IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-5, May 2018 E-ISSN: 2347-2693

A Novel Location Awareing Mapreducing Techniques Using Big Data Applications

Bhavani Buthukuri [1*], M E Purushoththaman[2]

[1]Department of CSE, Shri JagadishPrasad Jabarmal Tibrewala University, Jhun Jhunu, INDIA
[2]Department of Computer Science and Engineering, JNTUH, INDIA

[*]*Corresponding Author:* mepurushoththaman@yahoo.com , *Tel.: +91-99596 16498*

Available online at: www.ijcseonline.org

Accepted: 15/May/2018, Published: 31/May/2018

Abstract— There is a growing trend of applications that should handle big data. However, analyzing big data is a very challenging problem today. For such applications, the MapReduce framework has recently attracted a lot of attention. Google's MapReduce or its open-source equivalent Hadoop is a powerful tool for building such applications. In this paper, we will discuss the MapReduce framework based on Hadoop, and how to design efficient MapReduce algorithms and present the state-of-the-art in MapReduce algorithms for data mining, machine learning, query processing, data analysis and similarity joins. The intended audience of this tutorial is professionals who plan to design and develop MapReduce algorithms and researchers who aware of the state-of-the-art in MapReduce algorithms available today for big data analysis.

Keywords— Map Reduce Framework, Hadoop, Data Mining**,** Query Processing, Data Analysis,

I. INTRODUCTION

Due to popular of online social network, huge amount of data are generating and processing, these data with existing techniques is not efficient and these large amount data is called big data. Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. For example, the volume of data in Face book or YouTube are required to collect and manage on a daily basis, can fall under the category of Big Data. However, Big Data is not only about scale and volume, it also involves one or more of the following aspects − Velocity, Variety, Volume, and Complexity.

Online data, streaming data, are becoming more and more essential in online social networks and organization applications. Applications facilitate the fact that data are acquired immediately electronically and then often made accessible to other enterprise information systems. An example is the usage of sensor data (like e.g. GPS coordinates) providing context information about a user. This information is used in all kinds of context aware information systems, e.g. location based services.

In Twitter, tweets are streamed continuously and in order to process real stream data required efficient workflow transformation. Sensor information (i.e., context information) is continuously acquired and published. Information systems have to continuously process this information. That is, after a specified number of information has been accumulated, the available information is processed. Workflows, where the availability of data, coordinates (i.e., controls) the processing in the workflow are called data driven workflows. Classical business workflows are coordinated by interactions with humans or other information systems (called control flow driven) and terminate after a case is complete. Processing of continuous sensor data (i.e., streaming data), however, does not terminate without user interaction, since a stream is per definition infinite.

An example of a data driven workflow is an online navigation system providing additional location based services to the driver of a car, like the availability of gas stations, rest rooms, weather forecasts, or accommodations. All of the above mentioned location based services rely on the GPS coordinates. The GPS coordinates provide the context for the various location based services. Each location based service can be represented by a single data driven workflow.

II. BIG DATA

Big data is an evolving term that describes any voluminous amount of structured, semi-structured and unstructured data that has the potential to be mined for information. Although big data doesn't refer to any specific quantity, the term is often used when speaking about Petabytes and Exabytes of data.

Big data is used to describe a massive volume of data that is so large that it's difficult to process. The data is too big that exceeds current processing capacity.

Big data can be characterized by 3Vs: the extreme volume of data, the wide variety of types of data and the velocity at which the data must be must processed.

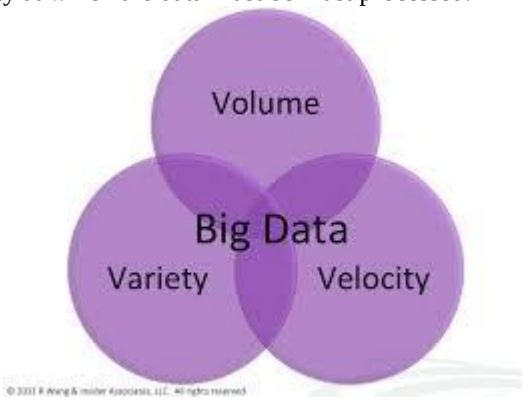

Fig.1:Big Data 3V's

An example of big data might be Petabytes (1,024 terabytes) or Exabyte's (1,024 Petabytes) of data consisting of billions to trillions of records.

