International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 02 Issue: 09 | Dec-2015 www.irjet.net p-ISSN: 2395-0072

Balancing & Coordination of Big Data in HDFS with Zookeeper and Flume

Smita Konda, Rohini More

[1] *Assistant Professor, Computer Science & Engineering, A. G. Patil Institute of Technology, Maharashtra, India*
[2] *Assistant Professor, Computer Science & Engineering, A. G. Patil Institute of Technology, Maharashtra, India*

Abstract - ***In the today's age of Computer science & Information technology data processing & storing in very crucial aspect. Currently even a terabytes and petabytes of data is not enough to store large chunks of database. Therefore companies these days make use of concept called Hadoop in their application. Today's data Warehouses are also not able to satisfy data storage needs. Hadoop is deliberate to store huge volume of data sets constantly. Hadoop is well-liked open source software which supports parallel and distributed data processing. Hadoop is highly scalable computer platform. Hadoop allows users to process and store huge amount which is not possible while using less scalable techniques. Some fault tolerance mechanisms are provided by Hadoop so that system works properly even if some failure occurs in system . In this paper we describe an application called Zookeeper which enables to manage, synchronize distributed clusters and allowing coordination in them.***

Key Words: *Hadoop, Fault tolerance, HDFS, Name node, Data node, Zookeeper.*

1. INTRODUCTION

Hadoop is an open source software framework created by Doug cutting and Michael J. Cafarella [1]
The Hadoop Distributed File System (HDFS) is intended to accumulate very big data sets and to stream those data sets at high bandwidth to user applications.
In a large cluster, thousands of servers are present to run user application tasks. A key component of Hadoop is the Hadoop Distributed File System (HDFS), which is used to store all input and output data for applications.
[3] Hadoop uses a distributed user-level filesystem which is written in Java and designed for portability across heterogeneous software and hardware platforms.

1.1 HDFS Architecture

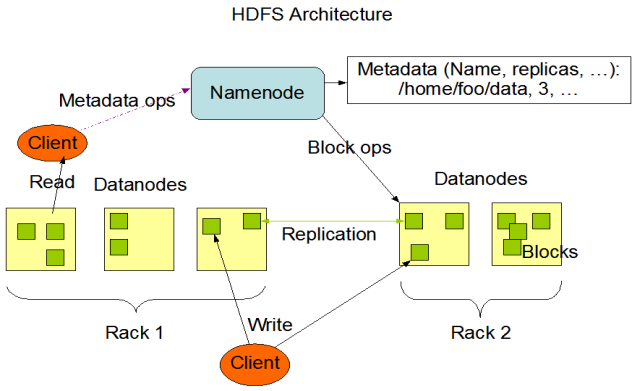

Fig -1: Architecture of HDFS

HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks[1]:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories[1][2].

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system[2].

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode[2][3].

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration[1][2][3].

1.2 Goal of HDFS

- **Fault detection and recovery**: Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets**: HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data**: A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

The File System Namespace

A traditional hierarchical file organization is supported by HDFS. The NameNode is responsible to maintain the file system namespace. Name node keeps the records of changes made to the file system namespace or its properties. HDFS maintains an application that specify the number of replicas of a file. The number of replicas of a file is called the replication factor of that file. This information is stored by the NameNode[4].

Data Replication

HDFS is intended to store very huge files across machines in a large cluster. Each file is stored as a sequence of block; all blocks in a file except the last block are of same size. The blocks of a file are replicated for fault tolerance. An application is used to identify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are written once and have strictly one writer at any time. All decisions regarding replication of blocks is made by Name node. It periodically receives acknowledgement as well as block report from data nodes. Acknowledgement specifies data nodes are working properly & Block report specifies list of blocks in the data node[4].

The Communication Protocols

The top layer of TCP/IP protocol contains all HDFS communication protocols. A client establishes a connection to a configurable TCP port on the NameNode machine. It connects the ClientProtocol with the NameNode. The DataNodes are connected to NameNode using the DataNode Protocol. A Remote Procedure Call (RPC) abstraction is used to wrap both the Client Protocol and the DataNode Protocol. By design, the NameNode does not initiate any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients[4].

Robustness

The primary purpose of HDFS is to accumulate data consistently even in the presence of failures. There are three common types of failures: NameNode failures, DataNode failures and network partitions[4].

