A PROPOSED RACK-AWARE MODEL FOR HIGH-AVAILABILITY OF HADOOP DISTRIBUTED FILE SYSTEM (HDFS) ARCHITECTURE

Timothy MOSES
Department of Computer Science, Federal University of Lafia, Nasarawa State, Nigeria
visittim@yahoo.com

Keywords NameNode, HDFS architecture, Rack_Unit NameNode, metadata management, high-availability model, rack-aware.

Abstract: *Data-driven models like Hadoop have gained tremendous popularity in big data analytics. Though great efforts have been made through the implementation of the Hadoop framework by decoupling of resource management infrastructure, the centralized design of metadata management of HDFS has adversely affected Hadoop scalability and has resulted in a performance bottleneck. A single master node called NameNode which manages the entire namespace (all the inodes) of a file system has resulted in a single point of failure, namespace limitation, and load balancing issues in the Hadoop cluster. This paper proposed a rack-aware model where each rack is provided with a Rack_Unit NameNode (RU_NN) to manage namespace of file system and heartbeat communication of DataNodes in its rack. This will reduce load on a single NameNode and will also provide less communication overhead from all DataNodes in the cluster to a single NameNode.*

1. INTRODUCTION

A new crystal ball of the 21st century that helps put massive data together, classifying them according to their kind or nature is referred to as Big Data. Big data is a platform that helps in the storage, classification and analyzing massive volumes of data [1]. Hortonworks in [2] defined big data as a collection of large datasets that cannot be processed using traditional computing techniques. These data includes black box data (data from components of helicopter, airplanes, and jets), social media data such as Facebook and twitter, stock exchange data that holds information about the "buy" and "sell" decisions made on a share of different companies, power grid data like information consumed by a particular node with respect to a base station, transport data which includes model, capacity, distance and availability of a vehicle. These ever-increasing data pools obviously have a profound impact not only on hardware storage requirements and user applications but also on the file system design, implementation and the actual I/O performance and scalability behaviour of today's IT environment. To improve I/O performance and scalability therefore, the obvious answer is to provide a means such that users can read/write from/to multiple disks [3]. Today's huge and complex semi-structured or unstructured data are difficult to manage using traditional technologies like RDBMS hence, the introduction of HDFS and MapReduce framework in Hadoop.

Hadoop is an open-source Apache Software Foundation (ASF) project which is written in Java programming language that provides cost-effective and scalable infrastructure for distributed and parallel processing of large datasets across the commodity of clusters [4]. The programming paradigm was inspired by Google File System (GFS) [16] and Google’s MapReduce distributed computing environment. The idea was first conceived by Dough Cutting, an employee then with Yahoo and together with Professor Mike Caferalla of the University of Michigan, developed Hadoop later called Apache Hadoop [18]. Hadoop was named after Dough Cutting’s son toy elephant [17]. The framework was designed basically to provide reliable, shared storage and analysis infrastructure to the user community. Hadoop has two components – the Hadoop Distributed File System (HDFS) and the MapReduce framework [5]. The storage portion of the framework is provided by HDFS while the analysis functionality is presented by MapReduce [3]. The first generation Hadoop called Hadoop_v1 was an open source of MapReduce [19]. It has a centralised component called NameNode which is the file metadata server for HDFS that stores application data [4]. With Hadoop_v1, scalability beyond 4000 nodes was not possible with the centralized responsibility of JobTracker/TaskTracker architecture. To overcome this bottleneck and to promote this programming framework so that it carries other standard programming models and not just the implementation of MapReduce, the Apache Hadoop Community developed the next generation Hadoop called YARN (Yet Another Resource Negotiator). This newer version of Hadoop called YARN decouples resource management infrastructure from JobTracker in Hadoop_v1. Hadoop YARN introduced a centralized Resource Manager (RM) that monitors and allocates resources. Each application also delegates a centralized per-application master (AM) to schedule tasks to resource containers managed by Node Manager (NM) on each compute node [20]. The HDFS and its centralized metadata management remain the same on this newer programming model [20]. The HDFS is a master/slave architecture consisting of NameNode called master, a secondary node called checkpoint and several DataNodes called slaves [6]. The major/centralized controller that handles all file system operations is the NameNode hence; any request to the file system (like create, delete and read a file) must go through the NameNode. NameNode also handles block mappings of input files. Block creation, deletion, and replication are managed by the DataNode upon instruction from the NameNode [6]. A periodic heartbeat message is always sent from the DataNodes to NameNode (usually, default heartbeat is 3s) to be sure that there is no loss of connectivity between the two.

