Dutse Journal of Pure and Applied Sciences (DUJOPAS) Vol. 2 No. 2 December 2016

Cluster Based File Systems
(A Review)

Aminu Aliyu Abdullahi
Department of Computer Science
Federal University Dutse.

Abstract

With a growth in the size of Internet users and other big data-reliant applications, the need for fault tolerant file systems that can provide concurrent and redundant access to files is growing. The Access, location, concurrency and failure transparency provided by Cluster based file systems has inspired many big data users like Google and Oracle to adopt Cluster based file systems. The heterogeneity and transparency provided by such systems also dampens expansion and managerial costs, however the management of such systems is not trivial.

Cluster based file systems often require complex operations to ensure synchronicity and concurrency control within the cluster. The immense distributed management of clusters also creates a performance overhead especially when involving remote requests. Cluster based systems are thus challenged to provide the seemingly redundant and resilient operations rooted in their architectures while not compromising on performance, security and other functionality metrics. This review will focus on the strategies adopted in the optimal deployment of cluster based file systems and the schemes used in some of them to overcome some of the challenges faced by cluster based systems. The review will focus on File clustered file systems (Hadoop File System, Lustre, Gluster File system, Ceph, Moose file system and Quick File system). These systems were selected based on their recentness. Additionally, servers like Hadoop are being used by industry leaders like Yahoo and Facebook and hence more prone to rigorous standardization. The survey will review them based on their Architectures, Naming method, Replication mechanism and fault management and detection mechanism.

Keywords: *Cluster Based File Systems; Network File Systems; Distributed File Systems*

Introduction

Distributed File systems are file systems designed to facilitate the distribution of data files across multiple computing systems over a network platform. These distributed systems are usually designed to provide access, location, concurrent-access, and failure transparency. The capacity for data migration, scalable composition and heterogonous configuration may also be design aims. (Guan *et al.* 2000).

According to Guan *et al.* (2000) and Satyanarayanan (1990), these distributed systems evolved from legacy time-sharing systems and other network attached storage systems invented in the early days of computing. With the development of the workstation and other network access media, the need of distributed file systems increased and became even greater with the development of the Internet.

According to Thanh *et al.* (2008), distributed file systems are usually classified according to their architecture, processes, communication protocol used, synchronization mechanism and other functional parameters (Thanh *et al.* 2008).

The process-based classification investigates the nature of the computing processes that manages the distribution of files. It mainly focuses on whether a system is a state-full or stateless system. According to Thanh *et al.* (2008) state full systems are systems that that have the ability to store the state of operations and other access parameters and or credentials of the various systems and or nodes within the overall system architecture, while stateless systems are systems that do not process such capabilities. Although from the onset a state full system might seem advantageous, stateless systems According to Levy and Silberschatz (1990) have overcome the problem of potential catastrophic failure due to server failures by ignoring systems states when performing operations. In these state less systems each request or operation is an encircling whole that requires no pre requisite 'states'. (Levy and Silberschatz, 1990).

The communication protocol classification explores the underlying protocol used by the distributed systems to communicate over a network, these are usually TCP or UDP or a combination of both. The naming categorization explores the mechanisms used by the file systems to manage file names across the different computing platforms. Naming schemes according to Thanh *et al.* (2008) might choose to combine logical paths while others might choose to ignore such. Additionally, naming schemes might employ a central server called a Central Meta data server to manage the name space of the system. A contrasting structure will distribute the name spaces Meta data across all or some other nodes. (Thanh *et al.* 2008).

Other categorizations of distributed file systems involve classifying them according to consistency, fault tolerance and security mechanisms (Thanh *et al.* 2008). This review however will focus on a subset of the architectural classification of File systems

The architectural classification focuses on the fundamental structure used to distribute files and to manage such distribution, Client server architectures, parallel architectures, asymmetric architectures, asymmetric architectures and cluster-based architectures are the chief categories in this classification (Than *et al.* 2008).

Client server architectures involve the use of a dedicated system or systems as providers of service or data called servers and the receivers of the data or service called clients (Thanh *et al.* 2008). According to Thanh *et al.* (2008) The use of client server architectures allows a master server to monitor the interactions within the system, this makes the employment of heterogeneous resources (different local file systems) easier as the management is done in a centralized place. This architecture however risks creating a single failure point. The failure of the server or connections to it will render the entire file system dysfunctional.

Cluster based systems on the other hand provide a master server along with numerous other servers connected across multiple links (Thanh *et al.* 2008). A system with a single metadata server is referred to as a Centralized cluster based file system while a system with more than one meta data servers is called a distributed cluster based file system (Depardon *et al.* 2013).

