Highly Available Hadoop NameNode Architecture

Mohammad Asif Khan, Zulfiqar A. Memon
Department of Computer Science
Sukkur Institute of Business Administration (SIBA)
Sukkur, Pakistan
{asif.khan, zulfiqar}@iba-suk.edu.pk

Sajid Khan
Department of Electronics & Communication Engineering
Hanyang University (ERICA Campus), South Korea
sajidkhan@hanyang.ac.kr

***Abstract*— In past few years Hadoop Distributed File System (HDFS) has been used by many organizations with gigantic data sets and streams of operations on it. HDFS provides distinct features like, high fault tolerance, scalability, etc. The NameNode machine is a single point of failure (SPOF) for a HDFS cluster. If the NameNode machine fails, the system needs to be re-started manually, making the system less available. This paper proposes a highly available architecture and its working principle for the HDFS NameNode against its SPOF utilizing well-known 2-Phase Commit (2PC) Protocol and Election algorithm.**

Keywords: HDFS, NameNode, Availability, Fault Tolerance, 2PC protocol, Election by Bully.

I. INTRODUCTION

Now days there are so many organizations in the world that owns very large amount of data on the internet. To store and process such large amount of data, require special hardware machines, which is not a good deal in terms of money, as well as the processing capability of single machine cannot scale to process such huge data on it. Hadoop [3] is an open source frame work used for processing such very large data sets. Hadoop mainly do this job with the help of MapReduce [12] and Hadoop Distributed file system (*HDFS*) [3].

HDFS is doing its part of work very efficiently in Hadoop since it was developed. *HDFS* works with two types of hardware machines, the DataNode (Slave machine) [3] which is the machine on which application's data is stored and the NameNode (Master machine) [3] which store the metadata of file system. Application's data is stored on DataNode using small blocks of data with usually three replicas. Where NameNode is the only single machine for storing metadata of file system and is the Single Point of Failure (SPOF) for the *HDFS*. SPOF of NameNode machine affects the overall *availability* of Hadoop. When NameNode goes down the entire system become offline and cannot do any operation until NameNode gets restart.

We have thoroughly studied and analyzed the architecture of *HDFS* with focus on its NameNode's SPOF problem. To eliminate the problem of *HFDS* NameNode, we have devised architecture of *HDFS* and its working principle. This architecture and working principle eliminates SPOF of NameNode by initially replicating NameNode at separate machines (in our case two machines) only once and then starts working normally with its DataNodes. The core idea of this architecture is that once NameNode get replicated after that, it will do all operation on its metadata by using well known 2-Phase Commit (*2PC*) Protocol [11] with some variations. Initially the NameNode will be acting as Coordinator NameNode, and its replicas will be Participants NameNode machines. The Coordinator and Participants machines will perform any update to their metadata according to *2PC protocol*. To increase the throughput of update operations we have made some variations in 2PC according to our needs. Due to *2PC protocol* both of the replicas of Coordinator NameNode will be fully updated all time, hence in case Coordinator NameNode fails or even completely goes down then any of its replicas can become Coordinator and hosts the DataNodes. The new Coordinator can be elected using *Election by Bully* algorithm [11].

The section II of this paper discusses the related work which is done against the problem of NameNode's SPOF. Section III of this paper discusses the architecture of *HDFS* and NameNode's Single Point of Failure (SPOF), the section IV of this paper proposes our architecture and its working principle in the umbrella of 2PC protocol and *Election by Bully*. Section V is the future work and conclusion of our work.

II. RELATED WORK

To make Hadoop highly available is the hot topic for Research and Hadoop community, according to Apache software foundation [3], still there is no clear solution to *HDFS*'s NameNode SPOF [10].

Apache Hadoop provides optional feature of adding secondary[6][7] NameNode which periodically takes NameNode's check point by merging *fsimage* and *editlogs* in order to decrease the size of *editlog* file at NameNode. The Secondary NameNode is not capable of hosting DataNodes in the absence of NameNode, it can decrease the restart time of NameNode but cannot eliminate its SPOF.

The core project team of Apache Hadoop is working on Backup [3][7] NameNode which is capable of hosting DataNodes in case of NameNode fails. NameNodes can only have single Backup NameNode, according to Apache they need active participation from researcher for the completion of Backup NameNode. This backup Node continuously contacts Primary NameNode for synchronization purpose which adds more overheads.

