Data storage in Big Data Context: A Survey

A.ELomari, A.MAIZATE*, L.Hassouni#
RITM-ESTC / CED-ENSEM, University Hassan II
Km 7, Eljadida Street, B.P. 8012 Oasis,
Casablanca, Morocco
akramelomari@yahoo.fr, *maizate@hotmail.com, #lhassouni@hotmail.com

***Abstract*— As data volumes to be processed in all domains; scientific, professional, social …etc., are increasing at a high speed, their management and storage raises more and more challenges. The emergence of highly scalable infrastructures has contributed to the evolution of storage management technologies. However, numerous problems have emerged such as consistency and availability of data, scalability of environments or yet the competitive access to data. The objective of this paper is to review, discuss and compare the main characteristics of some major technological orientations existing on the market, such as Google File System (GFS) and IBM General Parallel File System (GPFS) or yet on the open source systems such as Hadoop Distributed File System (HDFS), Blobseer and Andrew File System (AFS), in order to understand the needs and constraints that led to these orientations. For each case, we will discuss a set of major problems of big data storage management, and how they were addressed in order to provide the best storage services.**

Keywords— Big Data; Data Storage; Blobs ; HDFS; GFS; AFS; GPFS; Blobseer

I. INTRODUCTION

Todays, the amount of data generated during a single day may exceed the amount of information contained in all printed materials all over the world. This quantity far exceeds what scientists have imagined there are just a few decades. Internet Data Center (IDC) estimated that between 2005 and 2020, the digital universe will be multiplied by a factor of 300, so it will pass from 130 Exabyte to 40,000 Exabyte, the equivalent of more than 5,200 gigabytes for each person in 2020 [1].

The traditional systems such as centralized network-based storage systems (client-server) or the traditional distributed systems such as NFS, are no longer able to respond to new requirements in terms of volume of data, high performance, and evolution capacities. And besides their cost, a variety of technical constraints are raised, such as data replication, continuity of services etc.

In this paper, we try to discuss a set of technologies used in the market and that we think the most relevant and representative of the state of the art in the field of distributed storage systems.

978-1-5090-4926-4/16/$31.00 ©2016 IEEE

II. WHAT IS "DISTRIBUTED FILE SYSTEMS (DFS)"

A distributed file system (DFS) is a system that allows multiple users to access, through the network, a file structure residing on one or more remote machines (File Servers) using

a similar semantics to that used to access the local file system. This is a client / server architecture where data is distributed across multiple storage spaces usually called nodes. These nodes consist of a single or a small number of physical storage disks residing usually in basic equipment, configured to only provide storage services. As such, the material can be relatively low cost.

As the material used is generally inexpensive and by large quantities, failures become unavoidable. Nevertheless, these systems are designed to be tolerant to failure by having recourse to data replication which makes the loss of one node an event "of minimal emergency" because data is always recoverable, often automatically, without any performance degradation.

III. DFS ARCHITECTURES

A. Andrew File System (AFS) architecture

AFS (or OpenAFS currently) is a standard distributed file system originally developed by Carnegie Mellon University. It is supported and developed as a product by Transarc Corporation (now IBM Pittsburgh Labs). It offers a client-server architecture for federated file sharing and distribution of replicated read-only content [2].

AFS offers many improvements over traditional systems. In particular, it provides the independence of the storage from location, guarantees system scalability and transparent migration capabilities.
As shown in Figure 1, the distribution of processes in AFS can be summarized as follows:

- A process called "Vice" is the backbone of information sharing in the system; it consists of a set of dedicated file servers and a complex LAN.
- A process called "Venus" runs on each client workstation; it mediates access to shared files [3].

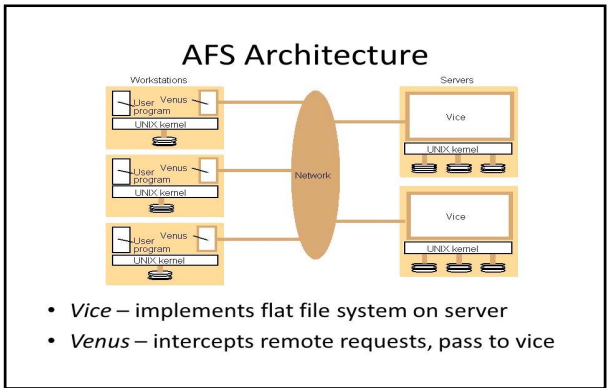

Figure 1 : AFS Design

.

