2018 International Conference on Advances in Computing and Communication Engineering (ICACCE-2018)
Paris, France 22-23 June 2018

A New Software Architecture Style for Hadoop Systems

Fatima El Jamiy
Computer Science Department
University of North Dakota
Grand Forks, North Dakota, USA
fatima.eljamiy@ndus.edu

Hassan Reza
Computer Science Department
University of North Dakota
Grand Forks, North Dakota, USA
reza@cs.und.edu

AbdElRahman ElSaid
Computer Science Department
University of North Dakota
Grand Forks, North Dakota, USA
abdelrahman.elsaid@und.edu

***Abstract*— Big data provides a challenging environment to store, process and analyze large scale data. Various requirements that impact the architectural solution of big data include the source of data and its involved features such as volume, velocity, variety and the type of data. While processing big data, other challenges raised related to scalability, availability, integrity, concurrency, parallelism and performance. Because of all these features and requirements, building a suitable big data solution needs to consider the architectural solutions to satisfy those different elements of the system. Different software architectural styles exist. The aim of this paper is to provide an overview of the important commonly used architecture styles for building big data systems, compare their benefits, performance and main components. And mostly cover how each style can help in resolving the challenges required by developing big data software systems. The goal is to identify and discuss software architectural issues imposed by Hadoop that impact its performance. A new hybrid architecture system to achieve the requirement for Hadoop is proposed and all possible challenges the Hybrid system could face are investigated.**

Keywords—Distributed systems; Hadoop; Software architecture styles; HDFS

I. INTRODUCTION

Apache Hadoop (high-availability distributed object-oriented platform) is a distributed system that offers a distributed storage system via its HDFS file system (Hadoop Distributed File System) and provides a data analysis system called MapReduce that uses the HDFS file system to perform processing on large volumes of data. In this work we will be focusing on improving the architecture of HDFS to improve the performance of Hadoop system.

MapReduce is a parallel computing paradigm for large scale data systems [1]. It distributes data across a set of computing nodes. In MapReduce, each node processes the blocks of data stored on it and does not communicate with other nodes without shared state, but their data flow is dependent where the output of a node (Mapper) is the input of another (Reducer). MapReduce can change the system behavior by connecting or removing machines and thus slowing down and speeding up the computation. However, the main issue about the performance of the system appears when we demystify the architecture of Hadoop in which only the NameNode manages the whole system and the various features of the system are challenging to handle and consider while designing the architecture style.

The paper is exploring the different architecture styles and compare them to come up with a hybrid architecture style for big data system that will combine different features from each candidate architecture style in order to cover the different functional and non-functional requirements of large scale systems.

This paper is mainly a survey on software architecture styles for Hadoop and it is organized as follows: In Section 2, a discussion of Hadoop system is given, we present the features of this system, and present some detailed design and architecture issues. MapReduce architecture style is presented in section 3. The main approach with the new hybrid architectural style for Hadoop and the related different challenges related to the new system are discussed in section 4. Finally, we conclude the paper in section 5.

II. BACKGROUND

We are in the era of production of massive data (Big Data) in which a definition involves five dimensions (5Vs): Volume, Velocity, Variety (frequency), Veracity and Value. The data sources are numerous. On the one hand, the applications generate data from logs, sensor networks, transaction reports, traces of GPS, etc. And on the other hand, users produce data such as photographs, videos, music or data on the health status (heart rate, pressure or weight). A problem then arises as to the storage and analysis of data. The storage capacity of hard drives increases but reading time is also growing. It then becomes necessary to parallelize the processing by storing on multiple hard disk drives. However, this raises the hard disk reliability problem that generates hardware failure [2]. The proposed solution is the duplication of data like a RAID 6 system.

Apache Hadoop (High-availability distributed object-oriented platform) is a distributed system that addresses these issues. On the one hand, it offers a storage system via its HDFS distributed file system (Hadoop Distributed File System) and it offers the possibility of storing the data by duplicating it, a Hadoop cluster therefore does not need to be configured with a RAID system that becomes useless [1]. On the other hand, Hadoop provides a data analysis system called MapReduce. It works with the HDFS file system to perform processing on large data volumes.