Feng Wang, Jie Qiu with team of IBM China [1] have worked on the *availability* of Hadoop through metadata replication. Their solution comprises of three phases, Initialization phase, replication phase and failover phase,

978-0-7695-4959-0/13 $25.00 © 2013 IEEE
DOI 10.1109/ACSAT.2012.52

This solution is the way to the problem solution but adds complexity to the system by having different machines for cluster management and different for failover. In our work this is done with same machine having much less overheads.

One of the projects (*HDFS*-976) [8] of Apache Hadoop has worked for the *availability* of Hadoop by introducing the concept of Avatar Node [8]. They have performed their experiment over the cluster of 1200 nodes. Their main concept is that the system can switch between primary and standby NameNodes as if it was switching to Avatar. In this solution the standby Avatar machine need assistance of Primary NameNode that produces overhead and extra load for the Primary NameNode which is already very busy in client's requests, this can slow down the performance of system.

Ekpe Okorafor and Mensah Kwabena [2] have worked to increase the *availability* of Hadoop by presenting a mathematical model. This mathematical model is based on zookeeper leader election service. In this solution they have only focused on the *availability* of JobTracker [3] not the NameNode.

III. Hadoop Distributed File System (HDFS)

Hadoop provides a distributed file system called Hadoop Distributed File System (*HDFS*) which runs on commodity hardware machines and store very large data set. It provides high bandwidth to application data and suits with applications having large data sets. One of the assumptions made at the time of *HDFS* designing was that "Moving computations is cheaper than Moving Data" [3]. The core idea is to move large set of applications closer to where the data is located and perform computations on it. The throughput is increased in *HDFS* by moving computations near data. *HDFS* can scale up to ten thousands of low cost hardware machines on which data can be stored in the blocks and computation can be partition on these machines using MapReduce[6][12]. MapReduce is a programming model used for processing very large data sets stored on *HDFS* clusters.

A. HDFS Architecture

The system architecture of *HDFS* is made with concept of Master/slave. *HDFS* clusters have two types of machines, the NameNode (master machine) and the DataNode (slave machine).

The NameNode is central point of focus in *HDFS* architecture as it is the only server machine which store metadata of file system, it manages namespace and grant access to file on the request of client. Where the DataNode is machine which stores the application's data. The data on DataNode is stored in small blocks which are made up by splitting large files. Each of these blocks are stored on more than one DataNode machines, usually a block is replicated on three different DataNode machines to have redundancy in case of any DataNode's failure [4]. The NameNode keeps all metadata in its RAM to have fast access to it [6], hence NameNode have more RAM requirement as compare to DataNode. The file operation like opening, creating and closing file is done at DataNode upon the instruction of NameNode [2].

The architecture of *HDFS* is depicted in Fig.1 that shows the machines are connected with each other. The data blocks are replicated on three different DataNodes with replica factor three (in our case), i.e., the small boxes in Fig.1 with label 1,4,7,2 etc are split blocks of huge file. The circles 1, 2, 3 and 4 show the flow of request from client to NameNode and DataNode. All client requests will flow through NameNode and are written below in steps according to Fig.1:

- 1. Client request to NameNode Machine for any operation (read/write) on data stored on *HDFS* cluster.
- 2. NameNode responds to client by giving him concerned DataNode's address where its desire data is replicated.
- 3. Client contacts with DataNode and request operation on data.
- 4. DataNode responds by providing access to client on that data. DataNode also continue to update other replicas in case of write operation.