AFS logic assumes the following hypothesis [4]:

- Shared files are rarely updated and local user files will remain valid for long periods.
- An allocation of a large enough local disk cache, for example 100 MB, can keep all users files.

Using the client cache may actually be a good compromise to system performance, but it will only be effective if the assumptions adopted by AFS designers are respected, otherwise this can make a huge issue for data integrity.

B. Google File System (GFS) architecture

Another interesting approach is that proposed by GFS, which is not using special cache at all.

GFS is a distributed file system developed by Google for its own applications. Google GFS system (GFS cluster) consists of a single master and multiple Chunkservers (nodes) and is accessed by multiple clients, as shown in Figure 2 [5]. Each of these nodes is typically a Linux machine running a server process at a user level.

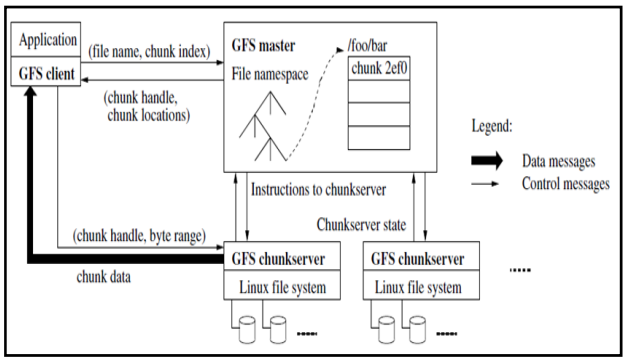

Figure 2 : GFS Design

The files to be stored are divided into pieces of fixed size called "chunks". The Chunkservers store chunks on local disks as Linux files. The "master" maintains all metadata of the file system. The GFS client code uses an application programming interface (API) to interact with the master regarding transactions related to metadata, but all communications relating to the data themselves goes directly to Chunkservers.

Unlike AFS, neither the client nor the Chunkserver use a dedicated cache. Customer's caches, according to Google, offer little benefit because most applications use large which are too big to be cached. On the other hand, using a single master can drive to a bottleneck situation. Google has tried to reduce the impact of this weak point by replicating the master on multiple copies called "shadows" which can be accessed in read-only even if the master is down.

C. Blobseer architecture

Blobseer is a project of KerData team, INRIA Rennes, Brittany, France[6]. The Blobseer system consists of distributed processes (Figure 3), which communicate through remote procedure calls (RPC). A physical node can run one or more processes and can play several roles at the same time.

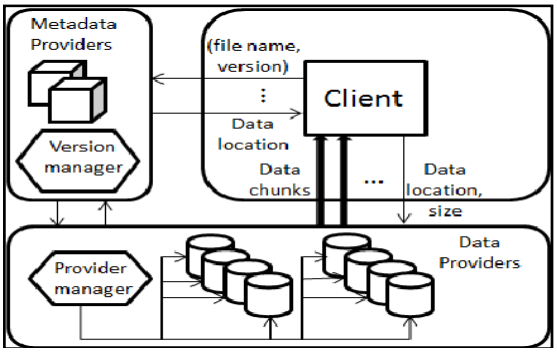

Figure 3 : Blobseer Design

Unlike Google GFS, Blobseer do not centralize access to metadata on a single machine, so that the risk of bottleneck situation of this type of node is eliminated. Also, this feature allows load balancing the workload across multiple nodes in parallel.

D. Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is a component of Apach Hadoop project [7]. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.

As shown in figure 4, HDFS stores file system metadata and application data separately. As in other distributed file systems, HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes [8].

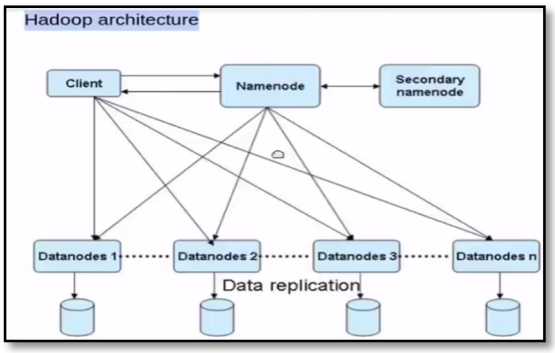

Figure 4: HDFS Design

There is one NameNode per cluster and it makes all decisions regarding replication of blocks [9].

