Hadoop: Understanding the Big Data Processing Method

Deepak Chandra Upreti[1], Pawan Sharma[2], Dr. Yaduvir Singh[3]

[1]PG Student, Department of Computer Science & Engineering, Ideal Institute of Technology

[2]Assistant Professor, Department of Computer Science & Engineering, Ideal Institute of Technology, Ghaziabad

[3]Professor, Department of Computer Science & Engineering, Ideal Institute of Technology, UPTU, Lucknow, India

Abstract: ***"Every day, we create 2.5 quintillion bytes of data so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. This data is "big data." Big data requires different approaches: Techniques, tools, architecture and data processing methods. The main focus of the paper is to draw the state-of-the-art techniques and technologies for Big Data*** **processing** ***with the help of Big Data application- Hadoop***

Keywords: Apache Hadoop, big data, Java, Google File System, Google MapReduce, open source, MapR, Oracle, distributed file system, HDFS, redundant array of inexpensive disks, replication factor, NameNode, DataNode, MapReduce, JobTracker, TaskTracke,r yet another resource negotiator, PigLatin, HiveQL, Hbase

1. Introduction

Big Data has gained much attention from the academia and the IT industry. In the digital and computing world, information is generated and collected at a rate that rapidly exceeds the boundary range. Currently, over 2 billion people worldwide are connected to the Internet, and over 5 billion individuals own mobile phones. By 2020, 50 billion devices are expected to be connected to the Internet. At this point, predicted data production will be 44 times greater than that in 2009. As information is transferred and shared at light speed on optic fiber and wireless networks, the volume of data and the speed of market growth increase. However, the fast growth rate of such large data generates numerous challenges, such as the rapid growth of data, transfer speed, diverse data, and security. This research direction facilitates the exploration of the domain and the development of optimal techniques to address Big Data.

Hadoop is a powerful technology, but it is just one component of the big data technology landscape. Hadoop is designed for specific data types and workloads. For example, it is a very cost-effective technology for staging large amounts of raw data, both structured and unstructured, which can then be refined and prepared for analytics. Hadoop can also help you avoid costly upgrades of existing proprietary databases and data warehouse appliances when their capacity is being consumed too quickly with raw, unused data and extract-load-transform processing.

Apache Hadoop3 is a Java-based open source platform that makes processing of large data possible over thousands of distributed nodes. Hadoop's development resulted from public- ation of two Google authored whitepapers (i.e., Google File System and Google Map –Reduce

2. Hadoop

Hadoop is a distributed file system and data processing engine that is designed to handle extremely high volumes of data in any structure. Hadoop focus in on supporting redundancy, distributed architectures, and parallel processing. It is an open-source software framework from Apache Inspired by

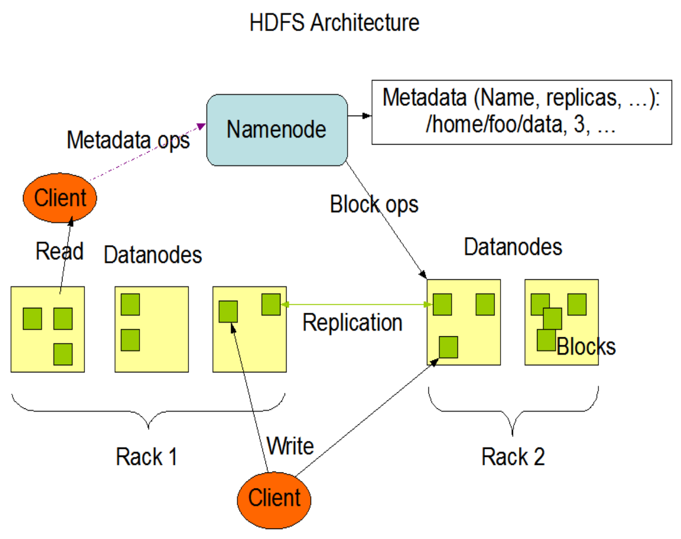
- GFS (Google File System)
- Google MapReduce

Hadoop has two base components

- The Hadoop distributed file system (HDFS), Which supports data in structured relational form, in unstructured form, and in any form in between
- The MapReduce programing paradigm for Managing applications on multiple distributed servers