Analysis of Selected File Systems

Although the diverse approaches adapted to tackling the lingering issues of File Clusters has meant that it is difficult to have a standard taxonomy classifying all of them, This review will attempt to categorize them based on some of the most prominent features that define distributed file systems. The review would explore five clustered file system (Hadoop, Lustre, Gluster, Ceph, Moose and QFS) based on their Architecture, Naming Method, Replication Mechanism and Fault management method.

Architecture
According to Depardon *et al.* (2013), The architectural classification of Clustered file systems is usually based on the architectural definition of their metadata servers. This measures the relation of the meta data service of the file system to the rest of the cluster. A Centralized distributed architecture has a single metadata server while a Distributed Server architecture harbors multiple metadata servers. Occasionally however certain clusters are designed with hybrid architecture. Some systems aiming at extreme redundancy might employ parallelism and or multi-threading tools to combine a centralized architecture with a distributed one (Depardon *et al.* , 2013).

The Hadoop Distributed File System (HDFS) employs a centralized architecture. A single meta data server (called a namenode) is used to service the cluster. According to Shvachko *et al.* (2010) however, in a bid to ensure redundancy Hadoop employs a backup metadata server called the secondary namenode. The metadata on the namenonde is steadfastly replicated on the secondary namenode so that in case a failure, the secondary namenode can seamlessly replace the namenode.

Lustre, according to Yu *et al.* (2007) employs the same binary meta data server architecture as Hadoop. Its centralized architecture utilizes two metadata servers called Metadata Targets. The two servers are then used as primary and secondary (Passive/Active) servers to provide a substitute in case of a server failure. Additionally, Lustre stores file data in file disks encompassed in an Object Storage Target which in turn are encompassed in Object Storage Servers (Yu *et al.* 2007).

The Gluster File System is a distributed clustered file system that does not utilize metadata servers. It employs a hash function-mapping scheme to store metadata data on different devices attached to different servers. This approach was employed to ensure simplicity and scalability in the design of the file system (Depardon *et al.* 2013). However, although such hash functions are relatively secure and collision resistant, such a scheme might subject Gluster to exploitation by a future innovation in hash function cryptanalysis.

Ceph is a distributed clustered file system. It employs a distributed Meta data architecture. A series of Meta data servers employ numerous Object Storage Devices to manage data and metadata alike (Depardon *et al.* 2013).

Moose File System (MooseFS) utilizes centralized clustered server architecture. It employs a central Metadata server and numerous other backup metalogger servers that replicate the data on the central server and are promoted to central server when there is a server failure (Depardon *et al.* 2013).

Quick file System (QFS) employs the same centralized architecture as MooseFS. A central metadata server services the cluster with numerous other potential servers that can take over in the event of service failure. (Oracle Corporation, 2011).

Naming Method
The management of a Files Systems namespace is usually dependent upon the architecture of the file system; this is especially prevalent in clustered file systems, as file names in the name space must be mapped to the respective metadata of the files.

The namespace of the Hadoop file system (HDFS) is managed by a central metadata server (namenode). The logical to physical mapping between file names (which are structured in a hierarchical fashion) and storage sectors is also handled by the namenode. This mapping function is transferred to the secondary namenode in the event of a server failure (Shyachko *et al.* 2010).

The central Metadata server manages Lustre's namespace. It employs inodes like traditional unix file systems to map file names to their respective Object Storage Targets. In the case of large files, more than one inode may be associated with an Object Storage Target (Yu *et al.* 2007).

Gluster file system employs a hash function-mapping scheme to map filenames to their storage sectors and their metadata. This Elastic Hashing Algorithm then presents the system with a globalized namespace that location transparent (Dapardon *et al.* 2013).

The distributed metadata architecture of Ceph uses the Unix file system's inodes and Controlled Replication Under Hashing function (CRUSH); a pseudo random data distribution algorithm, to map file names to storage blocks. Crush allows Ceph to evenly distribute the mapping task amongst its numerous metadata servers their by providing scalability. CRUSH is also used by the clients to probabilistically but accurately compute the address of a file's storage sector from its inodes (Depardon *et al.* 2013).

The central metadata server in Moose file system handles the system's namespace. It maps filenames to their physical location. (Depardon *et al.* 2013).

The namespace structure in Quick file system is the same as MooseFs. The central metadata server maps inodes to their physical storage with the aide of a Disk Allocating Unit (DAU). The DAU is used to reduce server overhead during file access operations and to efficiently structure the geometry of the physical storage of the servers (Oracle Corporation, 2011).

Replication Mechanism

Because data availability is an important motivation for the creation of clustered file systems, the process ensuring the presence of redundant copies of the same data is important. This process called replication has proven widespread amongst cluster based file systems (Depardon *et al.* 2013).

HDFS divides data into series of data blocks and distributes them amongst data nodes while ensuring that a data node has only a single copy of a data block. The namenodes handles this task according to predefined replication policy. It also periodically assesses the replication metrics of data blocks in order to ensure that no block is over or under replicated (Depardon *et al.* 2013).