978-1-5386-4485-0/18/$31.00 ©2018 IEEE

Hadoop was created by Doug Cutting to the needs of Apache Nutch project, an open source search engine [1]. It is important to note that Doug Cutting also created the Apache Lucene text search library. When the Apache Nutch project started in 2002, contributors have understood that the original architecture could not hold scalability on more than 20 billion pages from the Web. Google published in 2003 a paper about the architecture of its distributed file system GFS (Google's distributed filesystem). Google then published in 2004 a paper introducing the MapReduce system for analyzing data of a GFS system. Doug Cutting decided to take the concepts presented by the two items to solve problems in the Apache Nutch project. In 2006, Hadoop was a subproject of Apache Lucene and in 2008, an independent project of the Apache Foundation [1,11].

Apache HDFS is used for its storage capacity (about one terabyte per day) and its ease of scalability at lower cost.

III. RELATED WORK

Distinct recent models have been proposed for Hadoop to solve the failure point bottleneck. Majority of these implementations focus on using one of the most popular methods to manage metadata in a distributed environment. Hashing, Sub tree partitioning and consistent hashing are among the techniques proposed and which are the bases of the architecture of the most effective high-performance file system.

In [4], authors proposed a distributed model that deploy a hashing technique to split mainly the flattened namespace of Hadoop Distributed File System. The solution has been integrated to Hadoop and evaluated with a distributed NameNode. According to the results, the prototype has not been fully integrated into Hadoop ecosystem or tested with a real-world application. This means that scalability of the system has not been demonstrated and its capability to take over when failure occurs has not been assessed either. Besides, the load imbalance technique deployed by Hadoop to locate the data in the DataNodes has not been taken into consideration, which makes this method not suitable for Hadoop environment.

Over and above, in Hashing (Lustre, zFs file system), hierarchical directory structure is employed for metadata allocation that can create an enormous overhead and therefore the performance of the system will decrease. In fact, if a new name is given to the path, the location of the directory will be unable to find it and in that case, a movement of metadata is needed [5].

To avoid the issue addressed in the first related model, a mechanism based on both hashing and sub tree partitioning techniques was proposed in [10]. It is a distributed metadata approach for multi-NameNodes that implement these two techniques by using a two-level algorithm.

Placement of data between nodes generate large amount of data and make the system less scalable. So, they suggest that the integration of hash algorithm in the first level will solve the problem by setting the directory path name as an argument [9]. This phase is performed by the client. Dispatching the characteristics of the directory's files at the first level will enhance the rendering of its operations. In the second level, a dynamic algorithm to select the position of blocks depending on the actual load has been chosen. The NameNodes send their loads with all the other related information about their status to the master NameNode that keep a record of them and then use them in the second level to classify and scatter the position data to the NameNode with fewer loads.

A sub-tree technique was introduced to determinate the value of the load and improve it. According to the results, the performance of this two-level algorithm has been enhanced. However, this approach is still not exact and precise because the Sub Tree (in ceph and coda HPC file systems) presumes the computation of the load which changes dynamically, and a bottleneck in the network will be produced by the migration of metadata.

The latest interesting work [6] has mentioned the consistent hashing as a technique to overcome the drawbacks of the previous techniques. The proposed method has been compared to the above other techniques and has showed a good performance. The metadata is split into blocks and distributed relating to the loads to the different multiple NameNodes and log replication was used to assure consistency.

Consistent hashing is found in Amazon Dynamo. The return function is like a range in which each node is in charge of the data between it and the previous node and at the same time a virtual node is allocated for it [11]. This way, data and loads are shared and dispatched regularly among nodes. The access to files and the association of files to blocks and the namespace of the file system are managed by the master NameNode.