Hadoop has attracted the attention of engineers and researchers as a growing and very efficient framework for big data analytics. The controller in HDFS requires a single master node, called NameNode, which manages the entire namespace of a file system. This single and a centralized master node can cause Single Point of Failure (SPOF), namespace limitation and load balancing issues in Hadoop cluster. Though SPOF has been resolved by the introduction of Quorum Journal Manager (QJM) in Hadoop versions 2.1 and beyond, namespace limitation and load balancing issues still constitute a bottleneck for efficiency since all access requests to file systems have to contact the NameNode. While Hadoop 3.0 provides the option of running more than two namenodes in a cluster [21], only one namenode can be in an active state at a time. Though it is obvious that Hadoop HDFS is rack-aware [22], only a single (active) namenode exist in the whole cluster. This single namenode is a bottleneck for data block replication in the cluster. Several heartbeats communication to a single namenode is also a challenge. This paper, therefore, proposes a prototype rack-aware model for high-availability of HDFS for efficient file system operations in the Hadoop framework. Multiple namenodes are provided in the model where each namenode controls a rack in the cluster. The intention is to solve the issues of heartbeats communication and reduction in the time needed for data block replication in Hadoop cluster. The paper is organized into six sections. Section one is the introduction. Section two states the objectives of the study. While section three gives a description of several related works proposed to provide efficiency in HDFS, section four gives a description of Hadoop cluster and the network. Section five describes our proposed rack-aware model for high-availability of HDFS in Hadoop framework. Section five is the conclusion and research gap in the study.

2. OBJECTIVES OF THE STUDY

The objectives of this study are;

i. To provide NameNode for each rack in the Hadoop cluster that will handle all file system operations for corresponding DataNodes in the same rack.
ii. Partition entire namespace of a file system into corresponding number of racks available in the cluster.
iii. Ensure improved availability of NameNode for heartbeat communication and load

balancing with corresponding DataNodes in its rack.

3. RELATED WORK

In the present world, every single second holds the age of the colossal measure of information [7]. The storage of such enormous and valuable information is one of the most essential assignments, and furthermore, the viable analysis of such a gigantic measure of data to get satisfactory outcome is significant. Such a tremendous measure of data may prompt utilization of an immense measure of time that requires an efficient big data analytics framework to process. To reduce time utilization and increase performance therefore, efforts have been geared towards developing a high-availability HDFS for the Hadoop framework. Azzedin in [8] proposed a framework which decreases the reliance on the size of metadata on the NameNode. The metadata for both storage allocation and replication of data is stored in the RAM of the NameNode. The author [8] argued that storing the metadata of an immense measure of data in a single NameNode brings about load balancing issues and efficiency bottleneck. Along these lines, the design builds up itself appropriately on overseeing huge metadata by selecting itself into a Chord protocol-based architecture that interfaces with the HDFS to provide scalability to the Hadoop framework [8]. As the framework utilizes a chord protocol, it impacts the framework by expanding intricacy of single HDFS NameNode architecture. Though [8] architecture ensured that there is no over-reliance on NameNode through chord protocol architecture, the NameNode still serves as the master node in HDFS hence, efficiency bottleneck through heartbeat communication with DataNodes in the cluster still persist.

The authors in [9] developed NCluster for high-availability of HDFS. The architecture used multiple active NameNodes instead of one NameNode in the cluster [9]. To handle metadata replication process across these active nodes, pub/sub system was used. This approach to providing high-availability helped NCluster architecture in exhibiting effectiveness in the Hadoop cluster. How DataNodes failure is handled in NCluster however, still remains a major concern. Kim *et al.*, in [10] suggested a distributed and cooperative NameNode cluster for highly-available HDFS. The architecture consists of several NameNodes to resolve the issue of SPOF, namespace limitation and load balancing problem. The framework was designed in a way that only one primary NameNode exists, others are backup NameNodes [10]. The entire namespace of a file system was partitioned into several fragments, and replicas of each fragment dispersed among NameNodes in the cluster. Performance bottleneck caused by a single NameNode can then be resolved by assigning different NameNodes to different fragments as the primary ones [10]. Fig. 1 and 2 gives a description of namespace partitioning in Hadoop 2.0 and [10] architecture respectively.

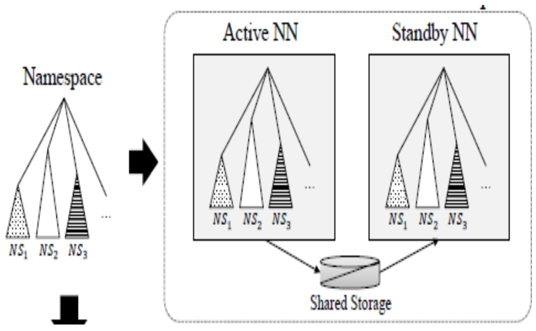

Fig. 1: Namespace partitioning in Hadoop 2.0 [10]

