AIM : Case Study On Distributed File System

THEORY :

1. What is a Distributed File system?

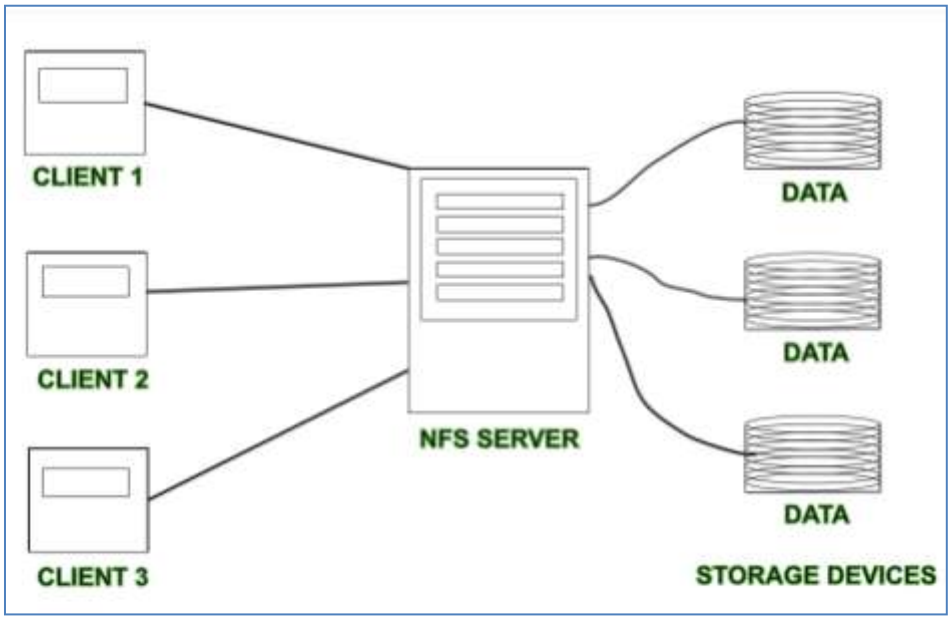
A Distributed File System (DFS) as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. It allows programs to access or store isolated files as they do with the local ones, allowing programmers to access files from any network or computer.

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) is a 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.

2. Architecture of DFS.



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). 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 across because 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>

3. Design Issues in Distributed File system.

1. Naming and Name Resolution: It conflicts with the goal of network tranparency. Moving a file from one host to another requires changes in filename and the application accessing that file that is naming scheme is not location independent.
2. Caches on Disk or Main Memory: Large files cannot be cached completely so caching done block oriented which is more complex. It competes with virtual memory system for physical memory space, so a scheme to deal with memory contention cache and virtual memory system is necessary. Thus, more complex cache manager and memory management is required.
3. Writing Policy:
   This policy decides when a modified cache block at client should be transferred to the server. Following policies are used:
   a) Write Through: All writes required by clients applications are also carried out at servers immediately. The main advantage is reliability that is when client crash, little information is lost. This scheme cannot take advantage of caching.
   b) Delayed Writing Policy: It delays the writing at the server that is modifications due to write are reflected at server after some delay. This scheme can take advantage of caching. The main limitation is less reliability that is when client crash, large amount of information can be lost.
   c) This scheme delays the updating of files at the server until the file is closed at the client. When average period for which files are open is short then this policy is equivalent to write through and when period is large then this policy is equivalent to delayed writing policy.
4. Cache Consistency: When multiple clients want to modify or access the same data, then cache consistency problem arises.
5. Availability: Extra storage space is required to store replicas. Extra overhead is required in maintained all replicas up to date.

6. Scalability: The design of DFS should be such that new systems can be easily introduced without affecting it Structure of server process play an important role. If server is designed with single process, then many clients have to wait for a long time whenever a disk input/ output is initiated. This can be avoided if separate process is assigned to each client.
7. Semantics: The semantic of a file system represent the affects of access on file. The basic semantic is that a read operation will return the data (stored) due to latest write operation. The semantic can be guaranteed in two ways: All read and writes from various clients will have to go through the server. Sharing will have to be disallowed either by server or by the use of locks by application. In first way, the server become bottleneck and in second way, the file is not available for certain clients.

4. Services provided by distributed file system.

1. Transparency:
    a. Structure transparency: There is no need for the client to know about the number or locations of file servers and the storage devices. Multiple file servers should be provided for performance, adaptability, and dependability.
    b. 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 it to the client's side.
    c. Naming transparency: There should not be any hint in the name of the file to the location of the file. Once a name is given to the file, it should not be changed during transferring from one node to another.
    d. 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.
2. User mobility: It will automatically bring the user's home directory to the node where the user logs in.
3. 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 File System be similar to that of a centralized file system.
4. Simplicity and ease of use: The user interface of a file system should be simple and the number of commands in the file should be small.
5. High availability: A Distributed File System should be able to continue in case of any partial failures 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.

