Comparative Analysis of Andrew File System and Hadoop Distributed File System

Monali Mavani, *Member, IACSIT*

Abstract—Sharing of resources is the main goal of distributed system. The sharing of stored information is the most important aspect of distributed resource sharing. A file system was originally developed to provide convenient programming interface to disk storage for the centralized system. With the advent of distributed systems distributed storage has become very prominent. A distributed file system enables users to store and access remote files exactly as they do local ones, allowing users to access files from any computer on a network. The objective of this paper is to compare very first open source wide distribution of distributed file system called Andrew file system and the latest widely used distributed file system -Hadoop distributed file system. Parameters which are taken for comparison are Design Goals, Processes, File management, Scalability, Protection, Security, cache management replication etc.

Index Terms—Andrew file system, Google file system, Hadoop distributed file system.

I. INTRODUCTION

In the distributed system components located at networked computers communicate and coordinate their actions by message passing. The main aspects that can be considered by designing distributed systems are heterogeneity, openness, security, scalability, failure handling, concurrency; transparency etc. The sharing of the resources is a main motivation for constructing distributed systems. Distributed file system allows users to share files and access files from anywhere within the distributed system. A distributed file system typically provides three types of the service:

- 1) Storage service
- 2) True file service
- 3) Name service [1]

In this paper comparison is made in terms of the features of two distributed file system: Andrew file system (first wide distribution of distributed file system) and the latest distributed file system Hadoop distributed file system (HDFS) [2] which is an open source implementation of Google file system (GFS) [3].

II. ANDREW FILE SYSTEM (AFS)

AFS was conceived in 1983 at Carnegie Mellon University with the goal of serving the campus community and spanning at least 5000 workstations. The design and implementation went through three major revisions: AFS-1

in 1984, AFS-2 in 1986, and AFS-3 in 1989. After 1989, responsibility for further development of AFS was transferred to Transarc Corporation. As of early 1989 there were four cells on the CMU campus with a total of 30 file servers and 1000 clients [4]. The general goal of AFS as cited in [5] was widespread accessibility of computational and informational facilities, coupled with the choice of UNIX, an integrated, campus-wide file system with functional Characteristics as close to that of UNIX as possible. They wanted a student to be able to sit down at any workstation and start using his or her files with as little bother as possible. Furthermore they did not want to modify existing application programs, which assume a UNIX file system, in any way. Thus, our first design choice was to make the file system compatible with UNIX at the system call level.

A. AFS Architecture

Fig. 1 shows the distribution of processes in the Andrew file system. Vice, is the information sharing backbone of the system, it consists of a collection of dedicated file servers and a complex local area network. A process called Venus, running on each workstation, mediates shared file access. Venus finds files in Vice, caches them locally, and performs emulation of UNIX file system semantics. Both Vice and Venus are invisible to workstation processes, which only see a UNIX file system. A workstation with a small disk can potentially access any file in Andrew by name. The three variant of AFS are AFS-1, AFS-2 and AFS-3. AFS-1 was in use for about a year from late 1984 to late 1985. At its peak usage there were about 100 workstations and six servers. Performance was usually acceptable to about 20 active users per servers. The system turned out to be difficult to operate and maintained because it provided few tools for system administrators. In AFS-2 improvement was made especially in terms of cache coherence. Venus now assumed that cache entries were valid unless otherwise notified using callback tokens. AFS-3 introduces concept of cells to have decentralized autonomous controls. AFS-3 supports multiple administrative cells, each with its own servers, workstations, system administrators, and users [6].

Fig. 1. Distribution of processes in the andrew file system.

Manuscript received October 8, 2012; revised January 17, 2013.

Monali Mavani is with the SIES College of Management Studies, Mumbai University (e-mail: monamavani@gmail.com).

III. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

HDFS is the file system which is used in Hadoop based distributed system. Hadoop is open source implementation of framework Map Reduce. Map Reduce is a well-known framework for programming commodity computer clusters to perform large-scale data processing in a single pass. Google File System (GFS) and Map Reduce [8] are the representative distributed file system and programming models for cloud computing. Based on these technologies, Apache is managing an open-source based Hadoop project, derived from the Nutch project [9]. The Hadoop project releases the Hadoop Distributed File System (HDFS) and Map Reduce framework. Hadoop, an open-source Map Reduce implementation, has been adopted by Yahoo!, Facebook, and other companies for large-scale data analysis. It is used for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. It is written in Java for cross-platform portability. Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage [2]. Hadoop makes it significantly easier to access and analyze large volumes of data, due to its simple programming model. It hides the 'messy' details of parallelization, allowing even inexperienced programmers to easily utilize the resources of a large distributed system. Although it is written in Java, Hadoop streaming allows its implementation using any programming language [10]. HDFS is highly fault-tolerant distributed file system designed to run on low-cost commodity hardware. It is suitable for applications that have large data sets. HDFS implements a permissions model for files and directories that shares much of the POSIX model. Each file and directory is associated with an owner and a group. The file or directory has separate permissions for the user that is the owner, for other users that are members of the group, and for all other users.