3. Hadoop Distributed File System (HDFS)

Hadoop distributed file system (HDFS) is a base component of the Hadoop framework that manages the data storage.It stores data in the form of data blocks (default size: 64MB) on the local hard disk. The input data size defines the block size to be used in the cluster. A block size of 128MB for a large file set is a good choice. Keeping large block sizes means a small number of blocks can be stored, thereby minimizing the memory requirement on the master node (commonly known as NameNode) for storing the metadata information. Block size also impacts the job execution time, and thereby cluster performance. A file can be stored in HDFS with a different block size during the upload process. For file sizes smaller than the block size, the Hadoop cluster may not perform optimally.



Functions of a NameNode

a) Manages File System Namespace
- Maps a file name to a set of blocks
- Maps a block to the DataNodes where it resides

b) Cluster Configuration Management

c) Replication Engine for Blocks

NameNode Metadata

a) Metadata in Memory
- The entire metadata is in main memory
- No demand paging of metadata

b) Types of metadata
- List of files
- List of Blocks for each file
- List of DataNodes for each block
- File attributes, e.g. creation time, replication factor

c) A Transaction Log
- Records file creations, file deletions etc

NameNode Failure

a) A single point of failure

b) Transaction Log stored in multiple directories
- A directory on the local file system
- A directory on a remote file system (NFS/CIFS)

c) Need to develop a real high availability (HA) solution

Data Node

a) A Block Server
- Stores data in the local file system (e.g. ext3)
- Stores metadata of a block (e.g. CRC)
- Serves data and metadata to Clients

b) Block Report
- Periodically sends a report of all existing blocks to the NameNode

c) Facilitates Pipelining of Data
- Forwards data to other specified DataNodes

Block Placement

a) Current Strategy
- One replica on local node
- Second replica on a remote rack
- Third replica on same remote rack
- Additional replicas are randomly placed

b) Clients read from nearest replicas

c) Would like to make this policy pluggable

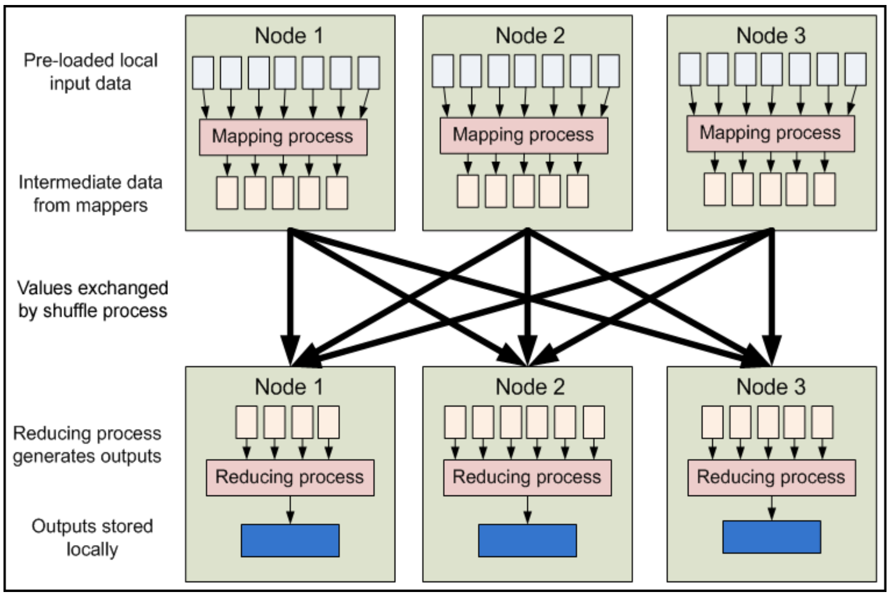
Replication Engine

a) NameNode detects DataNode failures
- Chooses new DataNodes for new replicas
- Balances disk usage
- Balances communication traffic to DataNodes

4. MapReduce

MapReduce – Dataflow

MapReduce is the second base component of a Hadoop framework. It manages the data processing part on the Hadoop cluster. A cluster runs MapReduce jobs written in Java or any other language (Python, Ruby, R, etc.) via streaming. A single job may consist of multiple tasks. The number of tasks that can run on a data node is governed by the amount of memory installed. It is recommended to have more memory installed on each data node for better cluster performance. Based on the application workload that is going to run on the cluster, a data node may require high compute power, high disk I/O or additional memory.