5. Mechanisms for building the DFS.

1. Mounting

This mechanism provides the binding together of different filename spaces to form a single hierarchically structured name space. It is UNIX specific and most of existing DFS (Distributed File System) are based on UNIX. A filename space can be bounded to or mounted at an internal node or a leaf node of a namespace tree. A node onto which a name space is mounted is called mount point. The kernel maintains a mount table, which maps mount points to appropriate storage devices.

Uses of Mounting in DFS

File systems maintained by remote servers are mounted at clients so that each client have information regarding file servers. Two approaches are used to maintain mount information.

- Approach 1: Mount information is maintained at clients that is each client has to individually mount every required file system. When files are moved to a different server then mount information must be updated in mount table of every client
- Approach 2: Mount information is maintained at servers. If files are moved to a different servers, then mount information need only be updated at servers.

2. Caching

This mechanism is used in DFS to reduce delays in accessing of data. In file caching, a copy of data stored at remote file server is brought to client when referenced by client so subsequent access of data is performed locally at client, thus reducing access delays due to netwrok latency. Data can be cached in main memory or on the local disk of the clients. Data is cached in main memory at servers to reduce disk access latency.

3. Hints

Caching results in the cache consistency problem when multiple clients cache and modify shared data. This problem can be avoided by great level of co-operation between file servers and clients which is very expansive. Alternative method is to treat cached data as hints that is cached data are not expected to be completely accurate. Only those class of applications which can recover after discovering that cached data are invalid can use this approach. Example: After the name of file or directory is mapped to physical object, the address of object can be stored as hint in the cache. If the address is incorrect that is fails to map the object, the cached address is deleted form the cache and file server consult the same server to obtain the actual location of file or directly and updated the cache.

4. Bulk Data Transfer

In this mechanism, multiple consecutive data blocks are trasferred from server to client. This reduces file access overhead by obtaining multiple number of blocks with a single seek, by formatting and transmitting multiple number of large packets in single context switch and by reducing the number of acknowledgement that need to be sent. This mechanism is used as many files are accessed in their entirety.

5. Encryption

This mechanism is used for security in Distributed systems. The method was developed by Needham Schrodkar is used in DFS security. In this scheme, two entities which want to communicate establish a key for conversation with help of authentication server.

6. Examples of DFS.

Some distributed File systems are

1. HDFS : Distributed file system providing high-throughput access.
2. Lustre : File system for computer clusters.
3. CephFS : Unified, distributed storage system.
4. Alluxio : Virtual distributed file system.
5. GlusterFS : Scale-out NAS file system.
6. MooseFS : POSIX-compliant distributed file system.
7. XtreemFS : Object-based, distributed file system for wide area networks.
8. Quantcast : File System High-performance, fault-tolerant, distributed file system.
9. OrangeFS : Multi-server scalable parallel file system.