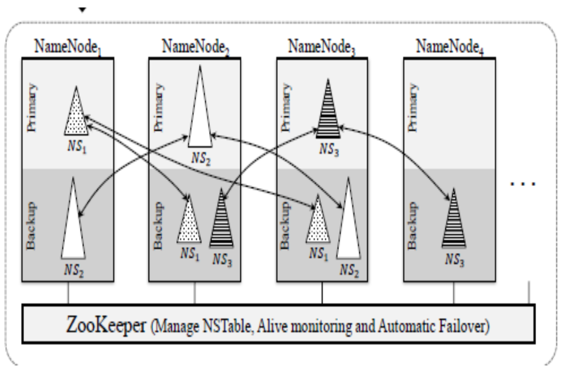

Figure 2: Namespace partitioning in [10]

Though Hadoop versions 2.0 and beyond replicates complete namespace using shared storage as shown in Fig. 1, the architecture in [10] however, ensure that replicas of each fragment are dispersed among NameNodes; with only one of the NameNodes serving as primary NameNode while others are backup NameNodes. The architecture uses Zookeeper to manage the hash table called NSTable [10]. With the entire namespace partitioned into several fragments and replicas of each fragment dispersed among NameNodes, the architecture will be able to tolerate up to (k/2 – 1) faulty NameNodes in k replicas. Handling of heartbeat communication of DataNodes in the cluster by this multiple NameNodes however, is a bottleneck since the architecture has only one primary NameNode with others serving as backup.

Islam in [11] proposed a hybrid design called Triple-H that guarantees effective data placement policies to accelerate HDFS on HPC Clusters. The significant thought behind the hybrid plan was the consideration of high-performance hardware to the HDFS. The author [11] opined that, since Hadoop architecture keeps up the total metadata into the essential memory of the NameNode, it will be essential if the NameNode is supplanted with high-execution hardware to solve issues of I/O bottlenecks [11]. The expansion of such high-execution hardware has made the framework more optimized and scalable. However, scaling the NameNode deep does not change the issue of centralization as obtained in HDFS architecture. High-performance hardware for NameNode will still reach a threshold limit that will downgrade performance in the Hadoop cluster. Stamatakis *et al.*, in [12] proposed a general-purpose architecture for replicated metadata services in distributed file systems. The motivation for the work came from the disadvantages of Parallel Virtual File System (PVFS) and HDFS architectures. The authors in [12] argued that PVFS provides stateless replication of data blocks of a file on defined metadata servers in a shared network accessible storage, which, however, suffers a single point of failure. HDFS, which provides quorum based replication, puts a limit to store metadata on main memory due to its checkpoint and roll forward solution [12]. This limitation prompted the development of the Replicated Metadata Service (RMS). RMS uses a transactional key-value data store to provide sufficient support for the type of operations in file system metadata and also provides better scalability in a distributed setting. The system was implemented on HDFS, calling the resulting system HDFS-RMS [12]. HDFS-RMS still remains a single master node for file system operations. The architecture did not resolve the issue of several heartbeats protocol coming from all DataNodes in the Hadoop cluster.

A high-availability HDFS architecture based on threshold limit and saturation limit of the NameNode was proposed by [7]. The architecture agreed with having multiple NameNodes in a cluster. The architecture was designed in a way that all NameNodes have the same number of DataNodes. The number of DataNodes connected to a NameNode is dependent on the threshold limit and saturation limit of the NameNode [7]. The threshold limit is regarded as the maximum number of DataNodes that a NameNode can connect to for optimal performance and load balancing. The saturation limit has to do with additional load (DataNodes) a NameNode can connect to once all NameNodes have reached their threshold limit [7]. Fig. 3 describes this architecture.

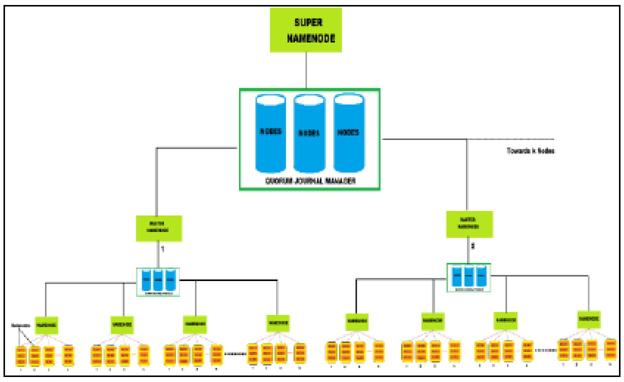

Fig. 3: Double layered Namenode Management [7]

The architecture is such that all DataNodes are connected to the NameNode with a super NameNode connected to all other NameNodes (Fig. 3). Once a DataNode gets disconnected from a particular NameNode, another DataNode can be connected as shown in Fig. 4.