E.g. Web, sales, customer contact center, social media, and mobile data.

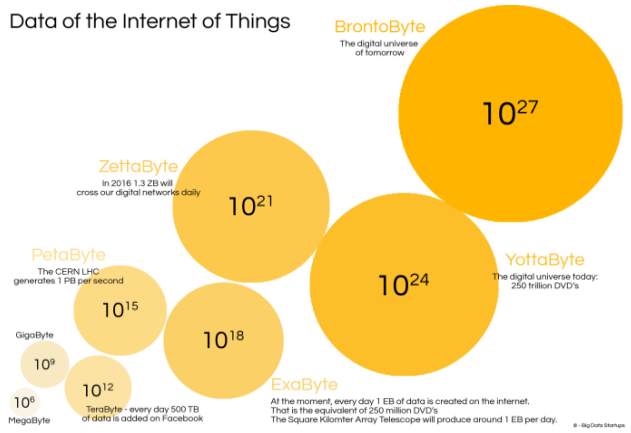

Fig 2. Data Measurements

III. APACHE HADOOP:

Fig.2:Hadoop-logo

Hadoop is an open-source software framework for storing and processing big data in a distributed fashion on large clusters of commodity hardware. Essentially, it accomplishes two tasks: massive data storage and faster processing.

Doug Cutting, Cloudera's Chief Architect, helped create Apache Hadoop out of necessity as data from the web exploded and grew far beyond the ability of traditional systems to handle it. Hadoop was initially inspired by papers published by Google outlining its approach to handling an avalanche of data, and has since become the de facto standard for storing, processing and analyzing hundreds of terabytes, and even petabytes of data.

Why is Hadoop important?
Since its inception, Hadoop has become one of the most talked about technologies. Why? One of the top reasons (and why it was invented) is its ability to handle huge amounts of data – any kind of data – quickly. With volumes and varieties of data growing each day, especially from social media and automated sensors, that's a key consideration for most organizations. Other reasons include:

Low cost. The open-source framework is free and uses commodity hardware to store large quantities of data.

*Computing power***.** Its distributed computing model can quickly process very large volumes of data. The more computing nodes you use, the more processing power you have.

Scalability. You can easily grow your system simply by adding more nodes. Little administration is required.

*Storage flexibility***.** Unlike traditional relational databases, you don't have to preprocess data before storing it. And that includes unstructured data like text, images and videos. You can store as much data as you want and decide how to use it later.

Inherent data protection and self-healing capabilities.
Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. And it automatically stores multiple copies of all data.

The main components of Hadoop are:

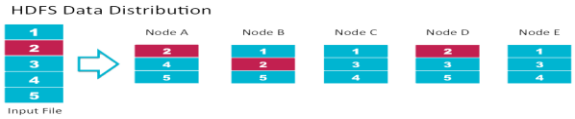

Fig 3. HDFS Data Distribution

Data in HDFS is replicated across multiple nodes for compute performance and data protection.

IV. MAPREDUCE

MapReduce is a massively scalable, parallel processing framework that works in tandem with HDFS. With MapReduce and Hadoop, compute is executed at the location of the data, rather than moving data to the compute location; data storage and computation coexist on the same physical nodes in the cluster. MapReduce processes exceedingly large amounts of data without being affected by traditional bottlenecks like network bandwidth by taking advantage of this data proximity.

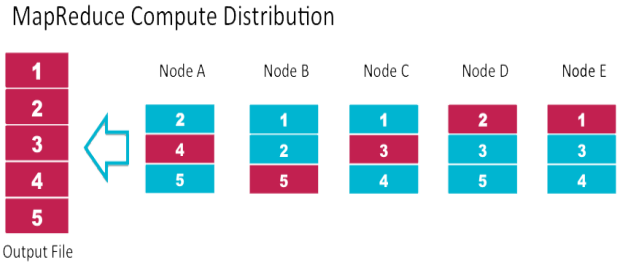

Fig 4. MapReduce Compute Distribution

MapReduce divides workloads up into multiple tasks that can be executed in parallel.

