Limitations and Challenges of HDFS and MapReduce

Jean-François Weets[1], Manish Kumar Kakhani[2], Anil Kumar[2] (Senior Member, IEEE)
[1]Ecole des Mines de Nantes (Nantes), France
[2]Mody University of Science and Technology, Lakshmangarh, India
jfweets1@gmail.com, manishkakhani@gmail.com, dahiyaanil@yahoo.com

Abstract*— Over these past 6 years, Hadoop has become a highly popular solution to store and process a large amount of data for analysis purpose. Those 6 years of utilization along with the researches undergone which focused on Hadoop enable researches to have a good overview of its advantages, drawbacks and limitations in order to improve the solution initiated introduced. Even though Hadoop 2.0 released in 2012 brought several improvements, especially regarding Hadoop Distributed File System (HDFS) availability and the cluster resource management during MapReduce job execution through the YARN architecture, numerous scopes of improvements are yet to be explored. This paper aims to present those drawbacks and limitations of Hadoop 1.0, explain what brings Hadoop 2.0 and what these remaining scopes of improvements for Hadoop are.

Keywords—Hadoop, MapReduce, HDFS

I. INTRODUCTION

The Big Data is a term to call a tremendous amount of heterogeneous data generated and fall into disuse at high speed. This can be summarize by the 3V: Volume, Velocity and Variety. Big Data analysis has become over the past few years a crucial issue for big companies as the growth of the data production has skyrocketed during the same time. Indeed, being able to proceed Big Data allows business leaders to make more accurate and then more truthful decisions. Long gone are the days when Bill Gates stated "640KB is ought to enough for anybody" (1981), in 2000, big firms stored hundreds of Giga Bytes of data. Nowadays the order of magnitude of data stored by the same one is near the Petabyte even Exabyte. To handle this huge amount of data, it requires to develop the new technologies and techniques to store, process and visualize it cost-effectively. Several big companies develop their own Big Data Analysis framework for distributed system as Microsoft with Dryad. In 2004, Google's GFS and MapReduce Programming language presentations were a huge step in this domain since it provides a real guideline to create efficient highly scalable distributed file system along with a new algorithm, MapReduce algorithm, to proceed large dataset fairly quickly that can be adapted too many problems despite of being restricted programming language. Five years after, Doug Cutting presents a new open-source distributed framework to store and process large amounts of data inspired by Google previous works, Hadoop. This new framework, which belongs to Apache software foundation projects since 2009, becomes rapidly popular and is now widely used by thousands of companies such as Yahoo! And Facebook for its high reliability, its gratuity and its implementation of MapReduce. Followed in 2012 by the release of Hadoop 2.0 which rectified several drawbacks and limitations of the previous version

The rest of this paper is organized as follows: the section 2 presents the two core components of Hadoop HDFS and MapReduce programming language in order to have a clear overview of them before discussing regarding their limitations respectively in the sections 3 and 4 along with solutions proposed in some research papers.

II. Background

2.1 HDFS

HDFS is the storage core component of Hadoop. It is a distributed open source file system widely inspired by GFS which is highly scalable and reliable. It is composed of the collection of nodes following master-slave architecture.

The NameNode is responsible for the metadata of the system that it stores in memory to have a quick access for them. It handles three types of metadata:

- The namespace
- The records of all transactions undergone
- The block locations

By handling these metadata, the NameNode is enabled to fulfil goals which are:

- Allowing and enables the client to get quickly access to the required blocks to use usual operations on them such as read, overwrite, append and delete operations
- Applying the block placement policy and the replication mechanism to ensure data availability and durability
- Assigning new block location while maintaining the load balance of the cluster

Whenever the NameNode restarts, since the memory is lost, to retrieve its previous state it reads two files stored in local disk: the EditLog which record in a real-time manner all the transactions made and the FSImage that is a snapshot the namespace metadata periodically generated. When the EditLog become too huge, the NameNode schedules the generation of a new FSImage. Once generated, the EditLog is cleared. Finally, it recovers the mapping of the block from each DataNode on the cluster during the "handshake" exchange sent when the NameNode is restarting.

978-1-4673-7910-6/15/$31.00 ©2015 IEEE

Furthermore, Hadoop 1.0 provides a secondary cold standby NameNode which takes a periodic snapshot of the HDFS metadata. It enables in case of NameNode failure, to manually launch the secondary NameNode to take the lead of the cluster.

The DataNode stored the data blocks. When new data is requested to be stored in HDFS, it is split into blocks with a fixed pre-configured size (usually 64/128/256 or 512MB). These blocks are stored in the DataNode and then are immediately replicated a fixed pre-configured number of times (usually 3 times) and stored in different nodes.

