Investigating Hadoop Architecture and Fault Tolerance in Map-Reduce

Armin Kashkouli[1*], Behzad Soleimani[2], mina rahbari[3]

[*1]Corresponding Author, Computer Department, Faculty of Engineering, Islamic Azad University, Isfahan (Khorasgan) Branch, Isfahan, Iran

[2]Computer Department, Faculty of Computer & Electrical, University of Kashan, Kashan, Isfahan, Iran.

[3]Computer Department, Faculty of Engineering, Islamic Azad University, Isfahan (Khorasgan) Branch, Isfahan, Iran

Abstract

Map-Reduce is often used in implementation of critical and important tasks such as analysis of the scientific data. However, evidences in the past indicate the presence of optional errors that can destroy the results of Map-Reduce. Of course, run times of Map-Reduce like Hadoop can tolerate crash errors, but do not tolerate arbitrary or Byzantine errors. Hence in this paper, at first, the Hadoop architecture in distributed system will be investigated and then Hadoop will be compared with Map-Reduce and finally the Map-Reduce fault tolerance will be investigated.

Key words:

Map-Reduce, Hadoop, Fault Tolerance, Task, Job, Architecture.

1. Introduction

Map-Reduce is a framework that has been created by Google for batch processing of large data which consists of a programming model and a run-time system that is widely used by Google in its data centers that supports its main work tasks such as processing parameters for its Web search engine. Google execution is not available publicly but it is an open source version that is widely used by computational companies, such as Amazon, Facebook, Rock spice, LinkedIn, Twitter and Yahoo.

Map reduce is also a new approach for scientific computing. The last argument on the importance of map reduce is related to the release of economic versions such as Map-Reduce window azure and Map-Reduce Amazon elastic.

Map reduce is designed to have fault-tolerant capability, because in the scale of thousands of computers of and hundreds other devices such as network switches, routers and power units, components errors occur frequently. For example, in the first year of Google Assembly the president of Google reported that, there were a thousand of machine errors and thousands of hard drive errors. Map-Reduce of Hadoop and Google Maps often endure crash and reduce tasks. If one of these tasks before the work outcome stops, it will be identified and new instance of the task will be created.

While enduring tasks crash and the loss of data on the disc is crucial, but other errors that affect the accuracy of Map-Reduce results certainly will occur in the future. A new study on the dram errors in a large number of servers in Google's database for a period of 2.5 years showed that such errors are more common than previously thought and Dual In-line Memory Module (DIMM) are affected 8 per cent annually; though, be protected by an error correct codes [1]. A research by Microsoft on the one million customers' PC revealed that CPU and core parts errors often occur. Slow implementation tolerance mechanisms of Map-Reduce that are called Hadoop cannot cope with this potential errors or optional errors (we did not consider pessimistic errors). These errors cannot be identified by CheckSum and often will not lead to a task crash that has an impact on it. Consequently, it can destroy the result of a task invisibly and gently. These errors should be identified and their effects should be covered by the execution of each duty [2]. This basic idea has been suggested in the voluntary computing platform to stimulate the biased volunteers that their activity as a result of wrong results will be returned. But that work considered the application of qualitative tasks that are easier than Map-Reduce jobs. A similar but more general solution is more precise machinery byzantine fault tolerant approach, in which a set of programs in parallel can be run by different servers that implement the commands similarly but such an approach directly would not be useful to respond Map-Reduce tasks that only follow client-server model services (such as a file server). If the results do not match a simple solution for Map-Reduce is implementation any work for 2 times and re-execution of the work but these solutions cost a lot if there is any mistake [3].

2. The outstanding feature of Hadoop

Hadoop is a software environment for writing and implementing distributed applications that processes huge amount of data. Hadoop was originally developed to support the search engine project. Nowadays Hadoop is an important part of the computing infrastructure for many internet companies such as LinkedIn. Many traditional tasks such as telecommunications and media have adapted themselves with these systems. Hadoop and processing

Manuscript received June 5, 2017
Manuscript revised June 20, 2017

distributed large-scale data rapidly are becoming a skill and an essential part of the programming toolbox [4].
Among the benefits of Hadoop we can mention the following elements:

2.1 Accessibility

Hadoop operates on large clusters of machines or cloud computing services.