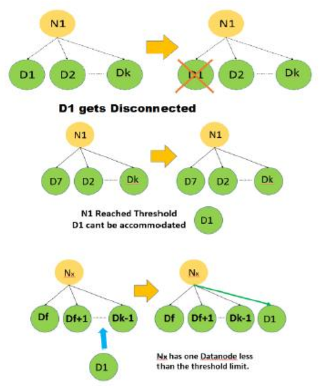

Fig. 4: Interpretation of connectivity in [7]

The DataNodes and NameNodes are represented by D and N respectively, with each having a unique identifier. If for instance, D_1 gets disconnected, and D_7 arrives, then D_7 will be connected to N_1 except if N_1 has reached its threshold limit. If D_1 arrives later but all NameNodes have reached their threshold limits, then D_1 can be accepted for connectivity by N_1. Though this architecture is efficient to solve issues of load balancing and namespace limitation, NameNodes in the cluster are not rack-aware hence, reconnecting DataNodes may lead to conflict in the Hadoop cluster. Load balancing between NameNodes in HDFS architecture was presented by [13]. The architecture also used multiple NameNodes to resolve the issue of SPOF. The NameNodes are connected to each other with their respective I/O operations. Anytime a client sends a request to a NameNode, the NameNode checks the entry of the request in the namespace. Once an entry exists, the client is notified and appropriate DataNodes contacted. If however, there is no entry in the namespace, the client is contacted. The architecture is similar to [7] and [10]. No provision for heartbeat communication between NameNodes and DataNodes. Load balancing issues may occur because no primary/super NameNodes is available in the architecture.

Bakshi in [14] and Datafliar in [23] proposed another design for keeping up metadata adequately and productively by incorporating the framework with two balanced nodes and Quorum-based third party node [14], [23]. As the NameNode in Hadoop is helpless to Single-point-Failure, [14] and [23] proposed two even NameNodes where one of the nodes is in a functioning state and the other one in the passive state as shown in Fig. 5. The active node stays as the essential NameNode when it is performing a task. If by any chance the active node fails, it is replaced by the passive node [14], [23]. Here the simultaneousness between the substances of the NameNode is kept up by the Quorum Journal Nodes (QJN). The QJM holds the total edit logs of the Namenode, and the passive node is capable of reading all the edit logs and can make changes to its individual namespace from the QJM. QJM in this architecture provides for simple concurrency control in keeping up the metadata over the Namenodes and furthermore avoids the single-point-failure issue. The architecture solely resolve the issue of SPOF in the Hadoop cluster. This development has been implemented in Hadoop versions 2.1 and beyond. Balancing load on several DataNodes across the Hadoop cluster with a single active NameNode will still lead to a performance bottleneck.

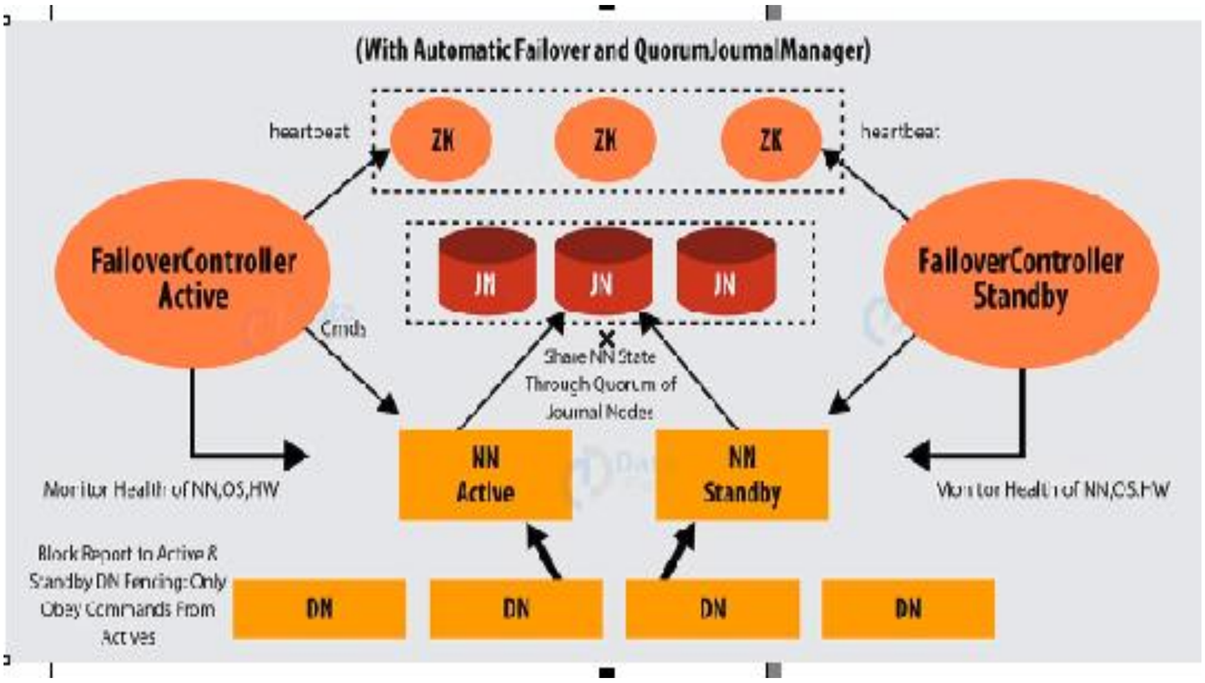
(With Automatic Failover and QuorumJournalManager)
heartbeat
ZK
ZK
ZK
heartbeat
FailoverController Active
JN
JN
JN
FailoverController Standby
Cmds
Share NN State Through Quorum of Journal Nodes
NN Active
NN Standby
Monitor Health of NN,OS,HW
Monitor Health of NN,OS,HW
Block Report to Active & Standby DN Fencing: Only Obey Commands From Actives
DN
DN
DN
DN