The Map Reduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and value classes have to be serializable by the framework and hence need to implement the Writable interface. Additionally, the key classes have to implement the WritableComparable interface to facilitate sorting by the framework.

Input and Output types of a MapReduce job:

(input) <k1, v1> -> map -> <k2, v2> -> combine -> <k2, v2> -> reduce -> <k3, v3> (output)

Hadoop Ecosystems:

The Hadoop platform consists of two key services: a reliable, distributed file system called Hadoop Distributed File System (HDFS) and the high-performance parallel data processing engine called Hadoop MapReduce, described in MapReduce below.

The combination of HDFS and MapReduce provides a software framework for processing vast amounts of data in parallel on large clusters of commodity hardware (potentially scaling to thousands of nodes) in a reliable, fault-tolerant manner. Hadoop is a generic processing framework designed to execute queries and other batch read operations against massive datasets that can scale from tens of terabytes to petabytes in size.

The popularity of Hadoop has grown in the last few years, because it meets the needs of many organizations for flexible data analysis capabilities with an unmatched price-performance curve. The flexible data analysis features apply to data in a variety of formats, from unstructured data, such as raw text, to semi-structured data, such as logs, to structured data with a fixed schema.

Hadoop has been particularly useful in environments where massive server farms are used to collect data from a variety of sources. Hadoop is able to process parallel queries as big, background batch jobs on the same server farm. This saves the user from having to acquire additional hardware for a traditional database system to process the data (assume such a system can scale to the required size). Hadoop also reduces the effort and time required to load data into another system; you can process it directly within Hadoop. This overhead becomes impractical in very large data sets.

Many of the ideas behind the open source Hadoop project originated from the Internet search community, most notably Google and Yahoo!. Search engines employ massive farms of inexpensive servers that crawl the Internet retrieving Web pages into local clusters where they are analyzes with massive, parallel queries to build search indices and other useful data structures.

The Hadoop ecosystem includes other tools to address particular needs. Hive is a SQL dialect and Pig is a dataflow language for that hide the tedium of creating MapReduce jobs behind higher-level abstractions more appropriate for user goals. Zookeeper is used for federating services and Oozie is a scheduling system. Avro, Thrift and Protobuf are platform-portable data serialization and description formats.

A. MapReduce

MapReduce is now the most widely-used, general-purpose computing model and runtime system for distributed data analytics. It provides a flexible and scalable foundation for analytics, from traditional reporting to leading-edge machine learning algorithms. In the MapReduce model, a compute "job" is decomposed into smaller "tasks" (which correspond to separate Java Virtual Machine (JVM) processes in the Hadoop implementation). The tasks are distributed around the cluster to parallelize and balance the load as much as possible. The MapReduce runtime infrastructure coordinates the tasks, re-running any that fail or appear to hang. Users of MapReduce don't need to implement parallelism or reliability features themselves. Instead, they focus on the data problem they are trying to solve.

B. Pig

Pig is a platform for constructing data flows for extract, transform, and load (ETL) processing and analysis of

large datasets. Pig Latin, the programming language for Pig provides common data manipulation operations, such as grouping, joining, and filtering. Pig generates Hadoop MapReduce jobs to perform the data flows. This high-level language for ad hoc analysis allows developers to inspect HDFS stored data without the need to learn the complexities of the MapReduce framework, thus simplifying the access to the data.

The Pig Latin scripting language is not only a higher-level data flow language but also has operators similar to SQL (e.g., FILTER and JOIN) that are translated into a series of map and reduce functions. Pig Latin, in essence, is designed to fill the gap between the declarative style of SQL and the low-level procedural style of MapReduce.