2.2 Strength

Hadoop is the architecture by assuming the frequent disorders in hardware performance and consequently it is very convenient to deal with such disruptions.

2.3 Scalability

Hadoop is scalable linearly to handle big data by adding more nodes to the cluster.

2.4 Simplicity

Hadoop allows users to write efficient parallel codes [5].

3. Hadoop against distributed systems and SQL databases

A viable alternative for Moore's law, when making larger servers, is connecting a large number of smaller machines to each other to act as a distributed system. Although they are suitable for storing large amounts of data but are inefficient to process this massive and intensive data, these architectures require continuous data transfer between the server and the client but Hadoop focuses on the code transfer to the data place instead of continuous transfer and in a Hadoop cluster there are both of the data and the calculations. Most important of all is that philosophy of data transferring to the code location within the cluster has been broken and as far as possible calculations are done on the data that resides on the same machine. Here data transferring the data will take more time than calculations action on data, data remains in its current location and only executable code is transferred to the host machine language [5].
The main reason for the popularity of Hadoop compared with SQL databases is that SQL is designed to work with structured data; on the other hand, early Hadoop applications are presented to work on the unstructured data such as texts, from this perspective, Hadoop is a more general pattern.
In fact, Hadoop and SQL complete each other.
Hadoop unlike SQL that stores data in relational tables with the resident schema, uses pairs of (key, value) as a base unit that is flexible for a variety of unstructured data. Hadoop, instead of using SQL declarative language uses the Map Reduce functional programming language and instead of query expressions codes and under Map Reduce scripts. Hadoop is the best choice for write-once and read many times and by default uses KFKF queue and five scheduling priorities for tasks scheduling [5].

4. Hadoop architecture

Hadoop uses Master/slave architecture for data storage systems and distributed computing.
Data storage file system in Hadoop is known as HDFS. A small cluster of Hadoop includes the following components:

4.1 Name Node

Let’s begin with arguably the most vital of the Hadoop daemons—the Name Node. Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop File System, or HDFS. The Name Node is the master of HDFS that directs the slave Data Node daemons to perform the low-level I/O tasks. The Name Node is the bookkeeper of HDFS; it keeps track of how your fi les are broken down into fi le blocks, which nodes store those blocks, and the overall health of the distributed file system. The function of the Name Node is memory and I/O intensive. As such, the server hosting the Name Node typically doesn’t store any user data or perform any computations for a Map-Reduce program to lower the workload on the machine. This means that the Name Node server doesn’t double as a Data Node or a Task Tracker.
There is unfortunately a negative aspect to the importance of the Name Node—it’s a single point of failure of your Hadoop cluster. For any of the other daemons, if their host nodes fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or you can quickly restart it. Not so for the Name Node [5].

4-2- Data Node

Each Data Node in the Hadoop file system is the location of reserving actual data and performing assigned tasks. When you want to read or write a HDFS file, file is broken down into blocks and Name Node tells the client that which blocks does have each Data Node. Clients directly communicate with Data Node to process local File in the blocks. Data Node may communicate with each other to replicate data blocks.
Figure 1 shows the performance of Name Node and Data Node [5].

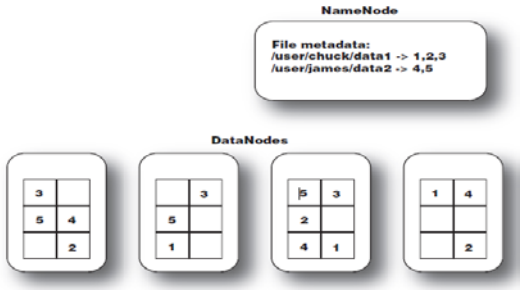

Fig 1: Name Node /Data Node interaction in HDFS. The Name Node keeps track of the fi le metadata—which fi les are in the system and how each fi le is broken down into blocks. The Data Nodes provide backup store of the blocks and constantly report to the Name Node to keep the metadata current [5].