In this approach [8], the nodes send heartbeats so the master node can discover a failure and recover it. After that, the blocks are assigned to another NameNode. Adding or deleting NameNodes does not need to restart the system and the redistribution of metadata is performed right away. However, this approach still represents some limitations in terms of scalability and availability.

IV. MAPREDUCE AS ARCHITECTURE STYLE

MapReduce is an internal implementation proposed by Google to process big amounts of data across multiple nodes. It is a fault tolerance system with write once and read many. MapReduce style is composed of two different styles, Master-slave style and batch sequential style [6].
The master node controls execution and manages replicated file system. The slave nodes execute the mappers and reducers functions and contain replicated data blocks. The execution environment is taking care of :

- The planning of each job by dividing it into tasks
- Placement of data and code where each node contain its data locally
- The synchronization is assured by making reduce tasks wait for map tasks
- High tolerance to node failures

HDFS, the distributed file system is used by Hadoop along with MapReduce. It is a master/slave architecture. Two types of nodes are defined, the Master NameNode and data servers DataNodes. Files are divided into chunks and the blocks are replicated across the DataNodes. The NameNode knows the location of each block but the clients communicate directly with DataNodes.

As mentioned before MapReduce is not a tool as some can confused it with. It is a paradigm and a framework that you must match your system into it of Map and Reduce Modules, and that might be a challenging task. MapReduce is a constraint more than a feature [7].

This design and constraint make problem solving easier and harder at the same time because it limits the possible options that you can have and gives you a limit range of design choices, which will narrow what it can and cannot be done. So, more thinking and work on algorithm will be needed in order to be able to solve and develop the system with those constraints [8].

1) Components: MapReduce system consists of a single master worker component and multiple map worker components. A worker controller is used by the master worker component to communicate with others. Data is written by Map workers components to their local file system with the help of a local file system connector, and Reduce worker's components read data with the same way. The two components Map and Reduce use a global filesystem connector.

2) Constraints: while writing the program, two functions are developed, a map function and a reduce function. Creating equal chunk sizes is important in the whole process. If splitting the original input set is not done efficiently, that will slower the overall system and decrease the performance of the system because some map workers will take longer to run than others [9]. The parallelized computation is basically the same as the sequential computation if we use deterministic map and reduce functions.

3) Qualities:

a) Scalability: It is the primary quality attribute that MapReduce improves and it is its key goal. Dividing task into small tasks and distribute them across many nodes improves the performance than executing the task sequentially, especially if the task is not easy to manage sequentially. Programs written following the MapReduce style can run on one cluster as they can run on more than one cluster of thousands of machines.

b) Availability: MapReduce fosters availability. If a failure occurs the system recover from it by rescheduling the task on another node close to the one that caused the failure.

c) Data locality: It impacts strongly the performance of the MapReduce architecture style. To mitigate the bandwidth use, intermediate outputs should be kept close to the map and reduce worker components. The global file system used is often a distributed and redundant file system.

4) MapReduce and Pipe filter:

MapReduce is a very strong paradigm for parallel and distributed computing and processing. Most real problems in the real world cannot be solved by a single MapReduce job. And that is why a set of MapReduce jobs are linked together so that the input to one job is the output of a previous job. If not handled and architected adequately this can become a development and maintenance issue.

MapReduce system is often a combination of MapReduce architecture style and the batch sequential style, in which the output from one MapReduce job is the input for the next. Each MapReduce job is a phase in the batch sequential network. Mixing these two architectural styles can make a problem that was not adequate for MapReduce into one that is.

MapReduce is considered as an architectural style and Hadoop is an open source implementation of MapReduce that implement that architecture style.

V. APPROACH: HYBRID SOFTWARE ARCHITECTURE FOR HADOOP

A. Overview

The fundamental concept of Hadoop is the manipulation, processing and analysis of very large datasets (which are in petabytes, Po), which are then automatically distributed in storage spaces and batches of processing on a set of low cost server clusters. From one server to several thousands of machines, Hadoop is a scalable solution with a capacity for fault tolerance. Failure detection and automation give Hadoop an excellent resistance. Behind Hadoop, there are two important technologies: MapReduce and HDFS, Hadoop's file system. MapReduce is the infrastructure that identifies and assigns batches processing to nodes in a Hadoop cluster.