Fig. 5: HDFS namenode high availability architecture [23].

4. DESCRIPTION OF TYPICAL HADOOP CLUSTER AND THE NETWORK LAYOUT

The three major layers of machine roles in any Hadoop deployment are the client machines, master nodes and the slave nodes [15]. The master nodes are responsible for overseeing two key functional processes that make up the Hadoop framework; HDFS that stores massive data and MapReduce responsible for running parallel computation [15]. The slave node layer makes up the vast majority of workstations that store and also process data. Client machine has Hadoop installed with all the cluster settings so as to enable for loading of data into the cluster, submission of MapReduce/other applications describing how the applications should retrieve and process data and how it should view results once a task is completed. Fig. 6 shows typical server roles with these three major layers.

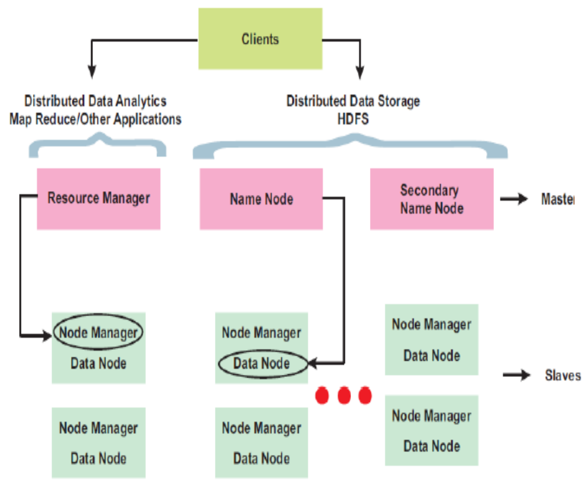

Fig. 6: Hadoop Server Roles [15]

Typical architecture of the Hadoop cluster has rack servers populated in racks connected to a top of a rack switch [15]. The rack switch has uplinks which are also connected to another tier of switches, which connects all other racks with uniform bandwidth to form a cluster (see Fig. 7).

Hadoop has the concept of "rack awareness". There are two reasons for setting rack awareness when storing data in HDFS; data loss prevention and network performance [15]. Since data are replicated to avoid losing all copies of data, it is expected that while doing this, all data are not replicated at different nodes on the same local rack. If this happened and the rack experiences a failure such as a switch or power failure, then that data will be lost. It is also believed that two machines in the same rack have more bandwidth and lower latency between each other than two machines in two separate racks. This is true because rack switch uplink bandwidth is usually less than its downlink bandwidth. Also, in-rack latency is lower than cross-rack latency. Hence, network performance can be enhanced if the framework is rack aware.
The NameNode of the HDFS in a cluster holds all the file system metadata for the cluster. It oversees the healthy state of each data node in the cluster and coordinates access to them [6]. It keeps track of the cluster storage capacity making sure that each block of data meets its minimum defined replica policy. Name Node is the central controller of HDFS. It does not hold cluster data itself but knows what blocks make up a file and where these blocks are located in the cluster [15].

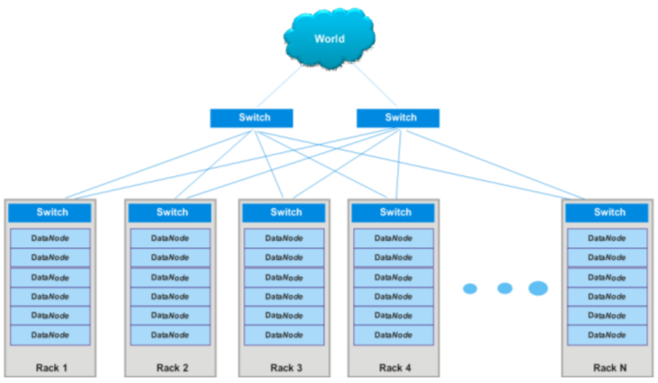

Figure 7: Hadoop Cluster [15]