This illustration, each block has three replicas. For example, block 1 (used for data1) is replicated over the three rightmost Data Nodes. This ensures that if any one Data Node crashes or becomes inaccessible over the network, you'll still be able to read the files. Data Nodes are constantly reporting to the Name Node. Upon initialization, each of the Data Nodes informs the Name Node of the blocks it's currently storing. After this mapping is complete, the Data Nodes continually poll the Name Node to provide information regarding local changes as well as receive instructions to create, move, or delete blocks from the local disk. [5].

4.3. Secondary Name Code (SSN4)

The Secondary Name Node (SNN) is an assistant daemon for monitoring the state of the cluster HDFS. Like the Name Node, each cluster has one SNN, and it typically resides on its own machine as well. No other Data Node or Task Tracker daemons run on the same server. The SNN differs from the Name Node in that this process doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the Name Node to take snapshots of the HDFS metadata at intervals defined by the cluster configuration. As mentioned earlier, the Name Node is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data. Nevertheless, a Name Node failure requires human intervention to reconfigure the cluster to use the SNN as the primary Name Node. We'll discuss the recovery process in chapter 8 when we cover best practices for managing your cluster.

4.4 Job Tracker

Job Tracker is the interface between applications and Hadoop. Initially the user delivers the code to the cluster and then Job Tracker provides the action plan to determine which files must be processed and determining the nodes for each section of the work and monitors during the run.

4.5. Task Tracker

Such as storage architecture, computational management sector obeys Master / Slave architecture. Job Tracker manages the implementation of separate tasks on each Slave node.
In Figure 2 Job Tracker structure is provided by Task Tracker.

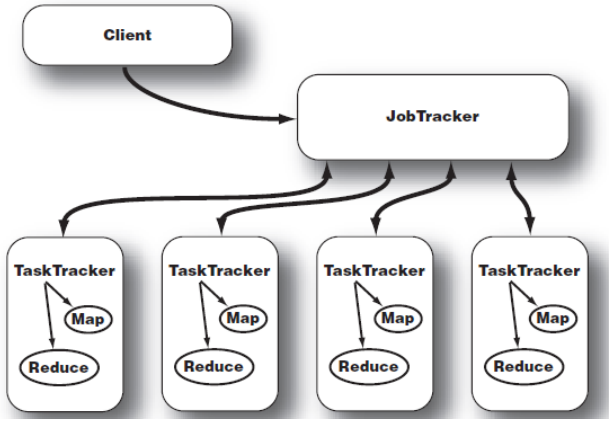

Fig 2: Job Tracker and Task Tracker interaction. After a client calls the Job Tracker to begin a data processing job, the Job Tracker partitions the work and assigns different map and reduce tasks to each Task Tracker in the cluster [5].

One of the Task Tracker tasks is constant communication with Job Tracker. If Job Tracker does not receive a communication from the Task Tracker in specified time then it is assumed that Task Tracker has been failed and again it is presented for other nodes in the cluster of related job [5].
Figure 3 displays the complete Hadoop architecture with different parts of a given cluster.

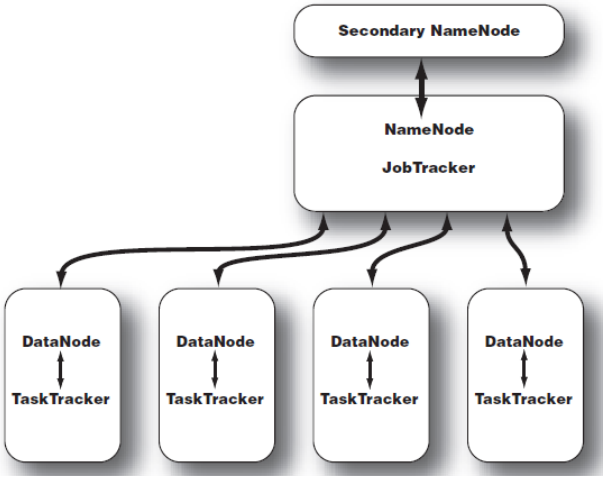

Fig 3: Topology of a typical Hadoop cluster. It's a master/slave architecture in which the Name Node and Job Tracker are masters and the Data Nodes and Task Trackers are slaves [5].