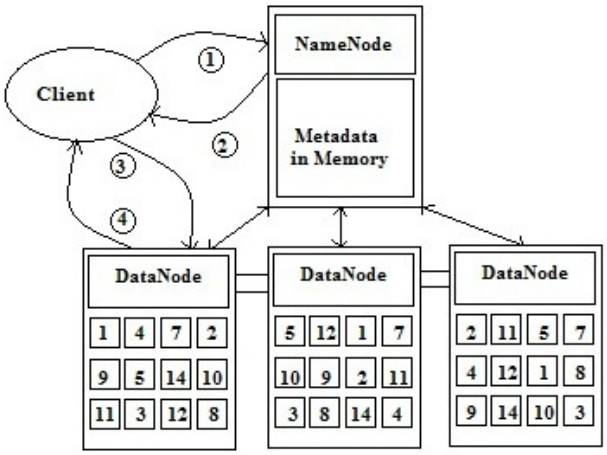

Fig 1. HDFS Architecture

B. Availability of HDFS Architecture

The architecture of *HDFS* is designed to deal with frequent hardware failure as the core theme of Hadoop and cluster computing is that they run over lost cost commodity hardware machines [6], which are more susceptible to failure. Hence the architecture of Hadoop was designed to deal with these failures. The other important feature of Hadoop is its scalability. It can scale up to ten thousands of machines. If you are using single hardware machine with some probability of failure, and on other side you are using ten thousands of such machines then probability of hardware failure will be ten thousand times great than that single machine[3][5].

Hadoop was designed to have features like reliability, *availability*, *fault tolerance*, affordability and scalability. Hadoop exhibits all of these features very well except the

availability feature that it has compromised a bit [5]. The *availability* of Hadoop is very good when it comes to DataNode; as blocks of data are being replicated at three different DataNodes. But when it comes to NameNode; Hadoop's *availability* is always on stack. If the NameNode machine goes down then the whole system becomes offline, as NameNode is the only entity which deals with the client queries and respond accordingly [6]. It also keeps the precious metadata of file system. Hence with single NameNode machine the *availability* of Hadoop is always on stack, because it can go down at any time [7].

C. NameNode's Single Point of Failure(SPOF) Solutions

The basic architecture of Hadoop clearly shows the importance of NameNode machine and its stacked *availability*. Apache software foundation is doing most of the work on this problem of NameNode SPOF. Firstly they introduced a machine called Secondary NameNode. The description of this machine is described earlier in Section II. This machine cannot take over when NameNode goes down. It can only perform check pointing of NameNode machine [6]. The other important solution to this problem is Backup Node which is under development and need contribution from research community [7]. This Backup node can take over in case of NameNode failure. It can do this by keeping the copy of file system namespace in its memory up to date.

As far as author knows, till now there is no complete solution of NameNode's SPOF problem. This has also been admitted by Apache Software foundation on their website that they need active contribution from researcher's to deal with this problem [7].

IV. Highly Available Hadoop HDFS Architecture For NameNode

To eliminate the Single Point of Failure (SPOF) of Hadoop's *HDFS* NameNode, we have devised an architecture of NameNode and its working principle under which the problem of NameNode's SPOF is eliminated and the *availability* of NameNode is ensured.

A. NameNode Replication

To increase the *availability* of any system, the very simple technique that can be used is redundancy [9]. Here at initial step, we have done the same thing by replicating NameNode at two different machines that are called Participant NameNodes. In our case we have taken replica factor two, but it can be varied according to degree of *availability* requirement. Hence when the NameNode starts for the first time [1], it will be replicated only once on Participant NameNode machines. Once replication is complete, the system will then work under the principle of Two Phase Commit Protocol with some modifications. Before discussing our proposed modified form of *2PC protocol*, the actual *2PC protocol* is discussed below.

B. Actual Two Phase Commit(2PC) Protocol

In distributed systems, the Two Phase Commit Protocol is used for committing the final result among the group of processes [11]. In our solution we have used *2PC protocol* but with some variations according to our needs, where the actual form of *2PC protocol* is also discussed under.

The machine which will initiate computation request will be called Coordinator, where the machines which will commit computation on the request of Coordinator will be known as Participant. The protocol is consists of the following two phases:

- Phase 1 a. Coordinator machine will send vote-request to Participant machines for committing a computation.
- Phase 1 b. Upon receiving vote-request, the Participant machine will either send vote-commit or vote-abort, in case if it sends vote-abort, then it will also abort its local computation.
- Phase 2 a. When Coordinator machine receives all votes from Participant machines then: if all of them have sent vote-commit, then Coordinator will send global-commit to all Participants, otherwise it will send global-abort.
- Phase 2 b. Every Participant machine will wait for global-commit or global-abort from Coordinator machine and will respond to it accordingly [11].