Anytime the client wants to read data, the NameNode points the client to the DataNodes they need to talk to. DataNodes send heartbeats to NameNode at an interval of 3seconds through a TCP handshake using the same port number that defines the NameNode daemon. Every tenth heartbeat of DataNode to NameNode is a block report that tells NameNode about the blocks it has [15]. This report makes NameNode build its metadata and ensure that three copies of each block of data exist on different nodes in different racks [15]. NameNode forms a crucial component of HDFS without which the client will be unable to read/write to HDFS and it will be difficult to schedule map-reduce jobs or other applications on the Hadoop framework. Anytime heartbeat communication stops between

NameNode and DataNode, it is presumed that such DataNode is dead and any data its holding gone as well. Previous block reports received from the said DataNode will help the NameNode to know which copies of blocks died along with the node. Using rack aware policy, the NameNode will re-replicate those blocks on other DataNodes. The limitation with this, however, is when an entire rack of servers falls off the network due to rack switch failure or power failure. It then means that the NameNode will instruct the remaining nodes in the cluster to re-replicate all the data blocks lost in the rack. This process may mean that hundreds of terabytes of data will need to begin traversing the network.

To guard against failure of NameNode, Hadoop has a sever called the Secondary NameNode. There is a common misconception however about the responsibility of Secondary NameNode in Hadoop. Many think that its role is to provide availability backup for NameNode but it is not the case. The Secondary NameNode occasionally connects to the NameNode (by default, every one hour) to fetch a copy of NameNode in-memory metadata and files used to store metadata, which sometimes both daemons may be out of sync.

If perhaps the NameNode dies, the copy retained by Secondary NameNode can be used to recover the NameNode but may not be the exact copy of what the NameNode holds before failure. However, Quorum Journal Manager has been provided in Hadoop 2.0 and beyond to help guard against SPOF provided both the NameNode and Secondary NameNode are in sync. The issue of heartbeat communication and balancing of loads still persist. Disconnection of NameNode from DataNodes due to rack switch failure or power failure is also a bottleneck since this will mean, Namenode replicating all data blocks of whole DataNodes in the affected rack to other DataNodes in another rack. This issues are what this proposed system intends to solve.

5. PROPOSED RACK-AWARE MODEL FOR HDFS ARCHITECTURE

This work intends to solve the problem of high-availability of NameNode in the existing HDFS architecture by allowing two NameNodes in the upper layer of the proposed architecture as shown in Fig. 8. Just as it is obtained in the existing system [14][23], one of the two NameNodes will be active while the second will be passive.

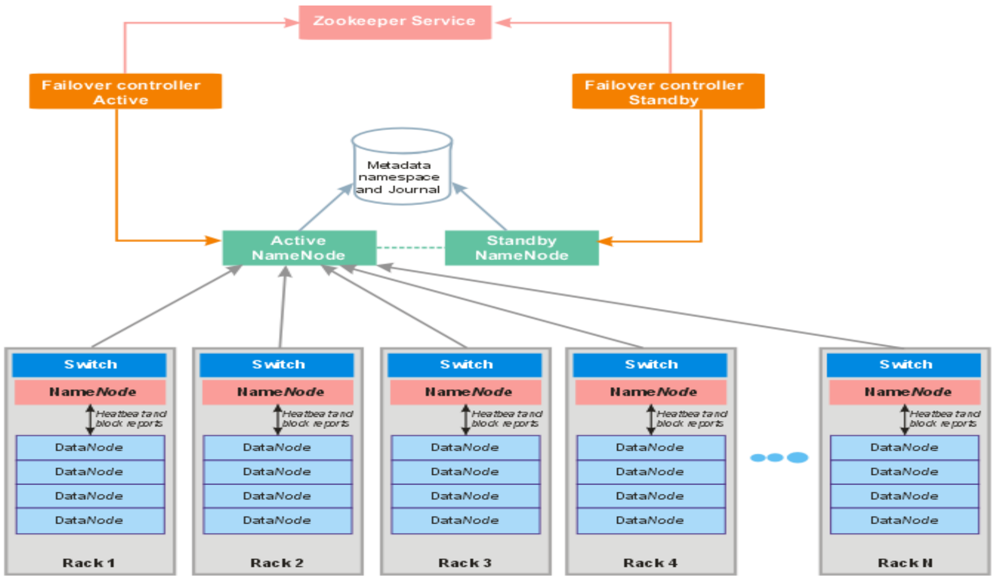
Zookeeper Service
Failover controller Active
Failover controller Standby
Metadata namespace and Journal
Active NameNode
Standby NameNode
Switch
NameNode
Heatbeat and block reports
DataNode
Rack 1
Rack 2
Rack 3
Rack 4
Rack N

Fig. 8: Proposed rack-aware model for high-availability HDFS architecture

For high-availability, Zookeeper service; as available in the existing architecture will also be used. The Zookeeper has failover controllers for both active and standby NameNodes. To ensure that the two NameNodes called Super NameNodes are in sync, a shared storage device is also provided. Active NameNode logs record of modification done in its namespace is transferred to EditLog in the shared storage. The standby NameNode reads EditLogs in shared storage and applies it to its namespace.