5. Comparing Map-Reduce and Hadoop

Map-Reduce includes a programming model based on reduction functions that can be found in programming (albeit with slightly modified meaning) and is a running space that can be found in groups and databases which has been created by a large number of computers. Programmers define map and define reduction functions. Map is processing input file and production of key-value pairs and reduction function is to create several of these pairs (with the same key). Environment first splits the input files and then feeds several map functions by those pieces then multiple maps outputs are classified as sensitive to keys and then re-split and this time they are placed under the feeding of multiple reduction functions. In the case that it is called disruption, additional reduction outputs finally come together in one output file. According to Dean and Ghemawet expression of all the real activities in the world is possible by using this model. Hadoop is one performance from Map-Reduce that has been created from scratching and it is accessible for free via the Apache License. Hadoop is not just a template for the construction and commissioning of Map-Reduce algorithms in Java but it is also a usable tool to create improved systems and alternative for Map-Reduce, as an example that is provided in this study. Hadoop users offer jobs to it. The input file should already be stored in the Hadoop file system (HDFS) that breaks files into identical copy chunks and it is called gap. These gaps are homogeneously stored in the same nodes that are available to run Hadoop jobs. HDFS is file system (file system) that is suitable for Hadoop. HDFS manages the namespace of a file and allows the user data that be saved in file in chunks and distributed. Blocks of files data can be done on multiple hosts and in three short hooks that two of them are in one hook and the other is other hook. Although it will not implement HDFS and postx but its performance has been set for large data files and result of data (Blocks within mb 1/4) HDFS by a node name (Name Node) unit will be run. Also by using the main server that manages name spare files performance (open and closed and renaming) and processes the client's access to files.

6. Map-Reduce

Map-Reduce is a programming model. It is used for computing large data sets with parallel and distributed algorithms in the cluster. Map-Reduce is the heart of Hadoop.

7. Materials and Methods Fault Tolerance

Fault tolerance is defined as, when the system functions properly without any data loss even if some hardware components of the system has failed. It is very hard to reach cent percent fault tolerance but faults can be tolerated up to some extent. HDFS provide high throughput to access data application and suitable to have large data sets as their input [15]. The main purpose of this fault tolerance is to remove frequently taking place failures, which occurs commonly and disturbs the ordinary functioning of the system.

Single point failure nodes occur when a single node failure causes the entire system to crashes. The primary duty of fault tolerance is to remove such node which disturbs the entire normal functioning of the system [14]. Fault tolerance is one of the major advantages of using Hadoop. The three main solutions which are used to produce fault tolerance are data replication, heartbeat messages and checkpoint and recovery.

8. Fault tolerance in Hadoop

Due to the lack of specific literature about fault tolerance in Hadoop it was necessary to inspect the source code in order to understand how it works. This section provides a summary of the main features that have been analyzed so far.

Regarding the infrastructure of Hadoop, in the rest of the document the term worker will be used to refer to the computing elements in the network, and the term node will be used to refer to a unit composed by both a worker and its corresponding part of the HDFS.

One of the main components of Hadoop is the Job Tracker, which is executed as a daemon process that executes on a master node. The Job Tracker is the scheduler and the main coordinator of tasks. It is in charge of distributing the Map-Reduce tasks between the available computing nodes. Each time the Job Tracker receives a new job to execute, it contacts a set of Task Tracker processes, which are daemons that execute on the working nodes (one Task Tracker exists for each worker in the infrastructure). Map-Reduce tasks are then assigned to those working nodes for which their Task Tracker daemons report that they have available slots for computation (several tasks assigned to the same worker are handled by a single Task Tracker daemon) [16].