IV. Data Storage as blob

The architecture of a distributed storage system must take into consideration how files are stored on disks. One smart way to make this possible is to organize these data as objects of considerable size. Such objects, called Binary Large Objects (BLOBs), consist of long sequences of bytes representing unstructured data and can provide the basis for a transparent data sharing of large-scale. A BLOB can usually reach sizes of 1 Tera Byte (TB).

Using BLOBs offers two main advantages:

- The Scalability: Maintaining a small set of huge BLOBs including billions of small items is much easier than directly managing billions of small ones. The simple mapping between the application data and file names can be a big problem compared to the case where the data are stored in the same BLOB and that only their offsets must be maintained.
- The Transparency: A data management system based on shared BLOBs, uniquely identifiable through ids, relieves application developers of the burden of explicit management and transfer of their locations on the codes. The system thus offers an intermediate layer that masks the complexity of access to data wherever it is stored physically [10].

V. Data striping

Data striping is a well-known technique for increasing the data access performances. Each BLOB or file is divided into small pieces that are distributed across multiple machines on the storage system. Thus, requests for access to data may be distributed over multiple machines in parallel way, allowing achieving high performances.
Two factors must be considered in order to maximize the benefits of this technique:

- Configurable strategy of distribution of chunks: Distribution strategy specifies where to store the chunks to achieve a predefined goal. For example, load balancing is one of the goals that such strategy can allow.
- Dynamic configuration of the size of the chunks: If the chunks size is too small, applications would have to retrieve the data to be processed from several chunks. On the other hand, the use of too large chunks will complicate simultaneous access to data because of the increasing probability that two applications require access to two different data but both stored on the same chunk.

A lot of systems that use this type of architecture, such as GFS and Blobseer use a 64 MB sized chunks, which seems to be the most optimized size for those two criteria.

VI. concurrency

Processing concurrency is very dependent on the nature of the desired data processing and of the nature of data changes. For example, Haystack system that manages Facebook pictures which never changes [11], will be different from Google GFS or IBM General Parallel File System (GPFS) which are managing a more dynamic data.

The “lock” method is used by many DFS to manage concurrency and IBM GPFS has developed a more effective mechanism that allows locking a byte range instead of whole files/blocks (Byte Range Locking) [12].

GFS meanwhile, offers a relaxed consistency model that supports Google highly distributed applications, but still relatively simple to implement.

Blobseer developed a more sophisticated technique, which theoretically gives better results. The "snapshot" approach using versioning that Blobseer brings is an effective way to meet the main objectives of maximizing competitive access [13]. The disadvantage of such a mechanism based on snapshots, is that it can easily explode the required physical storage space. However, although each write or append generates a new version of the blob snapshot, only the differential updates from previous versions are physically stored.

VII. DFS Benchmark

As we have detailed in this article, generally there is no better or worse methods for technical or technological choices to be adopted to make the best of a DFS, but rather compromises that have to be managed to meet very specific objectives.

In Table 2, we compare five distributed file systems: GFS, GPFS, HDFS, AFS and Blobseer. Choosing to compare only those specific systems despite the fact that the market includes dozens of technologies is led particularly by two points:

1. It is technically difficult to study all systems in the market in order to know their technical specifications, especially as several of them are proprietary and closed systems. Even more, the techniques are similar in several cases and are comparable to those of the five we compared.

2. Those five systems allow making a clear idea about the DFS state of the art thanks to the following particularities:

- GFS is a system used internally by Google, which manage huge quantities of data because of its activities.
- GPFS is a system developed and commercialized by IBM, a global leader in the field of Big Data
- HDFS is a subproject of HADOOP, a very popular Big Data system...
- Blobseer is an open source initiative, particularly driven by research as it is maintained by INRIA Rennes.
- AFS is a system that can be considered as a bridge between conventional systems such as NFS and advanced distributed storage systems.

In Table 2, we compare the implementation of some key technologies in those five systems.