If active NameNode fails, the Zookeeper does the following:

i. The Zookeeper controller for active NameNode fences the node using a technique called "shoot the other node in the head" (STONITH). This is a special technique in HDFS architecture that forcibly power down the NameNode machine.

ii. The Zookeeper controller of the standby NameNode opens a connection for it to become active.

iii. The standby NameNode updates its metadata information using the EditLogs in the shared storage.

iv. The standby NameNode resumes as active NameNode until the latter recovers from failure.

The novel approach in this architecture is the introduction of NameNodes in each rack called Rack Unit NameNode (RU_NN). Each of RU_NN will be responsible for monitoring the health status of DataNodes in its rack. A heartbeat communication exists between the active NameNode in the upper layer of the architecture and each RU_NN only to monitor the health status of the RU_NN not the entire DataNodes in the cluster.

This development will help reduce communication overhead from all DataNodes to a single NameNode. Though heartbeat communication also exists between RU_NN and its corresponding DataNodes in the rack, the communication mode is in-rack. In-rack communication latency is always lower than cross-rack latency since uplink bandwidth is usually less than downlink bandwidth.

Each RU_NN log records of modification done in its namespace are forwarded to the active NameNode which in turn updates the shared storage. RU_NN also reports failed DataNodes in its rack. This will enable the active NameNode in the upper layer of the architecture to have a global view of active DataNodes in the cluster.

In the case of rack failure due to either switch or power failure, active NameNode need not replicate all data blocks in the failed rack. Rather, data blocks available in other racks are used until such rack recovers from failure. It is however almost difficult to have rack failure like it is with DataNode failures.

5.1 Major distinction between the prototype model and existing models

Table 1.1 gives a summary of the architectural design of existing HDFS namenode high-availability models and the proposed prototype model.

Table 1.1 Distinction of the prototype model from exiting models

Author	Implementation details	Active NameNode(s)		Distinction from prototype model
		Single	Multiple	
[7]	Multiple NNs with each having the same number of datanodes are connected to a super NN for effective load balancing		√	Recovered DataNodes are recomected to any available NN (base on threshold lmit of NNs). This will be a bottleneck to rack-aware policy in Hadoop cluster
[8]	A chord protocol-based achitecture that interface with HDFS to provide scalability for Hadoop cluster	√		Single active NN will can lead to efficiency bottleneck

[9]	Designed to provide multiple active NN instead of one NN to handle metadata replication process across the active NN		√	How Ncluster handles datanode failure in the cluster still remains a major concern.
[10]	The architecture provides a distributed and cooperative NN cluster for highly-available HDFS. It has one primary NN with multiple NNs serving as backup.		√	Handling heartbeats communication of datanodes by one primary NN is a concern since other NNs only serve as backup.
[11]	A hybrid design that guarantees effective data placement to accelerate HDFS on HPC cluster. The design ensures that the NN is supplanted with high-execution hardware to solve issues of I/O bottleneck	√		Scaling the NN deep does not change the issue of centralization as obtained in existing models.
[12]	Developed HDFS-RMS to provide sufficient support for the type of operations in file system metadata.	√		Still remains a single mater node for file system operations. This may lead to performance bottleneck.
[13]	The design ued load balancing technique for data placement among multiple NNs in Hadoop cluster		√	Heartbeat communication between NN and datanotes is still a major concern.
[14][23]	Same architectural design with Hadoop 3.0 and beyond. The design has an active and standby NN with several Quorum Journal Nodes to provide edit logs which is shared between these two NNs.	√		Because only a single namenode is active at a time, several heartbeats to this single (active) namenode will still lead to load balancing and datablock replication bottleneck
Prototype model	Provides multiple NNs where each rack is provided with one NN called RU_NN to manage namespace file system and heartbeat communication of datanodes in its rack.		√	Since each NN handles namespace file system, heartbeat communication and health status of each datanodes in its rack, communication overhead is reduced. Performane bottleneck associated with I/O operation in the cluster is also reduced.

6. CONCLUSIONS

The proposed architecture is intended to provide continuous replication maintenance through active and rack-unit NameNodes, end-to-end and periodic check of super NameNodes, RU_NN, and DataNodes in the cluster. Heartbeat communication between several DataNodes to a single NameNode in the existing system is resolved by the provision of RU_NN to oversee the health status of DataNodes in their respective racks. This will reduce communication overhead and job delays caused by a single NameNode in the existing HDFS architecture. Namespace limitation is resolved through shared storage, and load balancing issue is taking care of through global view of all active DataNodes in the cluster. SPOF is resolved by the Zookeeper failover controller. This proposed architecture, promises to resolve the performance bottleneck in the existing HDFS architecture.