The DataNodes and the NameNode periodically (typically each few seconds) communicates through Heartbeat messages which enables the NameNode to know which node is unavailable among others useful information regarding the nodes

To avoid overloading the network, the client communicate directly with the DataNode whenever it needs to get access to data. When a client forwards a request to HDFS, the NameNode first send back the location of the blocks required by the client after verifying the associatedpermissions. Then the client directly orders to the DataNode to execute the required operations.

2.2 Map Reduce

Map Reduce programming language is a restricted programming language that aims to process a massive amount of data in parallel. Map Reduce jobs are composed of two tasks: the map tasks and the reduce tasks. To manage and execute the job, Hadoop provides two daemons.The JobTracker which submits the job to the cluster, schedules the tasks then assigns them to slave nodes. The TaskTracker is in charge of executing the task assigned by the JobTracker.

The job process follows several phases. First, the data to process are retrievedfrom HDFS, split of chunks of data (one key/value pair per chunk) and given to map workers which stored them into buffers. Afterwards the chunk provided is mapped into a collection of key/value pairs before being reduced, stored in local disk and partitioned in region to decide which record will be sent to which reduce worker. These map outputs are also called intermediate data.

During the shuffle phase, the reduce worker is notified by the JobTracker of the completion of all the involved map tasks along with the location of the intermediate data to retrieve. After collecting all this information, the reduce workers fetches throughout the cluster the remoted map outputs to sort, merge, and finally reduce them. At the end, the reduce output is written in a new file stored in HDFS.

To avoid too huge data transfers and overload the network during the job execution, Hadoop use the “move compute to data” paradigm are much as possible, processing the data locally whenever it is possible, especially during the map task.

III.HDFS limitations

HDFS as it has been first introduced in 2009 knows several drawbacks in different matter such as its high availability’s and security’s lake of guarantees or its scalability limitations. Yet, Hadoop 2.0 marks a huge step in providing high availability for HDFS.

2.3 Availability

One of the first problem pointed out after the release of Hadoop 1.0 in 2009 was its lack of high availability. HDFS is highly reliable and fault-tolerant and provides a strong durability guarantee through its replication and node decommissioning mechanisms which prevent HDFS from losing data along with the secondary NameNode which protects the metadata against NameNode failure [4].

However, the availability guaranteed by Hadoop 1.0 was not that strong. Its secondary NameNode stored just a periodic snapshot of the HDFS metadata and can only replace the primary NameNode manually. Thus, the whole process could take more than 1 hour which is extremely long which makes the primary NameNode a single point of failures a given moment it is the only one to fully possess the most updated metadata of HDFSsince the secondary NameNode is not synchronized [1].

In 2010, Robert Chansler, one of the developers of Hadoop project published a paper analyzed availability issues pointing out the causes of loss of availability: the Garbage Collector, excused failures such as loss of power, hardware failure or misconfiguration, software bugs, space contention and user applications ordered according to their frequency.[4]He also exposed axis of improvements such as better cluster management to avoid encountering hardware limits, better provisioned servers, reducing restarting duration, tuning carefully GC parameters, using of quota to the resource-intensive applications. To provide high availability, he also states some ideas such as having a hot standby second NameNode enables to execute the namespace application with a negligible delay.

In 2012, Hadoop 2.0 is released and among improvements brought, itallows the replication the NameNode following the same mechanism as for the DataNodes. Furthermore, all NameNode stay synchronized to each other, sharing the same directory to edit and read logs. Consequently, NameNode can fail over the standby NameNode in an only few seconds whenever a problem appears. This system thus provides high availability performance.

However some enhancements could still be undergone such as implementing a BookKeeper, a high available write-ahead logging system, replicating shared edits directories, acknowledging the receipt of an edit log entries by a quorum of other NameNodes before answering to the client’s request or storing edit logs in HDFS[1].

2.4 Security

While Hadoop is now widely used throughout the world, the security provided is still poor when in the same time, firms and private individuals store more and more critical data: credit card number, passwords, and others type of sensitive data. The security matter has been widely negligible and then remains one the main axis of improvements. We need to develop some cryptographic framework and robust algorithm must be develop in order to enhance the security of the data [8].

Regarding HDFS,The security system is very poor and HDFS needs to be protected from vulnerabilities and breaches. Usually the administrators ensures by others means the security of the cluster like providing a Firewall, providing an Intrusion detection system and/or encrypting blocks and nodes.

Some solutions that already exists and adaptable to Hadoop was proposed by B. Saraladevi in 2015 as Kerberos, a ticket-based network authentication protocol adapted for distributed system or blue eye algorithm which secures sensitive data (even encrypted one) by checking attacks, breaches and theft attempts and giving permissions accesses to particular clients.

2.5 Scalability

The sole real scalability threat is the limitations of the memory space available for the NameNode. Whereas, Hadoop administrator is allowed to easily add and remove as many nodes as he wants to his cluster to match his need, the quantity of metadata in the NameNode is limited since it is stored in memory. This memory limitation is a scalability bottleneck.