C. Two Phase NameNode Commit(2PNNC) Protocol

In our proposed solution once NameNode is replicated (for once in its life), after that NameNode and its replicas will work under the principle of Two Phase Commit (2PC) Protocol. We have not exactly used the same *2PC protocol*, but have made some variations in it according to our needs. Therefore we have renamed this protocol as *Two Phase NameNode Commit (2PNNC)* protocol. Here the NameNode will be called Coordinator NameNode and its replicas will be known as Participant NameNode machines. Every Participant NameNode is associated with an update-queue that will hold multiple vote-requests from the Coordinator. Any of the Participants can send a vote-commit to coordinator, as soon as it receives the vote-request and its update-queue (discussed below) is empty. So, in this case the Coordinator need not wait for all of the vote-commit replies from the Participants, rather it will continue to send global-commit against a single vote-commit from any of the Participant. The remaining Participants can perform their commit operation later, but there must be atleast one Participant who will update itself with the Coordinator at same time. After the vote-request from the Coordinator, if the update-queue is partially filled then the Participant can accept the vote-requests, but will send vote-commit to coordinator until it has performed all the pending operations in its update-queue. The Participant will send vote-abort in the case when its update-queue is full. The working principle of update-queue is explained in detail later in this Section. The *2PNNC* protocol for our proposed solution is explained below:

- Phase 1 a. Coordinator NameNode sends vote-request to all Participant NameNodes.
- Phase 1 b. When any of the Participant NameNode receives vote-request from Coordinator then it will either reply vote-commit or vote-abort. It will send vote-abort if its update-queue is full of previous

vote-requests, otherwise it will keep the vote-request in its update-queue and sends vote-commit to coordinator when its update-queue become empty by performing all previous requested updates.

- Phase 2 a. The Coordinator NameNode collects all votes from the Participants. If it gets vote-commit from any one of its Participants NameNode, it commits its update operation and sends out global-commit to all Participant NameNodes. If it receives vote-abort from any of its Participants NameNodes then it will abort its update operation and will send global-abort to all Participant NameNodes.
- Phase 2 b. Every Participants NameNode machines will wait for Coordinator global-commit and global-abort and will respond accordingly.

1) Participant NameNode Update-Queue: In this architecture every Participant NameNode will maintain a small length FIFO update-queue (see Fig. 2) which will keep track of the incoming vote-request from Coordinator NameNode for updating some part of metadata. The main purpose for this update-queue is to respond to Coordinator quickly, so that the throughput of the system can be increased. The throughput is increased in such a way that, the Coordinator need not to wait for vote-commit replies from all of its Participants. The Participants can commit updates on their metadata later, and will send vote-commit when its queue has become empty. When the update-queue is partially filled or empty, then the Participant NameNode can accept incoming vote-requests from Coordinator for update, but will not send vote-commit until its update-queue is empty and it has performed the all pending update requests. This ensures that there must exists atleast one Participant NameNode that is fully updated with Coordinator NameNode and is capable of hosting system in case of Coordinator failure.

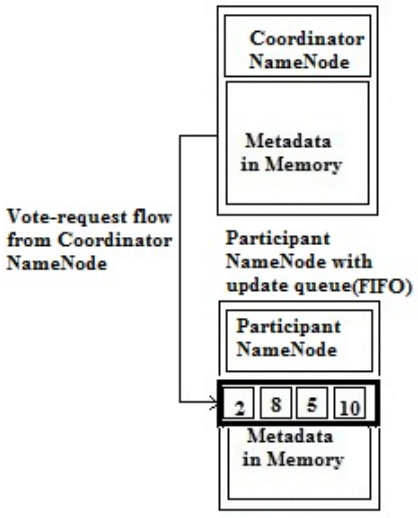

Fig 2. Participant NameNode's Updation Queue

2) Coordinator NameNode Working Principle: This part of the section will discuss the working principle of Coordinator NameNode under *Two Phase NameNode commit (2PNNC)* protocol. Fig .3 gives very clear illustration of its working principle.