7. REFERENCES

[1]. Camille, R. “Big Data open platforms”. Project final report for Department of Information and Systems Engineering, Polytechnic Institute of Coimbra, 2015. Accessed 24.06.2019: https://slidex.tips/download/big-data-open-platforms

[2]. Hortonworks, “Apache Hadoop YARN”, 2016. Accessed 18.10.2019: https://hortonworks.com/apache/yarn

[3]. Dominique, A. H. “Hadoop design, architecture and MapReduce performance”. *DH Technologies*, 2015. Accessed 10.03.2019: www.dhtusa.com

[4]. Shvachko, K.; Kuang, H.; Radia, S. and Chansler, R. “The hadoop distributed file system”. *2010 IEEE 26th symposium on Mass Storage Systems and Technologies (MSST)*, Washington D. C., 2010, USA: IEEE Computer Society.

[5]. Nagina, D. and Sunita, D. “Scheduling algorithm in big data: A Survey". *International Journal of Engineering and Computer Science*, 5(8), 17737-17743, 2016.

[6]. Ibrahim, A. H. T.; Nor, B. A.; Abdullah, G.; Ibrar, Y.; Feng, X. and Samee, U. K. "MapReduce". *Review and Challenges. Springer Journal*, 109(1), 389-421, 2016.

[7]. Chakraborty, S.; Barua, K.; Pandey, M. and Rautaray, S. "Architecture based on threshold limit and saturation limit of the NameNode". *I. J. Information Engineering and Electronic Business*, 6, 27-34, 2017.

[8]. Azzedin, F. "Towards a scalable HDFS architecture". *International Conference on Collaboration Technologies and Systems (CTS)*, IEEE, 2013.

[9]. Wang, Z. and Wang, D. "NCluster: Using multiple active NameNodes to achieve high availability for HDFS". *10th International Conference on High Performance Computing and Communication*, IEEE, 2013.

[10]. Kim, Y.; Araragi, T.; Nakamura, J. and Masuzawa, T. "A Distributed NameNode Cluster for a Highly-Available Hadoop Distributed File System". *33rd International Symposium on Reliable Distributed Systems (SRDS)*, IEEE, 2015.

[11]. Islam, N. S. "Triple-H: a hybrid approach to accelerate HDFS on HPC clusters with heterogeneous storage architecture". *15th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, IEEE/ACM China, 2015.

[12]. Stamatakis, D.; Tsikoudis, N.; Micheli, E. and Magoutis, K. "A General-Purpose Architecture for Replicated Metadata Services in Distributed File Systems". *Transactions on Parallel and Distributed Systems*, IEEE, 2017.

[13]. Jakkal, M.; Goli, S.; Dudam, A.; Nilgar, P. and Khan, A. "Performing load balancing between NameNodes in HDFS". *International Research Journal of Engineering and Technology*, 6(3), 1838-1840, 2019.

[14]. Bakshi, A. "How to set up Hadoop cluster with HDFS high-availability", 2019. Accessed 10.02.2020: https://www.edureka.co/blog/how-to-set-up-hadoop-cluster-with-hdfs-high-availability/

[15]. Brad, H. "Understanding Hadoop clusters and the network", 2011. Accessed 23.12.2019: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

[16]. Ghemawat, S.; Gobioff, H. and Leung, A. T. "The Google file system". *Paper presented at the ACM SIGOPS operating systems review*, New York City, 2003, NY: ACM.

[17]. White, T. "Hadoop: The definitive guide", O'Reilly Media, Sebastopol, CA, 2009.

[18]. Shouvik, B. and Daniel, A. M. "The anatomy of MapReduce jobs, scheduling and performance challenges". *Proceedings of the 2013 conference of the Computer Measurement Group*: Semantic Scholar, San Diego, CA, 2013.

[19]. Bialecki, A.; Cafarella, M.; Cutting, D. and O'Malley, O. "Hadoop: A framework for running applications on large clusters built of commodity hardware", 2005. Accessed 06.06.2019: http://lucene.apache.org/hadoop/

[20]. Wang, K. "Scalable Resource Management System Software for Extreme-Scale Distributed Systems", PhD Dissertation, 2015. Accessed 16.07.2019: http://datasys.cs.iit.edu/publications/2015_IIT_PhD-thesis_Ke-Wang.pdf

[21]. Apache Hadoop, "HDFS high availability using the Quorum Journal Manager", accessed 15.07.2020: https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

[22]. Alapati, S. R. "Expert Hadoop Administration: Managing, tuning and securing Spark, YARN, and HDFS", Pearson Education Inc, USA, 2017.

[23]. Dataflair, "NameNode high availability in Hadoop HDFS", accessed 15.07.2020: https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/