7. Explain any one DFS.

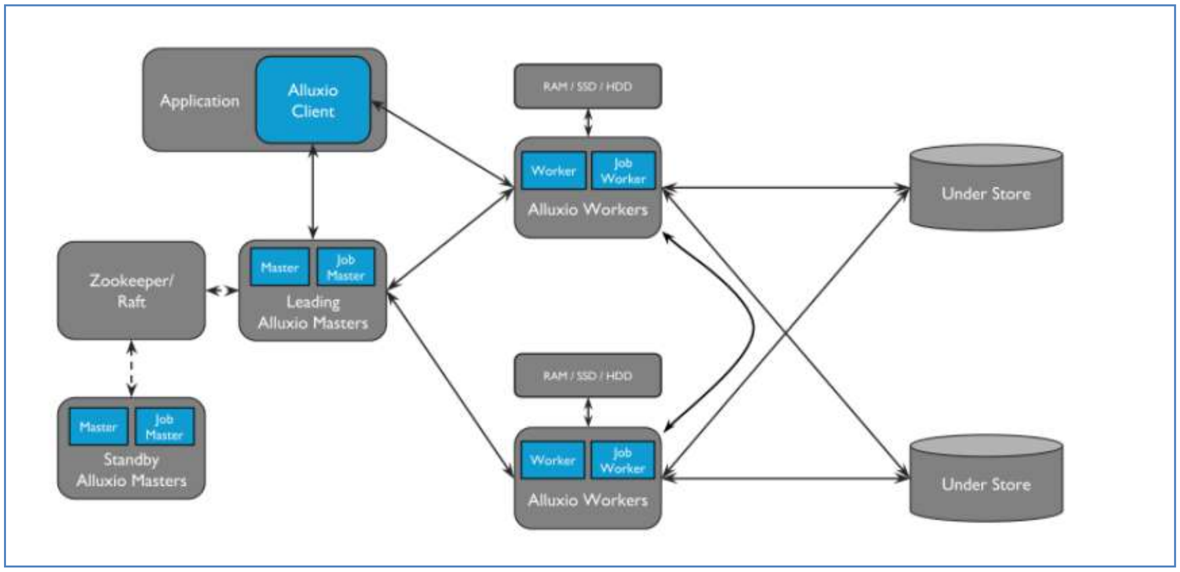
<u>**Alluxio**</u>

Alluxio is an open-source virtual distributed file system (VDFS). Initially as research project "Tachyon", Alluxio was created at the University of California, Berkeley's AMPLab as Haoyuan Li's Ph.D. Thesis, advised by Professor Scott Shenker & Professor Ion Stoica. Alluxio sits between computation and storage in the big data analytics stack. It provides a data abstraction layer for computation frameworks, enabling applications to connect to numerous storage systems through a common interface. The software is published under the Apache License.

Some Features :

- OS Buffer / Page Cache
  The operating system will automatically try to utilize a machine's memory accelerate disk I/O. When repeatedly running jobs on the same dataset which fits in memory, this approach is effective and can provide similar performance benefits as Alluxio. However, with larger datasets or more varied workloads, the performance is highly variable and on average much less efficient than a data-aware system like Alluxio.
- Spark Persist
  Spark provides options to temporarily persist data for subsequent use without using any other systems. However, these mechanisms are limited to a single Spark Context, which prevents multiple users from gaining the benefits of one user's persisted data. As a result, each Spark Context consumes resources for its own in-memory or on-disk storage, which is inefficient in a shared environment, especially when large amounts of memory are consumed needlessly.

Architecture :



Alluxio serves as a new data access layer in the big data and machine learning ecosystem. It resides between any persistent storage systems such as Amazon S3, Microsoft Azure Object Store, Apache HDFS, or OpenStack Swift, and computation frameworks such as Apache Spark, Presto, or Hadoop MapReduce. Note that Alluxio itself is not a persistent storage system.

Alluxio can be divided into three components: masters, workers, and clients. A typical cluster consists of a single leading master, standby masters, a job master, standby job masters, workers, and job workers. The master and worker processes constitute the Alluxio servers, which are the components a system administrator would maintain. Clients are used to communicate with the

Alluxio servers by applications such as Spark or MapReduce jobs, the Alluxio CLI, or the Alluxio FUSE layer.

Alluxio Job Masters and Job Workers can be separated into a separate function which is termed the Job Service. The Alluxio Job Service is a lightweight task scheduling framework responsible for assigning a number of different types of operations to Job Workers. Those tasks include

- Loading data into Alluxio from a UFS
- Persisting data to a UFS
- Replicating files within Alluxio
- Moving or copying data between UFS or Alluxio locations

The job service is designed so that all job related processes don't necessarily need to be located with the rest of the Alluxio cluster. However, we do recommend that job workers are co-located with a corresponding Alluxio worker as it provides lower latency for RPCs and data transfer.

Advantages:

- Remedies the performance drawback
- Acceleration due to memory-speed I/O
- Designed to improve the affinity of compute and storage
- For user applications and computation frameworks, Alluxio provides fast storage, and facilitates data sharing and locality between applications, regardless of the computation engine used.
- As a result, Alluxio can serve data at memory speed when it is local or at the speed of the computation cluster network when data is in Alluxio.
- For under storage systems, Alluxio bridges the gap between big data applications and traditional storage systems, expanding the set of workloads available to utilize the data.
- Since Alluxio hides the integration of under storage systems from applications, any under storage can support any of the applications and frameworks running on top of Alluxio.

Disadvantages:

- Alluxio persist data first in memory to achieve speed, so Spark jobs could have less memory to run jobs.
- There are overhead time to write dataframe to Allluxio first.

CONCLUSION :

From this experiment we learnt about distributed file system(DFS). A Distributed File System (DFS) as the name suggests, is a file system that is distributed on multiple file servers or multiple locations. We understood the architecture of DFS with the design issues and the mechanisms to create one. Then we saw what services are provided by these DFS with some examples of DFS. Finally we prepared a Case study on Alluxio which is a virtual distributed file system.