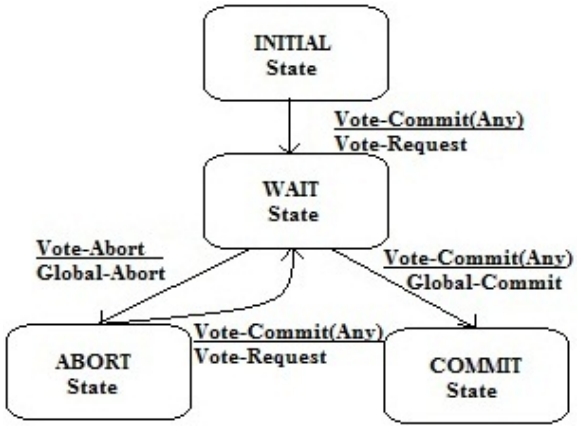

Fig 3. Coordinator NameNode Working Principle under 2PNNC protocol

At INITIAL state the Coordinator NameNode will send vote-request to Participant NameNodes and will go to WAIT state for vote-commit from Participant NameNodes. If any of the Participants send vote-commit then the Coordinator NameNode will commit its update and propagate global-commit to all Participant NameNodes. The reason for committing update without all Participants NameNode is to increase throughput of the process, because in case if Coordinator NameNode fails, then we have atleast one replica or Participant NameNode which is fully updated and is capable of taking over as a new Coordinator NameNode.

If Coordinator NameNode receives vote-abort from all or any Participant NameNode, in that case it will jump to ABORT state and will also abort it local update request, and sends global-abort to all Participant NameNodes. In ABORT state after some fixed amount of time the Coordinator NameNode will again send vote-request to Participant NameNodes and goes to WAIT state. It is because that the Coordinator NameNode wants to commit its previously aborted update and any of the Participant's NameNode might have performed some of their pending jobs in update-queue and they are now able to receive new requests from the Coordinator.

3) Participant NameNode Working Principle: The Participant NameNode will also be working in similar manner according to *2PNNC* protocol. Fig.4 clearly illustrates the working of Participant NameNode.

Here at INITIAL state the Participant NameNode is waiting for vote-request from Coordinator NameNode. Once it gets vote-request from Coordinator it will jump to READY state by sending vote-commit to Coordinator NameNode.

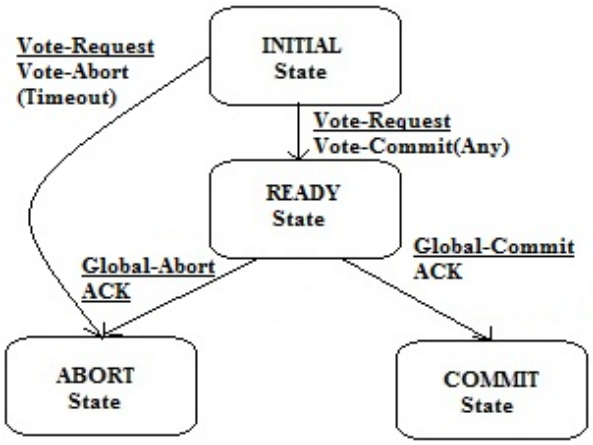

Fig 4. Participant NameNode Working Principle under 2PNNC protocol

At INITIAL state, if after some fixed amount of time the Participant does not receive vote-request from Coordinator NameNode then it will timeout and jumps to ABORT state.

In READY state when Participant NameNode receive global–commit then it will go to COMMIT state. If it receives global-abort then it will jump to ABORT state by aborting and removing last vote-request for update operation.

D. Failure of NameNodes in HDFS and Election Process

As we have described in previous section, that, Hadoop runs over low cost commodity hardware machines, where the probability of failure is higher. Our proposed architecture and working principle of Hadoop NameNode is fully equipped for dealing with such failures. Here in this architecture we can face two kinds of failures, the Coordinator and the Participant NameNode failure.

1) Participant NameNode Failure and Recovery: In the case of Participant NameNode failure, it will be noticed or detected by the Coordinator NameNode machine. This detection is done by Coordinator if it does not receive vote-commit from any of the its Participant NameNode for long duration of time. After that it will send a heartbeat message to that particular Participant NameNode. If it does not replied to Coordinator in fixed amount of time, the Coordinator will declare this Participant NameNode as failed Participant and will generate alarm of Participant's failure.

The failed Participant machine once recover and gets restart, then it sends a heartbeat reply to Coordinator. The Coordinator in response will assign it to another active Participant NameNode machine for replication and synchronization. Once it is being replicated and fully updated from its neighbouring active Participant, it can become a part of active Participant NameNode pool. The replication and updation was assigned to other active Participant to reduce the load over Coordinator NameNode machine.