The Job Tracker continuously monitors the Task Tracker nodes using control messages named heartbeat signals. These heartbeat signals are sent from the Task Tracker to the Job Tracker and, after receiving this signal, the Job Tracker sends a response including some commands (such as start or end a task, or if the Task Tracker needs to be restarted) to the Task Tracker called Heartbeat Response [17].

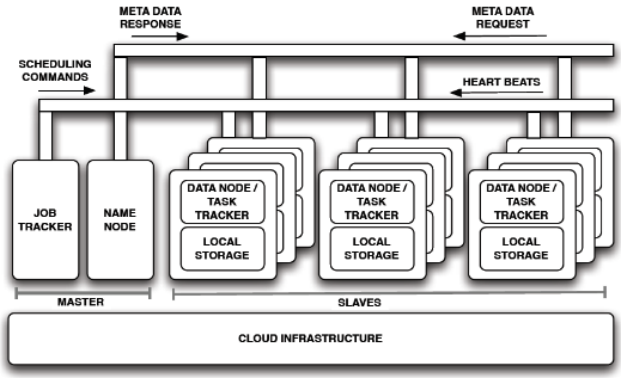

Fig 4: Communications between the Task Trackers and the Job Tracker in Hadoop [17]

In case the Job Tracker does not receive any heartbeat from a Task Tracker for a specified period of time, the Job Tracker understands that the worker associated to that Task Tracker has failed. When this situation happens, the Job Tracker needs to reschedule all pending and in progress tasks to another Task Tracker, because the intermediate data belonging to the failed Task Tracker may not be available anymore. All completed map tasks need also to be rescheduled if they belong to incomplete jobs, because the intermediate results residing in the failed Task Tracker file system may not be accessible to the reduce task. A Task Tracker can also be blacklisted. In this case, the blacklisted Task Tracker remains in communication with the Job Tracker, but no tasks are assigned to the corresponding worker. When a given number of tasks (by default, this number is set to 4) belonging to a specific job managed by a Task Tracker fails, the system considers that a fault has occurred. Then, when a specific Task Tracker has more than a given number of faults (by default, this number is also set to 4) the Task Tracker is blacklisted. There are two possibilities for a Task Tracker to be removed from the blacklist: if one or several faults expires (by default, a fault expires in 24 hours) the Task Tracker is automatically removed from the blacklist, without need of any communication. The other possibility is for the Task Tracker to reboot, when this happens the Job Tracker is notified in the Heartbeat sent by the Task Tracker, and it is removed from the blacklist.

Since one of the most important components of Hadoop related to fault tolerance features is the heartbeat procedure, after learning the basics about how Hadoop works, heartbeat became the focus of the research.

Some of the relevant information in the heartbeats the Task Tracker sends to the Job Tracker are:

- The Task Tracker Status
- Restarted
- If it is the first heartbeat
- If the node requires more tasks to execute

The Task Tracker Status contains information about the worker managed by the Task Tracker, such as available virtual and physical memory and information about the CPU.

The Job Tracker keeps the blacklist with the faulty Task Tracker and also the last heartbeat received from that Task Tracker. So, when a new restarted/first heartbeat is received, the Job Tracker, by using this information, may decide whether to restart the Task Tracker or to remove the Task Tracker from the blacklist. After that, the status of the Task Tracker is updated in the Job Tracker and a Heartbeat Response is created. This Heartbeat Response contains the next actions to be taken by the Task Tracker. If there are tasks to perform, the Task Tracker requires new tasks (this is a parameter of the Heartbeat) and it is not in the blacklist, then cleanup tasks and setup tasks are created (the cleanup/setup mechanisms have not been further investigated yet). In case there are not cleanup or setup tasks to perform, the Job Tracker gets new tasks. When tasks are available, the Lunch Task Action is encapsulated in each of them, and then the Job Tracker also looks up for:

- Tasks to be killed
- Jobs to kill/cleanup
- Tasks whose output has not yet been saved.

All this actions, if they apply, are added to the list of actions to be sent in the Heartbeat Response.

The fault tolerance mechanisms implemented in Hadoop are limited to reassign tasks when a given execution fails. In this situation, two scenarios are supported:

In case a task assigned to a given Task Tracker fails, a communication via the Heartbeat is used to notify the Job Tracker, which will reassign the task to another node if possible.