However, even put aside the memory constraint of the NameNode, the scalability of a Hadoop Cluster is not infinite since there are also physical issues like the cluster size by instance which does not allow the cluster's administrator to add nodes endlessly. Nonetheless, their impacts on the scalability feature of HDFS is sufficiently low to be negligible.

IV. MapReduce limitations

MapReduce programming language as it was originally implements in Hadoop possesses two main weak points, frequently discussed in papers [7],[10],[11],[12],[13] these last few years, both closely linked: the task scheduling system and the shuffling Phase. These weak points are themselves closely related to a main issue: the cluster and nodes resource management.

2.6 Poor tasks scheduling strategyand resource management

The task scheduling strategy of Hadoop 1.0 is a pre-configured slot-based one which assigns tasks according to the number of CPUs possessed by the cluster and the type of slots: map or reduce. Usually one task slot per CPU. However, this strategy is poor since it does not take into account others crucial parameters like the heterogeneous consumption of resource of the different phases the task go through, the memory resource, or the data skew that possibility may significantly increase the duration of the MapReduce job execution.

Hadoop 2.0 brings some improvements in one of main issue of the previous version: the resource manager during MapReduce job execution, thanks to YARN, the third core component of Hadoop. This is a cluster resource manager working along with the MapReduce. The user application can more precisely specifies the task needs in memory, CPU, disk, network among others which allows the application manager allocated resources through requesting a container for the application according to those needs. Consequently, YARN optimizes the resource utilization during the job execution.

Nonetheless, the current Hadoop job scheduler can still be improved. Comparing to the solutions proposed further, YARN remains a too coarse-grained resource manager since the Hadoop NextGen's cluster resource manager is a static task-level scheduler. PRISM proposed by Xie et al., phase-level job scheduler system which monitors and manages dynamically the resources allocated to the different phases of each tasks in order to maintain a high utilization of the hardware resources throughput the MapReduce job execution [13]. Furthermore, PRISM relies on jobs profiler such as Starfish to collect data regarding the job application needs. Another solution, Mammoth has been released this year by XuanHua Shi et al., which focus on optimizing the usage of memory for poor memory resource cluster such as HPC systems [12]. They noticed several weak points of Hadoop memory management for MapReduce: the static configuration of the memory allocation, the one-task assigned buffers, the lake of concurrent task running strategy and the I/O negative impact in memory in Hadoop during the shuffling phase. To solve this issue, they proposed a global node memory management where buffers are shared among tasks and dynamically assigned to one of them according to its needs, to replace the static split memory strategy among fully independent tasks of Hadoop through a Case Scheduler. Mammoth also provide an I/O concurrency strategy to mitigate the I/O impact on memory through its I/O scheduler.

The Data Skew is analso common issue while processing data in parallel. It occurs while processing of unbalanced amount of data among the different workers and stems from the inherent properties of the data. Thus, as the workload is unequally divided, some workers will fully complete their tasks later than others. Implied to the MapReduce, we can distinguish two type of data skew issues: those related to map task and those related to reduce task. However, map skew as it is called is not a real issue. Indeed, map skew only comes from the random difficulty to process the chunks of data since the chunks themselves have all the same size. The resulting gaps are negligible and fully compensated by reducing the network overhead by spreading through time the completion of the map tasks and avoiding the fetching of all the intermediate data produced by the reducers at once. The real challenge is the reduce skew since it is a consequence of both

the random difficulty of processing data but most the size of the data meant to be reduce by each reducers which can greatly vary and create stragglers. To address this issue, some solution have been issued like LIBRA, lightweight data skew Mitigation strategy proposed by Chen et al. [11] which mitigates Data skew by analyzing by the master a sample of map outputs in order to make a clever partition decision before notifying the workers regarding the adopted partition strategy. LIBRA also provides cluster split strategy for jobs with a high degree of data skew when the first method is not efficient enough to deal with this issue. Thus, LIBRA uses a proactive approach to prevent stragglers from slow the job completion rather than a speculative approach of Hadoop based on scheduling back-up tasks.

2.7 The Shuffling Phase

The shuffling phase have been over these last years the focus of many studies and researches. This phase is indeed the time-consuming phase of the MapReduce which often overloads the network which leads the intermediate data transfer to dominate the whole process and causes significant I/O contention [7], [9], [10].

The main reason explaining this fact is that the reducer fetch all intermediate data at once as soon as the last mapper notified the JobTracker regarding the completion of its task. It would be sufficient that several reducers triggered the shuffling phase barely at the same time and a numerous of reading requests would be mappers retrieve the map outputs which would eventually cause to I/O contention. As an answer, a huge amount of data would sent to the reducer nodes which would create a data transfer overhead.