According to Depardon *et al.* (2013). Lustre does not provide data level replication; therefore the failure of a service node will deprive the rest of the system any access to the data stored on the node. This shortcoming makes Lustre unsuitable for peer-to-peer services and other availability dependent services. Third party applications like the JOSHUA project and Intel's exist however that can provide Lustre clusters with some level of data replication.

According to Depardon *et al.* (2013), Gluster File System relies on Redundant Array of Independent Disks (RAID) virtualization to ensure data replication. This method of data replication however can only replicate data on the same physical volume and is therefore handicapped.

Ceph uses the Controlled Replication Under Hashing algorithm (CRUSH) to replicate data. It according to Depardon *et al.* (2013) uses the same policy as the Hadoop file system to replicate data on tripartite bases while using the probabilistic capabilities of CRUSH to distribute such replicates.

Moose file system replicates data on a uniform basis across its servers using a uniform method. A standard variable called goal maintains the uniformity of the number of replicated data across the system. When a certain server is chosen to harbor a data block, the server sends replications instructions to other servers according to the goal of the system.(Depardon *et al.* 2013).

Replication in the Quick File system (QFS) is handled on user-defined bases. In chunk replication, a data block is replicated in chunks a number of times, this number is a user defined variable initiated when the replication functionality is activated. A variable to the chunk replication is the Solomon Reed encoding which splits the data block instead of replicating it wholly as a chunk. (Oracle Corporation, 2012).

Fault Management

The High performance and redundancy definitions of cluster based file systems means that they must have appropriate fault detection and management mechanisms in order to detect server or connection failures and if possible adapt to such failures. Instruments like replication also create consistency issues that may result in errors if not properly handled. According to Depardon *et al.* (2013). As with other distributed file systems, Instruments like caching are usually used to increase performance, these however create their own cache consistency problems.

Intermittent messaging between servers called heartbeats are according to Depardon *et al.* (2013) used to detect server failures, these sort of networking inspired methods however consume the networks bandwidth and therefore require efficient utilization to ensure that they do not affect primary functionalities

Hadoop File system maintains data consistency by an hourly exchange of Block reports. These reports contain updated information about data blocks stored within the nodes. Server failures are detected by an exchange of heartbeats on a 3 second bases. Additionally, at startup Hadoop nodes undergo a process called registration, which compares the series of Identifiers that rare assigned to each node on installation (Depardon *et al.* 2013).

According to Depardon *et al.* (2013), Lustre employs the standard Lightweight Directory Access Protocol for fault detection and management. LDAP is used to make requests to metadata servers before file access and to switch to secondary servers when a server failure is detected.

Gluster Files Systems detects server failure on an input output request to a particular node. A non-reply is used to indicate service failure and the node is then taken off the cluster (Depardon *et al.* 2013).

Ceph file system uses the intermittent exchange of heartbeat messages to indicate service failure. Detected nodes called monitors are also used by Ceph to detect failures. These monitors have an overall map of the cluster and routinely monitor the cluster to detect failed and reactivated servers. A periodic exchange of information between monitors and nodes is used to maintain this cluster map (Depardon, 2013).

Moose File System detects server failure on an input output request to a particular node. A non-reply is used to indicate service failure and the node is then taken off and added to a quarantine list (Depardon *et al.* 2013).

Oracle Corporation's Storage and Arching Manager SAM is incorporated into the Quick File System to provide redundancy in the event of server failure. SAM according to Oracle Corporation provides a transparent and seamless archiving system for stored data on QFS. (Oracle Corporation, 2012).

Comparison

Depardon *et al.* (2013) performed a comparative studu between HDFS, MooseFs, Lustre, GlusterFS, iRODS and Ceph. While accessing their architectural optimism, They found that Lustre, HDFS and MooseFS are computationally limited to the computing power of a single computer in the cluster, that is the number of

computations that can be performed depends on the computing capability of a single server in cluster (Depardon *et al.* , 2013). GlusterFS however becaouse it combines metadata and data management on the same servers was not as handicapped. Ceph howerevr alghogh it distributes data and metadata management was found to be as efficient as GludsterFS (Depardon *et al.* , 2013), this may be as result of the Controlled Replication Under Hashing algorithm (CRUSH) it uses for data mapping.
HDFS, MooseFS and Lustre where additionally found to function more efficiently when storing a small quantity of large file files while Ceph and Gluster were found suitable to hold both (Large number of large files) (Depardon *et al.* , 2013). This may be a result of GluserFS's distributed data management policy and Ceph's use of CRUSH.

When measuring data access times, gauging an input output threshold over a 20-gigabyte data and a 1-mega byte data, both GlusterFS and Ceph were also found to be more efficient than MooseFS, Lustre and HDFS in both cases. With Ceph measuring an I/O of 419 seconds and 382 seconds over 20GB and GlusterFS measuring 341s and 403 respectively. (Depardon *et al.* 2013).