2) Coordinator NameNode Failure Detection and New Coordinator Election: On every vote-commit, the Participant NameNode will maintain a priority number with it. It will also have a serial number associated with it, which will be assigned to all Participants NameNodes at the time of becoming a member of the pool.This serial is number will not change in their course of execution, e.g in case of three Participant NameNode machines they will have serial numbers as 1, 2 and 3. However the priority number of each Participant will be initialzed as 0(zero) at the begining, and will be increamented when they send vote-commit to the Coordinator.

The Coordinator failure will be detected by any of the Participant NameNode if it gets no vote-request from Coordinator for some fixed amount of time, in response to this, it will send heart beat message to the Coordinator. If Coordinator does not reply to it, then that Participant NameNode machine will start election by Election Algorithm "*Election by Bully*" [11]. Below is the complete process illustration of *Election by Bully* with some variation in it according to our needs

- If any of the Participant NameNode detects the Coordinator NameNode failure, it can start election process by sending an election message with its priority number to all Participant NameNode machines. Assuming that it does not know the priorities of other Participant NameNodes, but knows their serial numbers.
- If a Participant NameNode with high priority number gets an election message from other Participant NameNode machine with low priority number, then it will send take over message to that Participant NameNode and that Participant NameNode is out of the election.
- If the Participants does not get take-over message from any other Participant, then it wins the race of Coordinator election, and declare itself as a Coordinator and start working normally as a Coordinator NameNode.

This is the process of election for new Coordinator. In case if all the Participant NameNodes are fully updated with the last Coordinator that has become failed, then they will have same priority numbers. In this case that Participant NameNode will be elected as Coordinator which have smallest serial number among them.

V. CONCLUSION AND FUTURE WORK

In this paper we have proposed the architecture and its working principle to deal with SPOF problem of NameNode machine. This architecture will greatly increase the *availability* of Hadoop by eliminating NameNode's SPOF. We have utilized two well know techniques of distributed systems, the *2PC protocol* and *Election by Bully* algorithm, with some variations in these techniques by reducing their waiting time and increasing their throughput. Both of these techniques are widely deployed and adopted [11] successfully.

In future, we will deploy this architecture on Hadoop and ensures its applicability. There is also another SPOF in the programming model of Hadoop that is JobTracker [7] which is the only entity for tracking MapReduce tasks. In future, we will extend our proposed model to address the SPOF problem of JobTracker.

REFERENCES

[1] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li and Ying Li, "Hadoop High Availability through Metadata Replication" CloudDB '09 Proceedings of the first international workshop on Cloud data management.

[2] Ekpe Okorafor1 and Mensah Kwabena Patrick,"Availability of Jobtracker machine in hadoop/mapreduce zookeeper coordinated clusters", Advanced Computing: An International Journal (ACIJ), Vol.3, No.3, May 2012.

[3] Dhruba Borthakur, "The Hadoop Distributed File System:Architecture and Design," in Apache Software foundation, http:/ hadoop.apache.org/common/docs/r0.18.0/hdfs _design.pdf .

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler Yahoo!, "The Hadoop Distributed File System," IEEE NASA storage conference, http:// storageconference.org/2010/Papers/MSST/ Shvachko.pdf.

[5] Aaron Myers, "High Availability for the Hadoop Distributed File System (HDFS)," Article at Cloudera, March 07, 2012.

[6] Tom White, "Hadoop: The Definitive Guide". O'Reilly Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009.

[7] The Apache Software Foundation. http:// http://hadoop.apache.org, Last release of march, 2012.

[8] Apache Hadoop Project HDFS-976, "Hadoop AvatarNode High Availability", http://hadoopblog.blogspot.com/2010/02/hadoop-namenode-high-availability.htm, Feburary 6, 2010.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System", Proceeding SOSP'03 ACM October 2003.

[10] Hadoop Wiki, "NameNode Failover", on Wiki Apache Hadoop, http://wiki.apache.org/hadoop/NameNodeFailover, July, 2012.

[11] Andrew S.Tanenbaum and Maarten Van Steen, "Distributed Systems: Principle and Paradigms", Pearson Prentice Hall, Upper Saddle River, NJ 07458.

[12] Jeffrey Dean and Sanjay Ghemawat, "MapReduce:Simplified Data Processing on Large Clusters", Google Inc, OSDI '04: 6th Symposium on Operating Systems Design and Implementation USENIX Association.