org/what-is-dfsdistributed-file-system/