Pre-loaded local input data
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Outputs stored locally
Node 1
Node 2
Node 3
Mapping process
Reducing process

MapReduce – Features

a) Fine grained Map and Reduce tasks
- Improved load balancing
- Faster recovery from failed tasks

b) Automatic re-execution on failure
- In a large cluster, some nodes are always slow or flaky
- Framework re-executes failed tasks

c) Locality optimizations
- With large data, bandwidth to data is a problem
- Map-Reduce + HDFS is a very effective solution
- Map-Reduce queries HDFS for locations of input data
- Map tasks are scheduled close to the inputs when possible

1. Hadoop: A Big Data Processing Method

Hadoop/MapReduce technology

a) Learn the platform (how it is designed and works)
- How big data are managed in a scalable efficient way

b) Learn writing Hadoop jobs in different languages
- Programming Languages: Java, C, Python
- High-Level Languages: Apache Pig, Hive

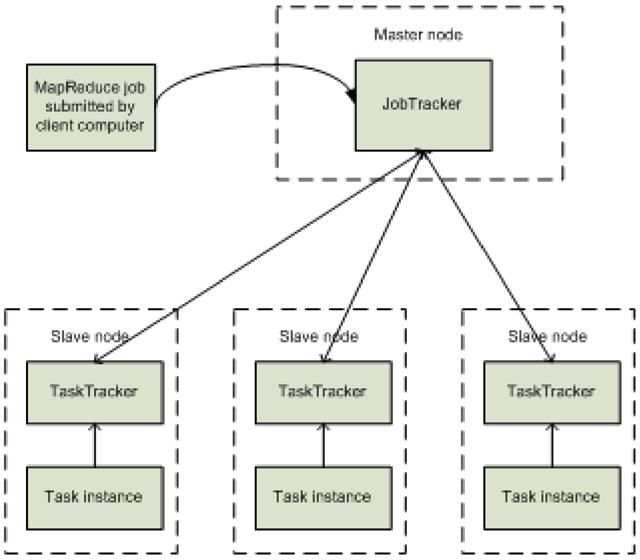
c) Learn advanced analytics tools on top of Hadoop
- RHadoop: Statistical tools for managing big data
- Mahout: Data mining and machine learning tools over big data.

d) Learn state-of-art technology from recent research papers
- Optimization, indexing techniques, and Other extensions to Hadoop

Hadoop Services Schematic

The job execution process is handled by the JobTracker and TaskTracker services. The type of

MapReduce: High Level



Nodes, Trackers, Tasks
- **Master node runs *JobTracker* instance, which accepts *Job* requests from clients**
- ***TaskTracker* instances run on slave nodes**
- **TaskTracker forks separate Java process for task instances**

Job Distribution
- MapReduce programs are contained in a Java "jar" file + an XML file containing serialized program configuration options
- Running a MapReduce job places these files into the HDFS and notifies TaskTrackers where to retrieve the relevant program code
- … Where's the data distribution?

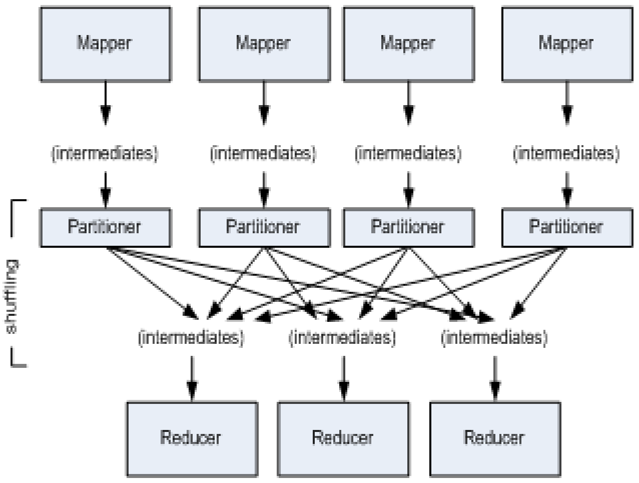
job scheduler chosen for a cluster depends on the application workload. If the cluster will be running short running jobs, then FIFO scheduler will serve the purpose. For a much more balanced cluster, one can choose to use Fair Scheduler. If the node running JobTracker service fails, all jobs submitted in the cluster will be discontinued and must be resubmitted once the node is recovered or another node is made available.