If a Task Tracker fails, the Job Tracker will notice the faulty situation because it will not receive the Heartbeats from that Task Tracker. Then, the Job Tracker will assign the tasks the Task Tracker had to another Task Tracker.

There is also a single point of failure in the Job Tracker, since if it fails, the whole exception fails.

The main benefits of the standard approach for fault tolerance implemented in Hadoop consists on its simplicity and that it seems to work well in local clusters. However, the standard approach is not enough for large distributed infrastructures such as the ones considered in the PERMARE project, as the distance between nodes may be too big, and the time lost in reassigning a task may slow the system. This approach neither considers a pervasive environment where new nodes can appear in the system, so if a new node is available tasks will not be assigned to it, this means that the system will be wasting available resources.

9. Related work in the field of fault tolerance in Map-Reduce

Map-Reduce has been the subject of many researches. Activities have been carried out that the Map-Reduce be used in such a way that works well in many environments and enforces different uses. These applications include multi-core and multi-processor systems in heterogeneous environments such as Amazon, EC2 dynamic competitive environments, possible homogeneous environments with high hiding such as windows azure, twister system applications and intensive applications of memory and CPU. Another important research trend is related to the use of Map-Reduce for scientific computing, Such as handling high-energy physics data analysis and clustering k-means and it is also present for the production of Digital Upgrade models of several systems that acts as Map-Reduce. The programming model provides large data for batch processing. Map-Reduce covers more complex transactions or provides a higher level of abstraction, such as: Nephle and Pig Latin dryad. All these jobs show the important of Map-Reduce model programming but from the standpoint of fault tolerance it has nothing more than the original Map-Reduce. Tolerating optional fault has broad power in the discussion fault tolerance. Voting mechanism to cover Byzantine errors and the distributed system was introduced in the early 1980s. State machinery copying is a generic solution to create a service or byzantine fault tolerance. Enforcement enough action of operating systems and byzantine errors will be practical and has already been shown and long range of its viability has been shown that include libraries such as: Upright and Gbawa. As was pointed out copying of state machine to create Map-Reduce byzantine fault tolerance is not enough. It is possible that the total of Map-Reduce run is copied on multiple servers but costs will rise. Byzantine systems maximum for data storage have been used to run simultaneously with multiple Semantics (even virtually). Although the voting techniques have similarities to function in our system but these solutions cannot be used in the implementation of the BFT[1] Map-Reduce. We do not have storage service, but we have a system for processing. For voluntary tasks set computing performance, Luis Sarmenta offered a method based on the voting for fault tolerance. That job is mainly related to the workers' timing that specific kinds of errors should not be occurred. While we also use voting, we do not consider our workers' malicious behavior and only consider the random errors. So we do not follow something in sophisticated timing unless the two orders of an activity in a knot should be prevented. In addition, most of our activities in the context of the implementation of the two-step processing (map and reduction) and high data volume (sample) are in line with enhancing the performance. This action is quite different from the aim of that article. In other activity, studied the same problem and provided optimal scheduling algorithms. Fernandez and others studied the same problem but few studies were based on probability analysis. We restate that the problem we have explored was different and this analysis is not the objective of the current research. Recently similar research has been offered from voluntary calculations for Map-Reduce users. The similarity of those activities was that both were based on voting. The main difference was that this work was focused on a different environment (voluntary calculations) and did not try to reduce costs and improve the performance. Therefore did not have any optimization that the core activity had. That article also offers possible model for algorithm that the makes it possible to assess the chances of a false result. Of course, we do not discuss it here. The problem of errors tolerance in parallel applications that run on parallel unreliable machines, in the distant past was studied by Kedem and others. However, they offered a solution based on calculations of intermediate steps to reveal errors. In contrast, we do assume that optional errors detection in optional programs is not applicable, so comparing the two or more runs of the same activity may be the only way to discover the wrong processing [11] [10] [9] [8] [7].