MapReduce executes these batches in parallel mode enabling their processing and analysis to carry large amounts of data in a short time. HDFS, for its part, gathers and connects all the nodes of the same Hadoop cluster into a single large file system. As there is a failure, HDFS guarantees reliability by replicating data at the level of several nodes. High availability, reliability and fault tolerance is achieved by using replication.

It is important to note that Hadoop can use any distributed file system but that will come with a cost because Hadoop will not benefit from data locality provided by HDFS. Hadoop needs to know where data is and what are the nodes that are close to the data. HDFS is a distributed file system developed specifically for Hadoop to provide locality, fault tolerance, reliability and mainly to be integrated with Hadoop and its architectural requirements.

Hadoop's performance is limited by the single point of failure, the NameNode and the operations allowed on it, one writer at a time, no overwrites and no appends.

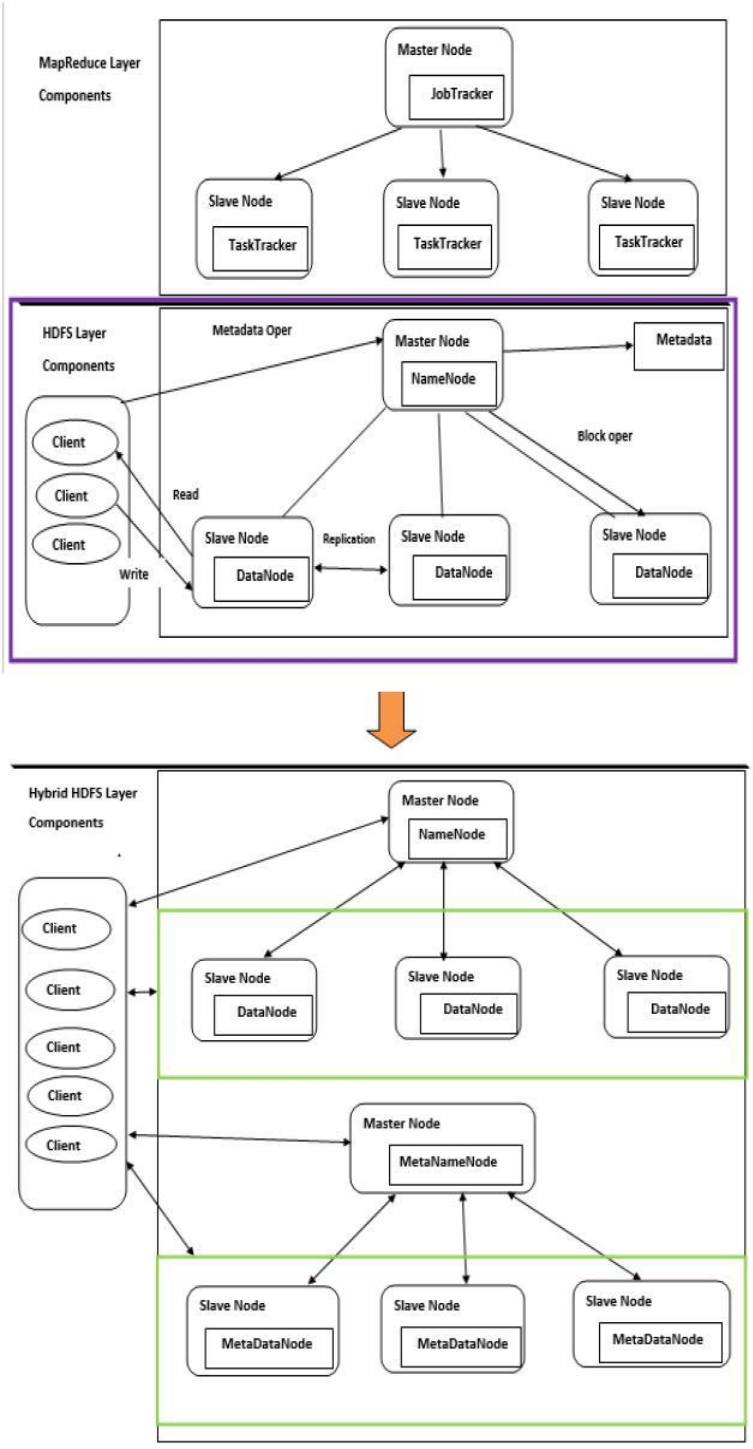

Figure 1. Hybrid distributed HDFS Architecture

B. *Design style principles*

As mentioned before, HDFS provides an extremely resistant and well documented system storage. It depends on a fault tolerance software architecture design. Unfortunately, its unique NameNode is a point of failure that reduces the availability of the system. The NameNode coordinates access to data in the file system and manage the distribution of data.

The purpose of this work is to identify the software architecture style that fulfill the architectural requirements for Hadoop and mainly remove the single point of failure. The structure of structures and modules of Hadoop will be defined and discussed which will help more understanding Hadoop system and will provide a basis for a more in-depth analysis of Hadoop design, consistency analysis, compliance testing and dependency analysis. It includes software components, how these components are organized, the visible external properties of these components and the relationship between them. The new model will help in designing and constructing the new Hadoop system architecture and its behavior. Detecting the new challenges and impact on the requirement of the system will allow to reduce these impact and risks on the new architecture and enhance its quality attribute.

To approach the increasing and main requirement and features for managing Hadoop, the proposed architecture is a hybrid style composed of a distributed style and a master slave style. It is based on separation of concern. This separation will be beneficial specifically because of the complexity of the whole system and will decrease the amount of responsibilities performed by the NameNode and thus get rid of the single point of failure. The NameNode is responsible of managing the whole system and coordination of metadata and access file to HDFS. It is the only component that can trace and localize data in HDFS which will slow down the system when many processes are willing to access the same file. The idea is to add a new distributed master slave system to the HDFS part to separate responsibilities and decentralize mainly the metadata management. Even if we think that generally the size of metadata is smaller than the data itself, it is actually the most data accessed by the client in the overall file system accesses. This is why it is crucial to include another architecture to manage it and distribute it because 50% to 80% accesses are assumed in a large scale distributed system like Hadoop.