Reflections and Future Recommendations
Efficient data replication, concurrent access, processing performance and fault tolerance are amongst the major challenges of designing an efficient cluster based file system.

Although some of the reviewed cluster systems have adopted good methods of data replication, none of the systems has so far been adopted to use intelligent methods to guide the replication of data. Such stochastic strategies have been adopted with encouraging results to the problem of Data Replication on Grid Computing by Lamehamedi *et al.* (2002). By using stochastic strategies that guide a highly replicative data system to replicate data based on usage and prospective usage, computing overhead due to unnecessary replication could be avoided further increasing the overall efficiency of the system.

Load balancing and concurrency control could also be improved using a cost model. A negotiating based cost model is already in use in numerous internet protocols (Wei *et al.* 2004), such models could be adopted to negotiate load sharing amongst the server nodes of a cluster based file system. As demonstrable by the efficiency of such models (such as negotiation based network routing protocols), the potential benefit of a balanced and efficiently running system might supersede any potential overhead that could be incurred through such a negotiation.

Fault tolerance in computer systems disregarding security issues, is usually a function of computing redundancy (Stapper, 1989). The use of highly distributed cluster based file system architectures can prove efficient. A potential communication overhead could be remedied using the load balancing strategies proposed above.

Failure prediction as explored by Yawei and Zhiling (2006) could also prove beneficial. Adopting check-pointing and other computational methods to predict server failures and restore server state before the occurrence of catastrophic failure may remedy some of the potential problems of losing metadata servers and of server failures in statefull file systems.

Conclusion
Although this review has used only four classification schemes to review Cluster Based File Systems, numerous other taxonomy's exist. Application Program Interface (API) access, client access, Load balancing and cache consistency have been used amongst others. Additionally, Oracle Corporations Quick File Systems has to best of our knowledge not been suitably compared with the other file systems reviewed in this work. It

is probable that its distributed structure and archiving instruments might give it an edge compared with the other systems.

Ceph File System's use of the Controlled Replication Under Hashing algorithm (CRUSH) has been found to have given it a competitive edge. In all the measurements of Depradon *et al.* (2013) Ceph has out competed MooseFs, Lustre and HDFS while coming at par with GlusterFS, it can then be deduced that it is the employment of the elastic hashing algorithm of CRUSH that has given it this computational advantage. But a future successful cryptanalysis of cryptographic hash functions would probably render it vulnerable.

The performance of GlusterFS has also highlighted the inherent advantages of combining metadata and data management while separating metadata and data storage. Observing the performances recorded by Depardon *et al.* (2013) we might surmise that such an architectural configuration adds to the systems scalability and they fore granting it a computational advantage.

References

Depardon, B., Le Mahec, G. & Séguin, C. (2013) 'Analysis of six distributed file systems', .

Depardon, B., Le Mahec, G. & Séguin, C. (2013) 'Analysis of six distributed file systems', .

Guan, P., Kuhl, M., Li, Z. & Liu, X. (2000) 'A survey of distributed file systems', *University of California, San Diego,* 1 .

Levy, E. & Silberschatz, A. (1990) 'Distributed file systems: Concepts and examples', *ACM Computing Surveys (CSUR),* 22 (4), pp.321-374.

Oracle.com (2011) 'Sun QFS and sun storage archive manager 5.2', *Sun QFS and Sun Storage Archive Manager 5.2[Online].* Available at: http://docs.oracle.com/cd/E22576_01/misc/SAMQ52.pdf .

Satyanarayanan, M. (1990) 'A survey of distributed file systems', *Annual Review of Computer Science,* 4 (1), pp.73-104.

Shvachko, K., Kuang, H., Radia, S. & Chansler, R. (2010) "The hadoop distributed file system", *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on,* IEEE. p1-10.

Stapper, C.H. (1989) 'Large-area fault clusters and fault tolerance in VLSI circuits', *IBM Journal of Research and Development,* 33 (2), pp.162-173.

Thanh, T.D., Mohan, S., Choi, E., Kim, S. & Kim, P. (2008) "A taxonomy and survey on distributed file systems", *Networked Computing and Advanced Information Management, 2008. NCM'08. Fourth International Conference on,* IEEE. p144-149.

Yawei Li & Zhiling Lan (2006) "Exploit failure prediction for adaptive fault-tolerance in cluster computing", *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on, Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on,*p8 pp.-538.

Yu, W., Vetter, J., Canon, R.S. & Jiang, S. (2007) "Exploiting lustre file joining for effective collective io", *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on,* IEEE. p267-274.*Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on,*IEEE. P267-274.

Depardon, B. G. & Séguin, C. (2015) 'Clusters in distributed file systems', .