2. RELATED WORK

Vishal S Patil, Pravin D. Soni et al. [1] described how fault tolerance is achieved by means ofdata duplication & the framework of Hadoop**.** There are two main methods that are used to create fault tolerance in Hadoop that is Checkpoint & recovery and Data duplication. In Data duplication, the similar copy of data is placed on various data nodes. Hence whenever that data is needed, it is provided by any of data node which is free that is the data node is not busy in communicating with any other nodes. The main benefit of this method is faster recovery from failures. But this method has main drawback that is consumption of huge amount of memory to achieve fault tolerance. It may cause data inconsistency problem. This method provide faster recovery from failures, therefore it is widely used technique than checkpoint & recovery. In checkpoint & recovery method, the concept called rollback is used. If the failure occurs in the middle of transaction, then it just rollback the transaction up to the last consistent stage and then it starts executing transaction again. The main disadvantage of this method is that it is time consuming method as compared to previous data duplication method.

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler et al.[2] illustrated HDFS architecture and given report on their experience using HDFS to handle 25 petabytes of organization information at yahoo. In this various FILE I/O OPERATIONS AND REPLICA MANGEMENT techniques are used to handle huge volume of data. These operations & techniques are: File read and write,block placement, replication management, balancer,block scanner, Decommissioing, Inter-Cluster Data Copy. In file read and write, client has to take name node's permission for reading, writing, creating or opening new file. In block placement scheme, various nodes are distributed across various racks & these nodes communicate with each other through various switches. In replication management, the block consists of certain

number of replicas. If block is under or over replicated, to manage the memory utilization of block, name node selects replicas to remove. All actions are taken by name node. A balancer acts as a tool which balances usage on disk space in HDFS cluster. Block scanner is used to scan the block replicas to check the saved checksum and match with block data. In decommissioning, the administrator creates include and exclude list. Include list contains list of host addresses of nodes that are registered and exclude list contains list of host addresses of nodes that are not registered. Inter-cluster copy uses MapReduce framework to manage parallel scheduling of tasks and fault recognition and recovery.

Jeffrey Shafer, Scott Rixner, and Alan L. Cox et al. [3] evaluate HDFS performance and reveals different performance issues. Three problems discussed in this paper are portability assumptions, portability limitations, architectural bottlenecks.

Vinod Kumar Vavilapalli, Arun C Murthy et al.[5] discussed about present state of deployment, design and development of future generation of Hadoop's component: YARN. YARN is used to separate resource management infrastructure from programming model and assigns different scheduling functions like task fault tolerance.

Asso.Prof. Ashish Sharma, Snehlata Vyas et al. [6], proposed Hadoop framework Apache Hadoop 2 to run variety of applications. YARN in Hadoop 2 puts both job scheduling functions and resource management in different layer. Hadoop 2 gives better performance than Hadoop 1. Also this paper addresses limitations and also overcomes difficulty in Hadoop 1. This paper shows the main differences between Hadoop 1 and Hadoop 2 by showing the working of both architectures. In Hadoop 1.0 only MapReduce is present which manages all work like data processing and cluster management. Due to this MapReduce has faced many limitations. In Hadoop 2.0, the architecture divides the work of MapReduce to only data processing and cluster resource management is handled by YARN. This makes MapReduce burden free. MapReduce in Hadoop 2.0 is called as MR2.

Mrudula Varade, Vimla Jethani et al. [7], described three techniques for evenly distributing metadata over server. Hashing, sub-tree partitioning and consistent hashing are the techniques used. This paper shows the differences between all the three techniques and their implementations. And concludes that hashing is the best technique of all three in terms of performance, scalability, reliability, load balancing.

Howard Karlo__ Siddharth Suriy Sergei Vassilvitskiiz et al. [10], shows the comparison between PRAM and MapReduce. The model uses MapReduce paradigm which aims is to reduce the number of machines and restricting the total memory per machine. The main purpose is to develop a model for proficient computation over larger data sets. This paper gives definition of MapReduce programming paradigm. Definition 1. A mapper is a (possibly randomized) function that takes as input one ordered <key; value> Pair of binary strings. As output the mapper produces a finite multiset of new <key; value> pairs. It is important that the mapper operates on one <key; value> pair at a time.