The new architecture is composed of different distributed processes that communicate with different other processes. The above figure (Figure 1) depicts the view of the main components and connectors of this hybrid architecture. HDFS contains two master slave architecture: data master slave architecture and metadata master slave architecture.

1) HDFS Clients component: HDFS Clients component: create, read, write and append data from/to files. Many concurrent clients are expected, and they may all access the same file.

2) Data master slave architecture:

a) Master NameNode component: Master NameNode component retains details about information of the available storage space and plan the location of the new blocks. A new strategy will be deployed to manage the distribution of the data.

*b) Slaves DataNode component***:** It stores the blocks by writes and appends.

3) Metadata master slave architecture:

a) Master MetaNameNode component: Responsible of allocating innovation numbers to writers and appenders and inform the readers about the innovation numbers for each block. This will allow to have a consistent system. Old and new innovations for the same block are accessible.

b) Slaves MetadataNode component: Store the metadata that allow identifying the chunks that make up a snapshot innovation. To improve concurrent access to metadata, a distributed metadata management architecture is presented.

Removing the single point of failure will improve the performance of Hadoop.

C. Design issues and challenges

The proposed architecture is a distributed model in which different components are distributed and managed over the whole platform and work together towards executing the same goal. The single point of failure will be removed by adding another distributed system for metadata and separate the responsibilities of the NameNode.

The underlying infrastructure of the proposed distributed model is based on availability transparency and reliability. It promotes consistency, availability and scalability. The new model introduces architectural software challenges that need to be addressed with a proper software architecture tradeoff in terms of distributed system:

1) Complexity of the system: Two distributed systems combined to assure the main functions of the system make the system more difficult to handle, maintain and coordinate. The parts of the system are independently managed. But if properly organized in its software components, the system will be able to manage that complexity. Different strategies will be employed to be sure each part of the system is well managed and consistent with other components of the system. With these new strategies, less synchronization will be needed.