10. Conclusion

This paper deals with the important issue of fault tolerance in Map-Reduce and based on the observed results, Map-Reduce been the subject of many researches. Activities has been conducted the Map-Reduce could be used in such a way that works well in many environments and launches different uses. These applications include multi-core and multi-processor systems in heterogeneous environments such as Amazon, EC2 dynamic competitive environments, possible homogeneous environments with high hiding such as windows azure, twister system applications and intensive applications of memory and CPU. Another important research trend is related to the use of Map-Reduce for scientific computing, Such as handling high-energy physics data analysis and clustering k-means and it is also present for the production of Digital Upgrade models of several systems that acts as Map-Reduce. The programming model provides large data for batch processing. Map-Reduce covers more complex transactions or provides a higher level of abstraction, such as: Nephle and Pig Latin dryad. All these jobs show the importance of Map-Reduce model programming but from the standpoint of fault tolerance it

[1] Byzantine Fault-Tolerant

has nothing more than the original Map-Reduce. The running time of one cycle essentially is doubled in small clusters which is almost twice the time. There is a true assumption, first, why intentional errors are few. Second, this equality means that the possibility of more than a false copy of something similar that returns the same output is unimportant.

It is important to note that Map-Reduce incur any number of wrong activities performances at low cost. This is not something that is happening by simple solutions such as the implementation of a job more than once by using the original Hadoop and comparing the output. If any performance is affected by a mistake in any activity job may be repeated for several times that no two outputs can paired together.

References

[1] P. NÄSHOLM, "Extracting Data from NoSQL Databases", Examen sarbeteförmaster examen, English, vol 74, 2012.

[2] A. LITH, and J. MATTSON, "Investigating storage solutions for large data" ,Department of Computer Science and Engineering G¨oteborg, Sweden, 2010.

[3] Ch. Strauch, Kriha, "NoSQL Databases", INTERNET, 2012

[4] T. WHITE, "Hadoop, The Definitive Guide", O'Reilly Media, Inc, 2011.

[5] Ch. Lam, "Hadoop in action", Printed in the United States of America, 2011.

[6] K. Shvachko, H. Kuang, S. Radia,"TheHadoop Distributed File System", IEEE, 2010.

[7] M. Edwards, A. Rambani, and y. ZHU, "Design of Hadoop-based Framework for Analytics of LargeSynchrophasorDatasets", Elsevier B.V., Procedia Computer Science 12, 254-258, 2012.

[8] R. Chansler, H. Kuang, S. Radia, K.Shvachko, and S. Srinivas, "The Hadoop Distributed File System", http://www.aosabook.org/en/HDFS.html.

[9] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design", copyright from internet, 2007.

[10] W. Zhao, "BFT-WS: A Byzantine fault tolerance framework for web services," in Proc. of EDOC'07, 2008, pp. 89–96.

[11] Z. Zheng, T. Zhou, M. Lyu, and I. King, "FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications," in Proc. of ISSRE'10, 2010, pp. 398–407.

[12] B.He, W.Fang, Q.Luo, N.Govindaraju, and T.Wang. Mars: aMap-Reduce framework on graphics processorsACM, 2008.

[13] C.Olston, B.Reed, U.Srivastava, R.Kumar, and A.Tomkins. Pig latin: a not-so-foreignlanguage for data processing. InSIGMOD '08: Proceedings of 2008 ACM SIGMOD international conference on Management of data, ACM, 2008.

[14] Selic, B. 2004. Fault tolerance techniques for Distributed systems.IBM.http://www.ibm.com/developerworks/rational/libray/114.htm.

[15] The Hadoop Distributed File System: Architecture andDesign" by Dhruba Borthakur, http://hadoop.apache.org/docs/r0.18.0/HDFS_design.pdf

[16] Apache Hadoop (2013). Hadoop documentation. Available at http://hadoop.apache.org/, accessed August 2013.

[17] S. Kadirvel and J. Fortes (2013). Towards self-caring Map-Reduce: a study of performance penalties under faults. Concurrency and Computation: Practice and Experience. Online first 28 May 2013. DOI:10.1002/cpe.3044.