C. Hive

Hive is a SQL-based data warehouse system for Hadoop that facilitates data summarization, ad hoc queries, and the analysis of large datasets stored in Hadoop-compatible file systems (e.g., HDFS, MapR-FS, and S3) and some NoSQL databases. Hive is not a relational database, but a query engine that supports the parts of SQL specific to querying data, with some additional support for writing new tables or files, but not updating individual records. That is, Hive jobs are optimized for scalability, i.e., computing over all rows, but not latency, i.e., when you just want a few rows returned and you want the results returned quickly. Hive's SQL dialect is called HiveQL. Table schema can be defined that reflect the data in the underlying files or data stores and SQL queries can be written against that data. Queries are translated to MapReduce jobs to exploit the scalability of MapReduce. Hive also support custom extensions written in Java, including user-defined functions (UDFs) and serializer-deserializers for reading and optionally writing custom formats, e.g., JSON and XML dialects. Hence, analysts have tremendous flexibility in working with data from many sources and in many different formats, with minimal need for complex ETL processes to transform data into more restrictive formats. Contrast with Shark and Impala.

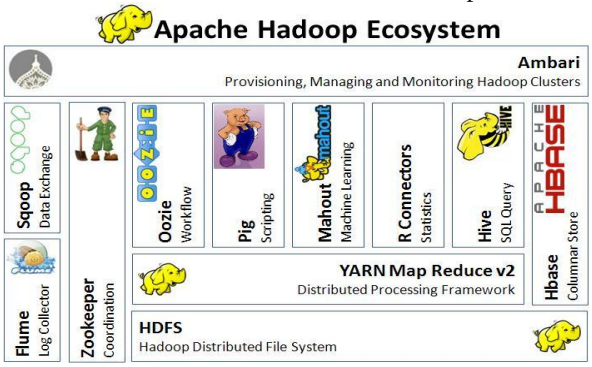

Fig 5. Apache Hadoop Ecosystem

V. PROBLEM STATEMENT

Continuous stream data processing and workflows have been examined in different field. Business workflows are quite different from continuous stream data processing.

Among many workflow models the neighboring workflow model is scientific workflow models, Dataflow process networks are stream based data processing approaches forming the basis for many scientific workflow systems, like e.g. Kepler or Taverna.

A dataflow process network specifies that each stream element is read at most once from an input stream, the read data is transformed, and new data is produced.

The control flow of a data flow process network is controlled by the consuming activity via a data pull. The activities in the scientific workflow performing the actual data transformation are not restricted.

In this generic model information required to perform an activity is buffered by an activity itself. Keeping so much information implicit in an activity makes data sharing between different workflows difficult, since workflow optimization does not have access to implicit buffer mechanisms.

A. SCOPE

This research work is carried out in a fixed number of data size, no. of nodes and Hadoop configuration parameter.

No. of Nodes

Experimental results carried out in single Node as well up to four nodes, including one master node and three slave modes which are physically distributed.

No. of Data Size In this research the workload for all Hadoop jobs has linearly increased from 1 GB Data 2 GB Data up to 3 GB all the experiments are carried out .

No. of Hadoop Jobs Experiment results carried out using below list of Hadoop jobs (BENCHMARK SUITE)

- Pi
- TeraGen
- TeraSort
- TeraValidate
- Word Count
- TestDFSIO –Read T
- estDFIO –Write

No. of Hadoop Configuration Parameters Experiment results carried by customizing default Hadoop Configuration Parameters setting. Hadoop job Performance analysis,

evaluation and tuning done through the customization of below Hadoop Configuration Parameter values.

- mapreduce.output.file output format. compress
- mapreduce. output. file output format. compress. codec
- mapreduce.map.output.compress.
- mapreduce.map.output.compress.codec
- mapreduce.reduce.speculative dfs. Block size
- mapreduce.task.io.sort.factor

LIMITATIONS

- Continuous data streams have become crucial requirement for many scientific and industrial applications
- .Computation and communication on geo distributed data centers are cost effective.
- In streaming workflow for the short life cycle of data stream is failure. Workflow scheduling requires high performance computing resources

OBJECTIVE

1. To study the data work flow transformation for continues tweet stream
2. To analyze the elements of reducing cost for data transformation and continuous tweet stream
3. To detect the situation for optimizing tweet stream work flow Transformation
4. To estimate latency for workflow creation and execution
5. To find Tweet stream workflow describes which tweets are generated between which activities. The aim of the tweet stream workflow approach is to process tweet and therefore creating continuous tweet steam workflow transformation is the main objective.