To enhance the performance of this phase, some solution was proposed as the predictive scheduling and prefetching process of Jiong et al., which consist in preloading data before assigning tasks, shortening the waiting period between two tasks[9]. Jiong et al., also proposed an adaptive pre-shuffling strategy using a pull model with two-stage pipeline structure rather than the usual push model of Hadoop MapReduce. The principle is simple rather than fetching map outputs all at once by the reducer whenever all the mappers complete their tasks, the map nodes will individually sent their intermediate data to the corresponding reducer without waiting for the other mappers to finish their own tasks. This strategy brings several improvements as enabling reduce task to overlap map tasks and improving Hadoop throughput.

More recently in 2015, Weikuan Yu et al. suggested a new shuffling process that they called virtual shuffling to oppose to the physical shuffling strategy of Hadoop [10]. This virtual shuffling combine three techniques to alleviate the network load and the I/O contention by delaying the fetching of the intermediate data to the last moment where the reduce phase requires it to be done. It is called “virtual” since at the beginning of the reduce task, the reduce stores in memory a 3-level tree structure where each leaves is a virtual representation of a record of map output that possesses some useful attributes such as the record location to retrieve the data whenever it is required.

V. Conclusion

In this paper, HDFS and MapReduce limitations are discussed such as the availability, security and scalability issues of HDFS, the coarse-grained resource management, the poor job scheduler and the time-consuming shuffling phase of MapReduce.If Hadoop 2.0 solves some issues such as the cluster resource management and HDFS availability, some limitations submitted in this paper are not overcome by this new version. Furthermore, some solutions were proposedlater especially to enhance the performance of MapReduce by providing a better job scheduler such as PRISM or to optimizing better the shuffling through virtual shuffling or implementing push model as suggested by JiongXie et al. works. Others issues do not yet get most researchers’ attention, such as the security matter for HDFS while this aspect is a very important one since more and more sensitive data is stored in Hadoop cluster such as Yahoo! Or obviously Facebook ones.

References

[1] Aaron Myers,”High Availability For The Hadoop DistributedFile System (Hdfs)”, Http://Blog.Cloudera.Com/Blog/2012/03/High-Availability-For-The-Hadoop-Distributed-File-System-Hdfs/, March 7 ,2012

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung,” The Google File System”, 2003.

[3] J. Dean and S. Ghemawat, “MapReduce: Simplified data processingon large clusters,” in Proc. 6th Symp. Oper. Syst. Design Implementation(OSDI), Dec. 2004, pp. 137–150.

[4] Robert J. Chansler, “Data availability end durability with HDFS”, 2010

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler,” The Hadoop Distributed File System”,IEEE,2010

[6] Ishwarappa,Anuradha J, “A brief introduction on Big Data 5Vs characteristics and Hadoop technology”, Procedia computer science 48,pp319-324,2015

[7] JiongXie, Yun Tian, Shu Yin, Ji Zhung, XiaojunRuan and Xiao Qin,” Adaptive pre-shuffling in Hadoop clusters”, Procedia computer science 18,pp 2458-2467,2013

[8] B. Saraladevi, N. Pazhaniraja, P. Victor Paul, M.S. SaleemBasha, P. Dhavachelvan, “Big Data and Hadoop-A study in security perspective” Procedia computer science 50,pp 596-601,2015

[9] Jiong Xie, FanJun Meng, Hailong Wang, HongFang Pan, JinHong Cheng, Xiao Qin, “Research on scheduling Scheme for Hadoop clusters” , Procedia computer science 18,pp 2468-2471,2013

[10] Weikuan Yu,Yandong Wang, Xinyu Que, and Cong Xu, “Virtual Shuffling for Efficient Data Movement in MapReduce”, IEEE Transactions on Computers, Vol. 64, No. 2, February 2015

[11] Qi Chen, Jinyu Yao, and Zhen Xiao,” LIBRA: Lightweight Data Skew Mitigation in MapReduce”, IEEE Transactions on Parallel And Distributed Systems, Vol. 26, No. 9, September 2015.

[12] Xuanhua Shi, Ming Chen, et al. ,” Mammoth: Gearing Hadoop Towards Memory-Intensive MapReduce

Applications", IEEE Transactions on Parallel And Distributed Systems, Vol. 26, No. 8, August 2015.

[13] Qi Zhang, M. F. Zhani, et al. ,"PRISM: Fine-Grained Resource-Aware Scheduling for MapReduce", IEEE Transactions on Cloud Computing, Vol. 3, No. 2, April/June 2015.

[14] Yunfeng Zhu, Jian Lin, Patrick P.C. Lee, and YinLong Xu, "Boosting Degraded reads in heterogeneous Erasure-Coded Storage Systems", IEEE Transactions on Computers, Vol. 64, No. 8, August 2015.

[15] Seema Acharya and Subhashini,"Big Data and Analysis", Wiley, 2015.