A. Server Architecture and Optimization

HDFS stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. A HDFS installation consists of a single name node as the master node and a number of data nodes as the slave nodes. The name node manages the file system namespace and regulates access to files by clients. The data nodes are distributed, one data node per machine in the cluster, which manage data blocks attached to the machines where they run. The name node executes the operations on file system namespace and maps data blocks to data nodes. The data nodes are responsible for serving read and write requests from clients and perform block operations upon instructions

from name node [11].

HDFS distributes data chunks and replicas across the servers for higher performance, load-balancing and resiliency. With data distributed across all servers, any server may be participating in the reading, writing, or computation of a data-block at any time. Such a data placement complicates power-management and makes it hard to generate significant periods of idleness in the Hadoop clusters and renders usage of inactive power modes infeasible [13]. Kaushik and Bhandarkar proposed in their research GreenHDFS, an energy-conserving, self-adaptive, hybrid, logical multi-zone variant of HDFS. GreenHDFS trades performance and power by logically separating the Hadoop cluster into Hot and Cold zones. Their simulation results show that GreenHDFS is capable of achieving 26% savings in the energy costs of a Hadoop cluster in a three-month simulation run. Analytical cost model projects a savings of \$14.6 million in 3-year TCO and simulation results extrapolate a savings of \$2.4 million annually when GreenHDFS technique is applied across all Hadoop clusters (amounting to 38000 servers) at Yahoo [14].

Fig. 2. HDFS architecture.

George Porter has mentioned in his work that they have deployed a prototype of a SuperDataNode in a Hadoop cluster. A SuperDataNode consists of several dozen disks forming a single storage pool. From this pool, several file systems are built, each one imported into a virtual machine running an unmodified copy of the Hadoop DataNode process. They have found that SuperDataNodes are not only capable of supporting workloads with high storage-to-processing workloads, but in some cases actually outperform traditional Hadoop deployments through better management of a large centralized pool of disks. For example, compared to traditional Hadoop, the use of a SuperDataNode reduced total job execution time of a Sort workload by 17%, and a Grep workload by 54% [15].

IV. COMPARATIVE ANALYSIS OF HDFS AND AFS

Properties	HDFS	AFS
Design goals	<ul style="list-style-type: none"> One of the main goals of HDFS is to support large files built based on the assumption that terabyte datasets will be distributed across thousands of disks attached to commodity compute nodes. Used for data intensive computing [16]. Store data reliably, even when failures occur within name nodes, data nodes, or network partitions. HDFS is designed more for batch processing rather than interactive use by users. 	<ul style="list-style-type: none"> Serving the campus community and spanning at least 5000 workstations at Carnegie Mellon University [4]. Design choice was to make the file system compatible with UNIX at the system call level.