Definition 2. A reducer is a (possibly random-ized) function that takes as input a binary string k which is the key, and a sequence of values v1; v2; which are also binary strings. As output, the reducer produces a multiset of pairs of binary strings <k; vk; 1>, <k; vk; 2>, <k; vk; 3> …. The key in the output tuples is identical to the key in the input tuple.

3. ZOOKEEPER

In the history every application was a single program which runs on a single PC with an only CPU. Now days, situation is changed. In Cloud Computing & Big Data world, multiple applications are running independently on different set of computers in parallel with each other. So its tedious job for developers to maintain coordination among all these independent applications. It may cause failure at certain point of time.So to avoid this problem Zookeeper was designed. Zookeeper is a robust service. With the help of Zookeeper application developers can focus only on their application logic rather than coordination. It exposes a easy API which helps developers to apply common coordination tasks like managing group membership, choosing a master server & Managing metadata. Zookeeper is an application library. It has 2 principal API implementations- C and java. It has a service component made in java that runs on an assembly of dedicated servers. Due to assembly of servers, Zookeeper can increase throughput & tolerate faults or errors [12].

[14] ZooKeeper is a building block for distributed systems.Zookeeper is a distributed coordination service. It is extremely consistent & highly available service. It is top level apache project created at yahoo. We can create distributed locks, distributed queues,group membership, master elections, distributed configuration & much more with the help of it. Various properties of zookeeper are operations are controlled, Updates are atomic & changes are robust. Zookeeper is an important part of HADOOP. Many HADOOP related projects like HBase, HDFS high availability & flume are based on it [13].

Zookeeper provides certain coordination services:

1. Name Service- This service is used to map a name to some data linked with that name. For example a telephone directory is a name service in which name of person is mapped to his or her telephone number. Another example is DNS service in which a domain name is mapped to an IP address. Zookeeper exposes easy interface to do that.
2. Locking- To make a serialized access to shared resources in distributed sys, we need distributed

mutexes. With the help of zoolkeeper, we can implement it efficiently.

Configuration management- ZooKeeper centrally manage and store the configuration of your distributed system. This means that any new nodes joining will pick up the up-to-date centralized configuration from ZooKeeper as soon as they join the system. This also allows you to centrally change the state of your distributed system by changing the centralized configuration through one of the ZooKeeper clients.

Synchronization- Zookeeper provides synchronized access between distributed systems and shared resources.

Figure 2. shows Client-Server architecture of Zookeeper.

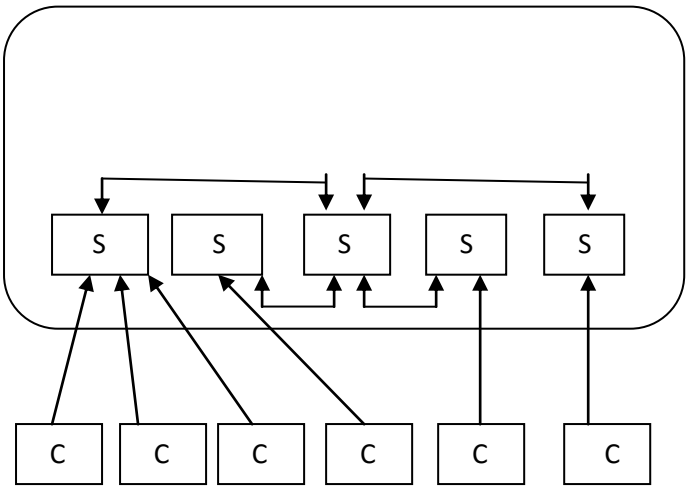

Fig -2: Client-Server architecture of Zookeeper

ZooKeeper, on its own is a distributed application. ZooKeeper follows a simple client-server model where *clients* are nodes that make use of the service, and *servers* are nodes that provide the service. A collection of ZooKeeper servers forms a ZooKeeper *ensemble.* At any given time, one ZooKeeper client is connected to one ZooKeeper server. Each ZooKeeper server can handle a large number of client connections at the same time. Each client periodically sends pings to the ZooKeeper server it is connected to let it know that it is alive and connected. The ZooKeeper server in question responds with an acknowledgment of the ping, indicating the server is alive as well. When the client doesn't receive an acknowledgment from the server within the specified time, the client connects to another server in the ensemble, and the client session is transparently transferred over to the new ZooKeeper server.

3.1 Implementation

Zookeeper components shows the service of zookeeper [15].