Data Distribution

a) **Implicit in design of MapReduce!**
   - **All mappers are equivalent; so map whatever data is local to a particular node in HDFS**

b) **If lots of data does happen to pile up on the same node, nearby nodes will map**

Partition And Shuffle



1. Example Program - Wordcount

a) **map()**
   - **Receives a chunk of text**
   - **Outputs a set of word/count pairs**

b) **reduce()**
   - **Receives a key and all its associated values**
   - **Outputs the key and the sum of the values**

```
package org.myorg;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {
```

Wordcount – main( )

```
public static void main(String[] args) throws Exception {
JobConf conf = new JobConf(WordCount.class);
conf.setJobName("wordcount");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(Map.class);
conf.setReducerClass(Reduce.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
JobClient.runJob(conf);
}
```

Wordcount – map( )

```
public static class Map extends MapReduceBase … {
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, …) … {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
output.collect(word, one);
}
}
}
```

Wordcount – reduce( )

```
public static class Reduce extends MapReduceBase … {
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, …) … {
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
}
output.collect(key, new IntWritable(sum));
}
}
}
```

2. Using Hadoop Distributed File System (HDFS)

- Can access HDFS through various shell commands (see Further Resources slide for link to documentation)

a) **hadoop–put <localsrc> … <dst>**
b) **hadoop –get <src> <localdst>**
c) **hadoop –ls**
d) **hdoop –rm file**

Running a Hadoop Job

a) **Place input file into HDFS:**
   - **hadoop fs –put ./input-file input- file**

b) **Run either normal or streaming version:**
   - **hadoop jar Wordcount.jar org.myorg.Wordcount input-file output- file**
   - **hadoop jar hadoop-streaming.jar**

**~ input input-file **
**~ output output-file **
**~ file Streaming_Mapper.py **
**~ mapper python Streaming_Mapper.py **
**~ file Streaming_Reducer.py **
**~ reducer python Streaming_Reducer.py **

Related Apache Sub-Projects

The Hadoop Ecosystem

Hadoop base components are surrounded by numerous ecosystem projects. Many of them are already top-level Apache (open source) projects; some, however, remain in the incubation stage. Here we highlights a few of the ecosys-tem components. The most common ones include

- **Pig** – High-level language for data analysis
- **HBase** – Table storage for semi-structured data

- **Zookeeper** – Coordinating distributed applications
- **Hive** – SQL-like Query language and Metastore
- **Mahout** – Machine learning

5. Conclusion

This paper presents one of the Big Data processing methods called *Hadoop*. This concept includes the increase in data, the progressive demand for data processing, and the role of Big Data in the current environment of enterprise and technology. To enhance the efficiency of data processing, we have devised a hadoop method that uses the technologies and terminologies of Big Data. The stages in this technology include basics, HDFC, MapReduce and Related Apache sub-projects. All these stages are important for big data processing method i.e. hadoop. Information is simultaneously increasing at an exponential rate, but information processing methods are improving relatively slowly. Currently, a limited number of tools are available to completely address the issues in BigData analysis. The state-of-the-art techniques and technologies in many important Big Data applications (i.e., Hadoop, Hbase, and Cassandra) cannot solve the real problems of storage,

References

[1] Big Data Technology
[2] Harnessing-Hadoop
[3] Considerations for Big Data: Architecture and Approach by Kapil Bakshi – Paper published in IEEE
[4] Big Data Analytics by Sachidanand Singh – Paper published in 2012 International Conference on Communication Information & Computing Technology (ICCICT), Oct. 19-20, Mumbai, India
[5] McKinsey Global Institute
[6] http://hadoop.apache.org/
[7] S. Kaisler, F. Armour, J. A. Espinosa, and W.Money, “Big data: issues and challe- nges moving forward,” in *Proceedings of the IEEE 46th Annual Hawaii Internation- al Conference on System Sciences (HICSS ’13)*, pp.995–1004, January 2013.