**KARNATAK LAW SOCIETY’S**

**GOGTE INSTITUTE OF TECHNOLOGY**

**UDYAMBAG, BELAGAVI – 590008**

**(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)**

**(Approved By AICTE, New Delhi)**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**COURSE PROJECT: DISTRIBUTED COMPUTING SYSTEM**

**REPORT ON ANDREW FILE SYSTEM**

Guided by: Prof. Rakesh Kadkol

Hemanth Tubachi 2GI18IS015

Laxmi Nyamagoud 2GI18IS020

Rachana Kampli 2GI18IS032

Rohan Kokatanur 2GI18IS066

**2020-2021**

**CERTIFICATE**



This is to certify that **Mr. Hemanth I T, Ms. Laxmi Nyamagoud, Ms. Rachana Kampli, Mr. Rohan Kokatanur** of **Sixth Semester** bearing **USN: 2GI18IS015, 2GI18IS020, 2GI18IS032, 2GI18IS066** has satisfactorily completed the course in Course activity of Distributed Computing System. It can be considered as a bonafide work carried out for partial fulfillment of the academic requirement of 6th Semester B.E. (Information Science & Engineering) prescribed by KLS Gogte Institute of Technology, Belagavi during the academic year 2020-21.

The report has been approved as it satisfies the academic requirements prescribed for the said degree.

**Signature of The Faculty Member Signature of The HOD.**

Date: 21/05/2021

**TITLE:**

ANDREW FILE SYSTEM

**File System:**

In [computing](https://en.wikipedia.org/wiki/Computing), a file system or filesystem (often abbreviated to fs) controls how data is [stored](https://en.wikipedia.org/wiki/Computer_data_storage) and retrieved. Without a file system, data placed in a storage medium would be one large body of data with no way to tell where one piece of data stops and the next begins. By separating the data into pieces and giving each piece a name, the data is easily isolated and identified. Taking its name from the way paper-based data management systems are named, each group of data is called a "[file](https://en.wikipedia.org/wiki/Computer_file)." The structure and logic rules used to manage the groups of data and their names is called a "file system".

**Distributed File System:**

*Distributed file systems* do not share [block level access](https://en.wikipedia.org/wiki/Block-level_storage) to the same storage but use a network [protocol](https://en.wikipedia.org/wiki/Protocol_(computing)).[[3]](https://en.wikipedia.org/wiki/Clustered_file_system#cite_note-SilberschatzOperatingSystemConcepts-3)[[4]](https://en.wikipedia.org/wiki/Clustered_file_system#cite_note-ostep-1-4) These are commonly known as [network file systems](https://en.wikipedia.org/wiki/Distributed_file_system), even though they are not the only file systems that use the network to send data.[[5]](https://en.wikipedia.org/wiki/Clustered_file_system#cite_note-SunNetworkFileSystem-5) Distributed file systems can restrict access to the file system depending on [access lists](https://en.wikipedia.org/wiki/Access_list) or [capabilities](https://en.wikipedia.org/wiki/Capability-based_security) on both the servers and the clients, depending on how the protocol is designed.

The difference between a distributed file system and a [distributed data store](https://en.wikipedia.org/wiki/Distributed_data_store) is that a distributed file system allows files to be accessed using the same interfaces and semantics as local files – for example, mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model. Distributed data stores, by contrast, require using a different API or library and have different semantics (most often those of a database).

**Design Goals**

Distributed file systems may aim for "transparency" in a number of aspects. That is, they aim to be "invisible" to client programs, which "see" a system which is similar to a local file system. Behind the scenes, the distributed file system handles locating files, transporting data, and potentially providing other features listed below.

* ***Access transparency*:** clients are unaware that files are distributed and can access them in the same way as local files are accessed.
* ***Location transparency***: a consistent namespace exists encompassing local as well as remote files. The name of a file does not give its location.
* ***Concurrency transparency*:** all clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.
* ***Failure transparency*:** the client and client programs should operate correctly after a server failure.
* ***Heterogeneity*:** file service should be provided across different hardware and operating system platforms.
* ***Scalability*:** the file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to bigger ones (hundreds through tens of thousands of systems).
* ***Replication transparency*:** Clients should be unaware of the file replication performed across multiple servers to support scalability.

*Migration transparency*: files should be able to move between different servers without the client's knowledge.