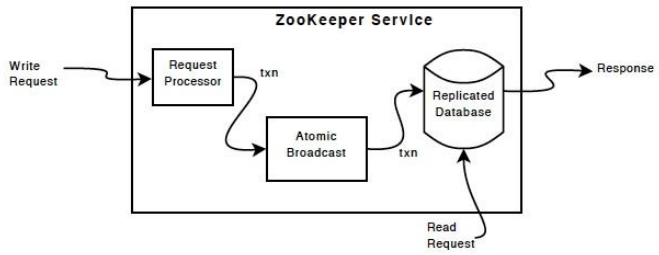

Fig -2: Zookeeper Components

Every ZooKeeper server services clients. Clients connect to exactly one server to submit irequests. Read requests are serviced from the local replica of each server database. Requests that change the state of the service, write requests, are processed by an agreement protocol.
ZooKeeper uses a custom atomic messaging protocol. Since the messaging layer is atomic, ZooKeeper can guarantee that the local replicas never diverge. When the leader receives a write request, it calculates what the state of the system is when the write is to be applied and transforms this into a transaction that captures this new state.

4. FULME

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.

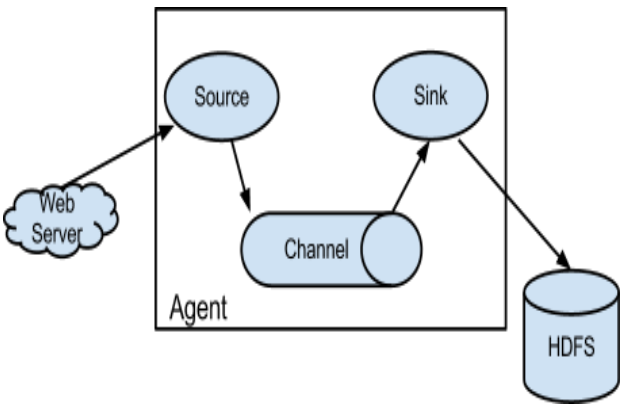

Fig -3: Flume service

A source can be any data source, and Flume has many predefined source adapters. A sink is the target of a specific operation (and in Flume, among other paradigms that use this term, the sink of one operation can be the source for the next downstream operation). A decorator is an operation on the stream that can transform the stream in some manner, which could be to compress or uncompress data, modify data by adding or removing pieces of information.

Three types of sinks in Flume-

1. Collector Tier Event - This is where you would land a flow (or possibly multiple flows joined together) into an HDFS-formatted file system.
2. Agent Tier Event - This is used when you want the sink to be the input source for another operation. When you use these sinks, Flume will also ensure the integrity of the flow by sending back acknowledgments that data has actually arrived at the sink.
3. Basic – This sink can be a text file, the console display, a simple HDFS path, or a null bucket where the data is simply deleted.

The purpose of Flume is to provide a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

4.1. Flow Pipeline

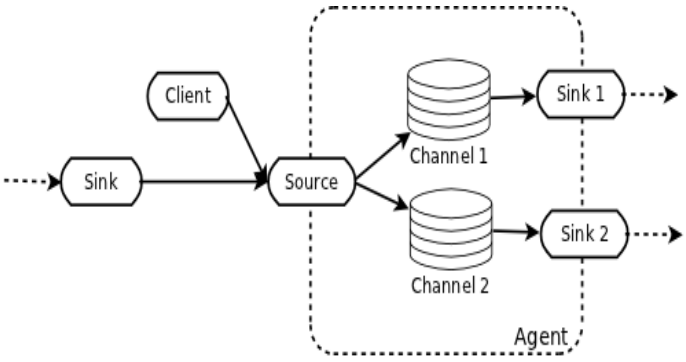

Fig -4: Schematic showing logical components in a flow [18]

The arrows represent the direction in which events travel across the system. This also shows how flows can spread out by having one source write the event out to multiple channels.

A flow in Flume NG starts from the client. The client transmits the event to it's next hop destination. This destination is an agent. More precisely, the destination is a source operating within the agent. The source receiving this event will then deliver it to one or more channels. The channels that receive the event are drained by one or more sinks operating within the same agent. If the sink is a regular sink, it will forward the event to it's next-hop destination which will be another agent. If instead it is a terminal sink, it will forward the event to it's final destination. Channels allow the decoupling of sources from sinks using the familiar producer-consumer model of data exchange. This allows sources and sinks to have different performance and runtime characteristics and yet be able to effectively use the physical resources available to the system.