Processes	<ul style="list-style-type: none"> Name node and data node 	<ul style="list-style-type: none"> Vice(server) and Venus(client)
File management	<ul style="list-style-type: none"> HDFS supports a traditional hierarchical file organization HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service. [17]. 	<ul style="list-style-type: none"> Location-transparent UNIX file name space to clients. <p>AFS groups files into volumes, making it possible to distribute files across many machines and yet maintain a uniform namespace. A volume is a unit of disk space that functions like a container for a set of related files, keeping them all together on one partition [18].</p>
Scalability	<ul style="list-style-type: none"> Cluster based architecture Hadoop currently runs on clusters with thousands of nodes. E.g. Face book has 2 major clusters: <ul style="list-style-type: none"> A 1100-machine cluster with 8800 cores and about 12 PB raw storage. A 300-machine cluster with 2400 cores and about 3 PB raw storage. Each (commodity) node has 8 cores and 12 TB of storage. EBay uses 532 nodes cluster (8 * 532 cores, 5.3PB). Yahoo uses more than 100,000 CPUs in >40,000 computers running Hadoop <ul style="list-style-type: none"> biggest cluster: 4500 nodes (2*4cpu boxes w 4*1TB disk & 16GB RAM) [19] K. Talatinnis et.al concluded in their work that Hadoop is really efficient while running in a fully distributed mode, however in order to achieve optimal results and get advantage of Hadoop scalability, it is necessary to use large clusters of computers [20]. 	<ul style="list-style-type: none"> client/server ratios of 200:1 Cell based structure. AFS cells can range from the small (1 server/client) to the massive (with tens of servers and thousands of clients [18]) the use of callback-based caching to minimize server and network load which contributes to scaling [21].
Protection	<ol style="list-style-type: none"> The Hadoop Distributed File System (HDFS) implements a permissions model for files and directories that shares much of the POSIX model. file or directory has separate permissions for the user that is the owner, for other users that are members of the group, and for all other users[21]. 	<ul style="list-style-type: none"> AFS uses an access list mechanism for protection. The granularity of protection is an entire directory rather than individual files. Users may be members of groups, and access lists may specify rights for users and groups[4].
Security	<ul style="list-style-type: none"> HDFS security is based on the POSIX model of users and groups Currently security is limited to simple file permissions. The identity of a client process is just whatever the host operating system says it is. Network authentication protocols like Kerberos for user authentication and encryption of data transfers are yet not supported [22]. 	<ul style="list-style-type: none"> Passwords and mutual authentication ensure that only authorized users access AFS file space [18]. Authentication is based on Kerberos[7].
Database files	<ul style="list-style-type: none"> HBase[23] provides Bigtable (Google)[24]-like capabilities on top of Hadoop Core. 	<ul style="list-style-type: none"> Not suitable for database type file and their frequent updates.
File serving	<ul style="list-style-type: none"> HDFS is divided into large blocks for storage and access, typically 64MB in size. Portions of the file can be stored on different cluster nodes, balancing storage resources and demand.[25]. 	<ul style="list-style-type: none"> Whole file(in AFS3 files larger than 64KB are sent in 64KB chunks)[7]. Local files are served like normal Unix like files
Cache management	<ul style="list-style-type: none"> HDFS uses Distributed Cache It is a facility provided by the MapReduce framework to cache application-specific, large, read-only files (text, archives, jars and so on) Private(belonging to one user) and Public (belonging to all the user of the same node) Distributed Cache Files [26]. 	<ul style="list-style-type: none"> Whole file or chunks. Cache is permanent surviving reboot of computer Cache on the local disk, with a further level of file caching by the UNIX kernel in main memory. [27].
Cashe consistency	<ul style="list-style-type: none"> HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access.[17] Client can only append to existing files(yet not supported) 	<ul style="list-style-type: none"> Using callbacks. A callback is a promise by a File Server to a Cache Manager to inform the latter when a change is made to any of the data delivered by the File Server. Callbacks ensure that you are working with the most recent copy of a file [18].
Communication	<ul style="list-style-type: none"> RPC based protocol on the top of TCP/IP 	<ul style="list-style-type: none"> RPC based protocol on the top of TCP/IP
Replication strategy	<ul style="list-style-type: none"> Automatic replication System. Rack based system By default, two copies of each block are stored by different DataNodes in the same rack and a third copy is stored on a DataNode in a different rack (for greater reliability) [25]. An application can specify the number of replicas of a file that should be maintained by HDFS [17]. Replication pipelining in case of write operation. 	<ul style="list-style-type: none"> Volumes containing frequently accessed data can be read-only replicated on several servers [18]. Replicates read-mostly data and AFS system information
Available Implemaentations	<ul style="list-style-type: none"> Yahoo, Facebook, IBM to name a few 	<ul style="list-style-type: none"> There are three major implementations, Transarc (IBM), OpenAFS and ArlaFS. It is also the predecessor of the CFS file system.

V. RECOMMENDATIONS

When sharing of file system among few thousands of users is needed and size of the files is small than AFS is a choice. HDFS is recommended for large files few gigabytes or more and there is stream of data read coming to file system and data is appended rather than overwritten. When small no. of servers have to handle many clients, AFS is generally used as it allows whole file caching so reduces server interactions. HDFS is recommended to manage data intensive applications which are running on low cost commodity hardware.

VI. CONCLUSION

Andrew File System was developed at the Information Technology Center of Carnegie-Mellon University jointly with IBM. Literature shows many experimental results on the usage and performance of AFS at Carnegie-Mellon University. AFS was popular as first wide distributed file system and were used by many organization in many countries. Later on it was supported and developed as a product by Transarc Corporation (now IBM Pittsburgh Labs). IBM branched the source of the AFS product, and made a copy of the source available for community development and maintenance. They called the release OpenAFS. [18]. HDFS is the emerging distributed file system which is an open source counterpart of Google File system. It simplifies the cache coherence model by write-once read many model. Database files cannot be stored in HDFS as they required frequent updates. Write-append are yet to implement in it. As this is yet emerging and not fully matured security aspects are poor in current versions. Certain APIs are written on top of Hadoop to retrieve data with SQL like programming. HDFS is used to store files which are used in large scale distributed data processing where AFS is used to provide abstraction of local unix like file system to distributed users connected in client server environment