Analysis of the results of Table 2 leads to the following conclusions:
- The five systems are expandable in data storage. Thus, they cover one of the principal issues that lead to the emergence of Distribute File System.
- Only Blobseer and GPFS offer the extensibility of metadata management to overcome the bottleneck problem of the master machine, which manage the access to metadata.
- Except AFS, all studied systems are natively tolerant to crash, relying essentially on multiple replications of data.
- To minimize the lag caused by locking the whole file, GPFS manage locks on specific areas of the file (Byte range locks). But the most innovative method is the use of versioning and snapshots by Blobseer to allow simultaneous changes without exclusivity.
- Except AFS, all systems are using the striping of data. As discussed earlier; this technique provides a higher input / output performance by "striping" blocks of data from individual files over multiple machines.
- Blobseer seems to be the only one among the systems studied that implements the storage on blobs technique, despite the apparent advantages of such technique.
- To allow a better scalability, a DFS system must support as much operating systems as possible. But while AFS, HDFS and GPFS supports multiple platforms, GFS and Blobseer run exclusively on Linux, this can be explained partly by the commercial background of AFS, HDFS and GPFS.
- Using a dedicated cache is also a point of disagreement between systems. GFS and Blobseer consider that the cache has no real benefits, but rather causes many consistency issues. AFS and GPFS uses dedicated cache on both client computers and servers. HDFS seems to use dedicated cache only at client level.

Table 2 Comparative table of most important characteristics of distributed file storage

	GFS by Google	GPFS IBM	HDFS	Blobseer	AFS (OPEN FS)
Data Scalability	YES	YES	YES	YES	YES
Meta Data Scalability	NO	YES	NO	YES	NO
Fault tolerance	Fast Recovery. Chunk Replication. Master Replication.	Clustering features. Synchronous and asynchronous data replication.	Block Replication. Secondary NameNode.	Chunk Replicatio n Meta data replication	NO
Data access Concurrency	Optimized for concurrent appends	Distributed byte range locking	Files have strictly one writer at any time	YES	Byte-range file locking
Meta Data access Concurrenc y	Master shadows on read only	Centralized management	NO	YES	NO
Snapshots	YES	YES	YES	YES	NO
Versioning	YES	unknown	NO	YES	NO
Data Striping	64 MB Chunks	YES	YES (Data blocks of 64 MB)	64 MB Chunks	NO
Storage as Blobs	NO	NO	NO	YES	NO
Supported OS	LINUX	AIX, Red Hat, SUSE , Debian Linux distributions, Windows Server 2008	Linux and Windows supported , BSD, Mac OS/X, Open Solaris known to work	LINUX	AIX, Mac OS X, Darwin, HP-UX, Irix, Solaris, Linux, Windows, FreeBSD, NetBSD OpenBSD
Dedicated cache	NO	YES by AFM technology	YES (Client)	NO	YES

VIII. CONCLUSION

In this paper, we reviewed some specifications of distributed file storage systems. It is clear from this analysis that the major common concern of such systems is scalability. A DFS should be extendable with the minimum cost and effort.

In addition, data availability and fault tolerance remains among the major concerns of DFS. Many systems tend to use non expensive hardware for storage. Such condition will expose those systems to frequent or usual breakdowns.

To these mechanisms, data striping and lock mechanisms are added to manage and optimize concurrent access to the data. Also, Working on multiples operating systems can bring big advantages to any of those DFS.

None of these systems can be considered as the best DFS in the market, but rather each of them is excellent in the scope that it was designed for.

[1] John Gantz and David Reinsel. THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. Tech. rep. Internet Data Center(IDC), 2012.
[2] OpenAfs : www.openafs.org/
[3] Monali Mavani : Comparative Analysis of Andrew Files System and Hadoop Distributed File System, 2013.
[4] Stefan Leue : Distributed Systems – Fall, 2001
[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung Google* : The Google File System.
[6] Blobseer: blobseer.gforge.inria.fr/
[7] Hadoop: hadoop.apache.org/
[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler Yahoo!: The Hadoop Distributed File System, 2010.
[9] Dhruba Borthakur : HDFS Architecture Guide, 2008.
[10] Bogdan Nicolae, Gabriel Antoniu, Luc Boug_e, Diana Moise, Alexandra, Carpen-Amarie : BlobSeer: Next Generation Data Management for Large Scale Infrastructures, 2010.
[11] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel, Facebook Inc: Finding a needle in Haystack: Facebook's photo storage,
[12] Scott Fadden, An Introduction to GPFS Version 3.5, Technologies that enable the management of big data, 2012.
[13] Bogdan Nicolae,Diana Moise, Gabriel Antoniu,Luc Boug´e, Matthieu Dorier : BlobSeer: Bringing High Throughput under Heavy Concurrency to Hadoop Map-Reduce Applications, 2010.