VI. RESEARCH METHODOLOGY

Twitter continuous tweets constitute a rich resource that can be used for discovering useful knowledge about any topic that you can think of. This tweet summery information can be used in different use cases such as finding trends related to a specific keyword, measuring brand sentiment, and gathering feedback about new products and services.
This is a short guide on using Tweet Streaming to analyze social network data. Generating a tweet streaming workflow that is useful for fetching Twitter data in real-time, and also clustering the tweets based on their text and location, using the k-means algorithm. Tweet stream workflow facilitates us to rapidly and easily Discover tweet information from continuous stream tweet.

Process for Generating Tweet stream workflow

First, we need to read Twitter data from large dataset. However, unlike conventional technique, we do not actually save the Tweets on our disk or into a database. Instead, we clean, analyze and visualize it all in real-time. So in this case, we won't have to face scalability issues – we can read data for hours and continue to visualize it on a map.

- *Scraper:* it will help to Read data from Twitter and it is input to the tweet stream workflow.

During workflow execution: input tweet data is consumed and new tweet data sets are created. For large-scale computational science simulations, runtime monitoring is critically important: intermediate data sets and special provenance information are often displayed on a web-based monitoring "dashboard" to inform the scientist about progress and possible problems during execution. Depending on this information, the scientist may decide to abort a simulation or workflow run.

What is Real Time Streaming Data

Streaming Data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers.

This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows, and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling. Information derived from such analysis gives companies visibility into many aspects of their business and customer activity such as –service usage (for metering/billing), server activity, website clicks, and geo-location of devices, people, and physical goods –and enables them to respond promptly to emerging situations. For example, businesses can track changes in public sentiment on their brands and products by continuously analyzing social media streams, and respond in a timely fashion as the necessity arises.

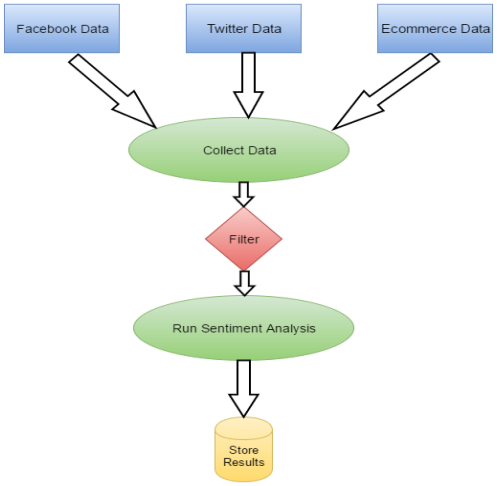

Fig 7: Stream data workflow process

Collect data from different sources such as Twitter, Facebook, ecommerce sites. On basis of some keyword such as "High Throughput" , We have to filter data. Generate sentiment of each message coming through various sources.Have a Storage mechanism for storing the processed data.

Analyzer

Hadoop generates a job history log after job execution. Hadoop Job History provides various Job Counters, File System Counters and Map-Reduce Framework and used parameter Configuration.

The analyzer is use of analysis of Hadoop Job History Logs and Hadoop configuration parameter. For analysis purposes of Hadoop Job two things CPU Utilization and Throughput (MB) should be calculated. 1. Analyzer performs analysis of both Hadoop job history log as well as configuration parameters of executing Hadoop jobs.
2. Analyzer receives default configuration parameter.
3. After analysis analyzer calculate CPU Utilization (%) and Throughput(MB) based on Hadoop Job history log CPU Utilization % = (Total CPU times in Second / Execution time (Seconds)) * 100 Throughput (MB) = (Input Bytes / Execution time (Seconds)) / 1024/102

VII. RESULTS AND DISCUSSIONS

a. MapReduce Workflow

MapReduce workflow,he framework will split the input into segments,passing each segment to a different machine. Each machine then runs the map script on the portion of data attributed to it. The map script takes some input data, and maps it to <key, value> pairs according to your specifications. For example, if w ant to count word frequencies in a text, we'd have <wor d, count> be our <key, value> pairs. Our map script, then, would emit a pair for each word in the input stream. Note that the map script does no aggregation this is what thereduce script it for. The purpose of the map script is to model the data into pairs for the reducer to aggreg ate. Emitted pairs are then shuffledwhich basically mean s that pairs with the same key are grouped and passed t o a single machine,which will then run the reduce script over them
The reduce script takes a collection of pairs and"reduce s" them according to the user-specified reduce script.