By configuring a source to deliver the event to more than one channel, flows can fan-out to more than one destination. This is illustrated in Figure 1 where the source within the operating Agent writes the event out to two channels – Channel 1 and Channel 2.

3. CONCLUSIONS

In this paper we discussed about the architectural framework of Hadoop and to overcome the faults tolerance in the HDFS that includes checkpoint and recovery, data duplication. The Hadoop Distributed File System (HDFS) is intended to store very huge data sets consistently, and to stream those data sets at high bandwidth to user applications.

ZooKeeper implement synchronization primitives, coordinates distributed systems

The purpose of Flume is to provide a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

REFERENCES

[1] Vishal S Patil, Pravin D. Soni, *HADOOP SKELETON & FAULT TOLERANCE IN HADOOP CLUSTERS,* International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 2, Issue 2, February 2013

[2] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, *The Hadoop Distributed File System,* http://pages.cs.wisc.edu/~akella/CS838/F15/838-CloudPapers/hdfs.pdf

[3] Jeffrey Shafer, Scott Rixner, and Alan L. Cox, *The Hadoop Distributed Filesystem: Balancing Portability and Performance,*www.jeffshafer.com/publications/papers/shafer_ispass10.pdf

[4] Dhruba Borthakur, *HDFS Architecture Guide.*

[5] Vinod Kumar Vavilapallih Arun C Murthyh Chris Douglasm Sharad Agarwali, *Apache Hadoop YARN: Yet Another Resource Negotiator,* https://www.sics.se/~amir/files/download/dic/2013%20-%20Apache%20Hadoop%20YARN:%20Yet%20Another%20Resource%20Negotiator%20(SoCC).pdf

[6] Ashish Sharma, Snehlata Vyas, *Hadoop2 Yarn*, IPASJ International Journal of Computer Science (IIJCS), Volume 3, Issue 9, September 2015

[7] Mrudula varade, Vimla Jethani, *Distributed MetaData Management Scheme in HDFS,* International Journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013.

[8] Harshawardhan S. Bhosale, Prof. Devendra P. Gadekar, *A Review Paper on Big Data and Hadoop,* International Journal of Scientific and Research Publications, Volume 4, Issue 10, October 2014

[9] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simpli_ed Data Processing on Large Clusters,* http://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

[10] Howard Karlo__ Siddharth Suriy Sergei Vassilvitskiiz, *A Model of Computation for MapReduce,* theory.stanford.edu/~sergei/papers/soda10-mrc.pdf

11. Anuradha G. Khade, Prof. Y. B. Gurav, *Big Data Analytics for Advertisement Promotion,* International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 12, December 2014

[12]www.safaribooksonline.com/library/view/zookeeper/9781449361297/ch01.html

[13]http://www.tomwheeler.com/publications/2012/zookeeper_tomwheeler_ll-20120607.pdf

[14] http://www.ibm.com/developerworks/library/bd-zookeeper/

[15] Hadoop wiki - powered by. http://wiki.apache.org/hadoop/PoweredBy.

[16]http://zookeeper.apache.org/doc/trunk/zookeeperOver.html

[17]http://www01.ibm.com/support/knowledgecenter/SSPT3X_1.3.0/com.ibm.swg.im.infosphere.biginsights.doc/doc/c0057868.html

[18]https://www01.ibm.com/software/data/infosphere/hadoop/flume/

[19]http://blog.cloudera.com/blog/2011/12/apache-flume-architecture-of-flume-ng-2/

BIOGRAPHIES

Mrs. Smita S. Konda working as Asst. Professor in AGPIT, Solapur. She has completed her BE in INFORMATION TECHNOLOGY from WALCHAND INSTITUTE OF TECHNOLOGY, SOLAPUR from Solapur University. She has completed her M.Tech in Computer Science and Engineering from JNTU University,Hyderabad. She has 5 years of teaching experience.

Ms. Rohini S. More working as Asst Professor in AGPIT, Solapur. She has completed her BE in Computer Science and Engineering from Bharat Ratna Indira Gandhi College of Engineering, Solapur from Solapur University. She has completed her M.Tech in Computer Science and Engineering from JNTU University, Hyderabad. She has 3.5 years of teaching experience.