2) Scalability: Scalability may decrease as implementation complexity of the system increases. So, being able to decrease the complexity of the system and well manage it will help in improving scalability of the system while there is a significant number of processes accessing and using concurrently the system.

3) Manageability: Managing the system will need more effort and methods. It is important to ensure that the system is easy to operate. Having a system combining three different architectural styles will make it difficult to manage. That is why we will need to make sure that the system is easy to modify and update and can handle failures. A strategy based on free concurrency access will be used. HDFS Clients writes concurrently the modifications and updates to the system and the MetaNameNode take in charge the assignment of new innovation numbers to the system.

4) Communication: Communication and more specifically between components may be more difficult specifically to access files in HDFS. For every read and write, the client Hadoop will contact the MetaNameNode or the NameNode and if that communication is not managed, it could make the system less performant. An invocation style will be used to manage that. Using the publish-subscribe style to handle the communication between clients and the MetaNameNode will improve the quality of the communication and the performance of the system. It may add more complexity to the system but will make the system more consistent and make the MetaNamenode and the client nodes low coupled.

5) Performance: Communication between components of the new hybrid system may add an overload to the system and thus impact the performance of the system. A method to make the system a lock free system will help in decreasing the overhead and waiting time among the concurrent processes nodes.

6) Consistency: Will be guaranteed besides replication by using a system for metadata management that depend on a uniquely recognized shared innovation number in the system. This method is explained above.

VI. CONCLUSION

In this paper, we have mainly addressed the challenge of building big data systems using Hadoop. We described the different features and architecture styles that exist and discussed their design for Hadoop. We came up with a new efficient architecture style for Hadoop that satisfy most of the requirements and quality attributes promoted by this kind of system. Challenges and issues raised by the new architectural pattern have been identified and analyzed.

REFERENCES

[1] Shvachko, K., Kuang, H.R., Radia, S., et al., The Hadoop Distributed File System. IEEE 26th Symposium on Mass Storage Systems and Technologies, p.1-10, 2010.

[2] Dev, D., Patgiri, R., Performance evaluation of HDFS in big data management. International Conference on High Performance Computing and Applications, p 1-7, 2014.

[3] Bing Li, Yutao He, Ke Xu, "Distributed Metadata Management Scheme in Cloud Computing ", In Proceedings of IEEE in PCN&CAD CENTER, Beijing University of Post and Telecommunication, China, 2011.

[4] Ze Deng, Dong Wei, Wei-zhou Peng, Si-fa Zhang, "Constructing a Two-Level Topology-Aware Distributed Hash Table with a Hierarchical Network Coordinate System", International Journal of Digital Content Technology and its Applications, vol. 5, no. 8, pp. 203-214, 2011.

[5] Fairbanks, George. Just enough software architecture: a risk-driven approach. Marshall & Brainerd, 2010.

[6] Mikhail J. Atallah & Marina Blanton, editors (2010): Algorithms and Theory of Computation Handbook: General Concepts and Techniques, 2 edition. Chapman & Hall/CRC.

[7] Jost Berthold, Mischa Dieterle & Rita Loogen (2009): Implementing parallel Google map-reduce in Eden. In: Euro-Par 2009 Parallel Processing, Springer, pp. 990–1002.

[8] Shumo Chu & James Cheng (2012): Triangle Listing in Massive Networks. ACM Trans. Knowl. Discov. Data 6(4), pp. 17:1–17:32.

[9] Jonathan Cohen (2009): Graph Twiddling in a MapReduce World. Computing in Science and Engineering 11(4), pp. 29–41.

[10] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy & Russell Sears (2010): MapReduce Online. In: Proceedings of the 7th USENIX Conference on Networked Systems

[11] Design and Implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, pp. 21–21.