**Andrew File System - Introduction:**

* Andrew File System (AFS) is one of the distributed file systems that been developed at Carnegie Mellon University (CMU) for use as a campus computing the information system.
* The design of AFS reflects an intention to support information sharing on a large scale by minimizing client-server communication.
* It is designed to be heterogeneous, scalable, it runs efficiently on variations on UNIX.
* AFS provide scalability to thousands of workstation at one site while offering users, applications administrations the conveniences of a shared file system.
* The design of the Andrew File System (henceforth abbreviated AFS) reflects an intention to support information sharing on a large scale by minimizing client-server communication.
* This is achieved by transferring whole files between server and client computers and caching them at clients until the server receives a more up-to-date version.
* AFS provides transparent access to remote shared files for UNIX programs running on workstations.
* AFS is designed to perform well with larger numbers of active users than other distributed file systems.

**Characteristics:**

AFS has two unusual design characteristics:

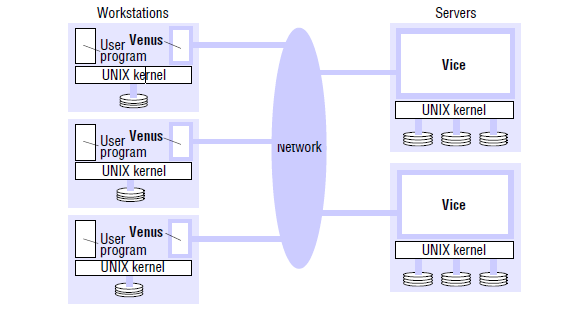
* **Whole-file serving**: The entire contents of directories and files are transmitted to client computers by AFS servers (in AFS-3, files larger than 64 Kbytes are transferred in 64-kbyte chunks).
* **Whole-file caching:** Once a copy of a file or a chunk has been transferred to a client computer it is stored in a cache on the local disk. The cache contains several hundred of the files most recently used on that computer. The cache is permanent, surviving reboots of the client computer. Local copies of files are used to satisfy clients’ open requests in preference to remote copies whenever possible.

**Scenario:**

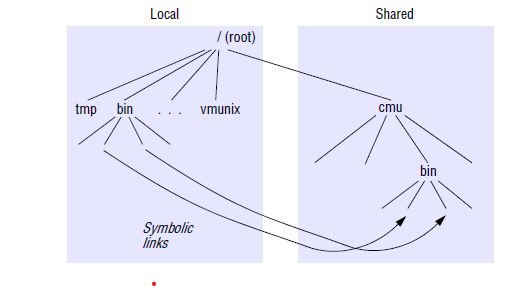
Here is a simple scenario illustrating the operation of AFS:

* When a user process in a client computer issues an open system call for a file in the shared file space and there is not a current copy of the file in the local cache, the server holding the file is located and is sent a request for a copy of the file.
* The copy is stored in the local UNIX file system in the client computer. The copy is then opened and the resulting UNIX file descriptor is returned to the client.
* Subsequent read, write and other operations on the file by processes in the client computer are applied to the local copy.
* When the process in the client issues a close system call, if the local copy has been updated its contents are sent back to the server. The server updates the file contents and the timestamps on the file. The copy on the client’s local disk is retained in case it is needed again by a user-level process on the same workstation.

**Implementation:**

****

* AFS is implemented as two software components that exist as UNIX processes called Vice and Venus.
* Vice is the name given to the server software that runs as a user-level UNIX process in each server computer,
* Venus is a user-level process that runs in each client computer and corresponds to the client module in our abstract model.
* The files available to user processes running on workstations are either local or shared.
* Local files are handled as normal UNIX files. They are stored on a workstation’s disk and are available only to local user processes. Shared files are stored on servers, and copies of them are cached on the local disks of workstations.



* It is a conventional UNIX directory hierarchy, with a specific subtree (called cmu) containing all of the shared files.
* This splitting of the file name space into local and shared files leads to some loss of location transparency, but this is hardly noticeable to users other than system administrators.
* Local files are used only for temporary files (/tmp) and processes that are essential for workstation startup.
* Other standard UNIX files (such as those normally found in /bin, /lib and so on) are implemented as symbolic links from local directories to files held in the shared space.

**Conclusion:**

* AFS demonstrated the feasibility of relatively simple architecture using serve state to reduce the cost of maintaining coherent client caches.
* AFS outperforms in several situations compare to other distributed file systems.

**References:**

* Kai Hwang, Geofrey C, Fox, Jack J, Dongarra: Distributed and Cloud Computing From Parallel processing to the Internet of Things.
* Sunita Mahajan, Seema Shah: Distributing Computing, Published by Oxford Universitypress 2010.
* https://www.powershow.com/viewfl/6ab13-ZDc1Z/Andrew\_File\_System\_powerpoint\_ppt\_presentation