REFERENCES

- [1] P. K. Sinha, "Distributed operating system-concepts and design," *IEEE press*, 1993, ch. 9, pp. 245-256.
- [2] D. Borthakur. HDFS Architecture Guide. [Online]. Available: hadoop.apache.org/docs/current/hdfs_design.html
- [3] S. Ghemawat, H. Gobioff, and S. Leung. (Oct. 2003). The Google File System. *SOSP '03 ACM*. [Online]. Available: <http://Research.google.com/archive/gfs.html>
- [4] M. Spasojevic and M. Satyanarayanan, "An Empirical study of a wide-area distributed file system," *ACM Transactions on Computer Systems*, vol. 14, no. 2, pp. 200-222, May 1996.
- [5] J. H. Howard, "An overview of the andrew file system," in *Proc. the USENIX Winter Technical Conference*, 1988, pp. 1-6.
- [6] M. Satyanarayanan, "Scalable, secure, and highly available distributed file access," *IEEE Computer*, vol. 23, no. 5, pp. 9-21, 1990.
- [7] J. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed systems-concepts and design*, 4th edition, pearson education, 2009, ch. 8, pp. 322-330.

- [8] J. Dean and S. Ghemawat, "Map reduce: simplified data processing on large clusters," in *Proc. Sixth Symposium on Operating System Design and Implementation*, December, 2004, pp. 137-150.
- [9] R. Khare, D. C. K. Sitaker, and A. Rifkin, *Nutch: A flexible and scalable open-source web search engine*, Oregon State University, 2004.
- [10] K. Talatinnis, A. Sidiropoulou, K. Chalkias, and G. Stephanides, "Parallel collection of live data using Hadoop," in *Proc. IEEE 14th Panhellenic Conference on Informatics*, 2010, pp. 66-71.
- [11] F. Wang, J. Qiu, J. Yang, B. Dong, X. H. Li, and Y. Li, "Hadoop high availability through metadata replication," in *Proc. the first international workshop on Cloud data management*, 2009, pp. 37-44.
- [12] D. Borthakur. (2007). Hadoop Distributed file system -Architecture and Design. [Online]. Available: http://hadoop.apache.org/common/docs/r0.17.1/hdfs_design.pdf
- [13] J. Leverich and C. Kozyrakis, "On the energy (in) efficiency of hadoop clusters," *SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 61-65, January 2010.
- [14] R. T. Kaushik and M. Bhandarkar, "Green HDFS: towards an energy-conserving, storage-efficient, hybrid hadoop compute cluster," in *Proc. international conference on Power aware computing and systems*, 2010, pp. 1-9.
- [15] George Porter, "Decoupling storage and computation in Hadoop with SuperDataNodes," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 41-46, 2010.
- [16] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio, "The changing paradigm of data-intensive computing," *IEEE Computer Society*, vol. 42, no. 1, pp. 26-34, Jan. 2009.
- [17] J. Hanson. (Feb 2011). An Introduction to Hadoop Distributed File System. [Online]. Available: <http://www.ibm.com/developerworks/web/library/wa-introhddfs>
- [18] Open AFS user Guide. [Online] Available: <http://docs.openafs.org/User Guide>
- [19] Hadoop Wiki. [Online]. Available: <http://wiki.apache.org/hadoop/PoweredBy>
- [20] K. Talatinnis, A. Sidiropoulou, K. Chalkias, and G. Stephanides, "Parallel collection of live data using Hadoop," in *Proc. 14th IEEE Panhellenic Conference on Informatics*, pp. 66-71, 2010.
- [21] M. Satyanarayanan and M. Spasojevic, "AFS and the web: competitors or collaborators?" in *Proc. of the Seventh ACM SIGOPS European Workshop*, pp. 1-6, 1996.
- [22] HDFS User guide. [Online] Available: http://hadoop.apache.org/common/docs/current/hdfs_user_guide.html
- [23] Hbase Big Table - like structured storage for Hadoop HDFS. [Online]. Available: <http://wiki.apache.org/hadoop/Hbase>
- [24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. W. M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proc. of the 7th conference on usenix symposium on operating systems design and implementation - vol. 7*, 2006, pp. 1-14.
- [25] J. Shafer, S. Rixner, and A. L. Cox, "The Hadoop distributed filesystem: balancing portability and performance," *IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pp. 122-133, 2010.
- [26] MapReduce tutorial. [Online]. Available: http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html
- [27] M. Satyanarayanan, "The influence of scale on distributed file system design," *IEEE transactions on software engineering*, vol. 18, no. 1, pp. 1-8, 1992.

Monali Mavani received her B.Tech (electronics and communication) and MBA (systems) from SNDT university and Madurai kamaraj University in 2002 and 2010 respectively. She is currently doing Masters in engineering (M.E.) in computers from Mumbai University. She is currently faculty in SIES College of management studies in MCA department since 2004. Her research areas are distributed computing, data analytics, network security.