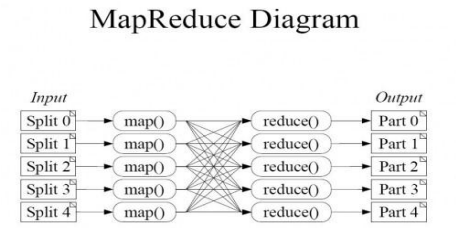

Fig 8: MapReduce work flow

Input Split Phase: The Input split depends upon the type of the input format. For example if we are using a text data, then we will be having our input of type Text Input Format. The input format splits the data line by line and each line which splitted is assigned to a mapper. This data is stored based upon the key value pairs. The Key here is the byte offset address which is of object type and each line will be the value here, excluding the line terminators. There are also different input formats like Key Value Text Input Format, Sequence File Input Format, Sequence File As Text Input Format, Sequence File As Binary Input Format, etc.[6]

Algorithm:MapReducing Algorithm

1. Begin.

2. **function** Map **is**
3. **input: integer** K1 between 1 and 1100, representing a batch of 1 million social.person records
4. **for each** social.person record in the K1 batch **do**
5. **let** Y be the person's age
6. **let** N be the number of contacts the person has
7. **produce one output record** (Y,(N,1))
8. **repeat**
9. **end function**
10. **function** Reduce **is**
11. **input:** age (in years) Y
12. **for each** input record (Y,(N,C)) **do**
13. Accumulate in S the sum of N*C
14. Accumulate in C_{new} the sum of C
15. **repeat**
16. **let** A be S/C_{new}
17. **produce one output record** (Y,(A,C_{new}))
18. **end function**

Mapper Phase: Mapper is all about the initialization work for processing data. They value that we get from Input Splitter phase becomes the key here now and each key is assigned a value here. Now, this becomes the new Key-value pairs which is further sent to Reducer or Sort and Shuffling based on the requirement of processing the data. *Sort and Shuffling Phase:* This phase comes into the picture whenever need some repetitive data or grouping of data is required before sending to the Reducer Phase. This will check whether if

there is any data that needs to be sorted and thus sorts the data and send it to reducer.

Reducer Phase: This is the actual phase where aggregation of sorted data happens and finally generating the processed data. Once the data is processed, it is sent back to HDFS using Output Format.

Reducer execution will not begin until all the Mappers are completed:

The Reducer phase starts execution only after all the Mappers completed.Even if one of the mappers fail the reducer phase will have to wait until the failed job is assigned to a different mapper and completed successfully. This could

One way to work around this problem is in the settings in new versions of Hadoop (at least 2.4.1) the parameter is called is mapreduce.job.reduce.slowstart.completedmaps. The value of the setting should be in the range 0 to 1 for example if we set mapreduce .job. reduce.slowstart.completedmaps = 0.7, 70 percent of the mappers should be completed before the reduce phase starts [8].

VIII. RUN YARN

HDFS is a distributed storage system, it doesn't provide any services for running and scheduling tasks in the cluster. This is the role of the YARN framework. The following section is about starting, monitoring, and submitting jobs to YARN.

Start and Stop YARN

- ✓ Start YARN with the script:start-yarn.sh
- ✓ Check that everything is running with the jps command. In addition to the previous HDFS daemon, you should see a ResourceManager on node-master, and a NodeManager on node1 and node2.
- ✓ To stop YARN, run the following command on node-master:stop-yarn.sh
- ✓ Monitor YARN
  //The yarn command provides utilities to manage your YARN cluster. You can also print a report of running nodes with the command:yarn node –list
  //Similarly, you can get a list of running applications with command:
- ✓ yarn application –list
  //As with HDFS, YARN provides a friendlier web UI, started by default on port 8088 of the Resource Manager. Point your browser to http://node-master-IP:8088 and browse the UI:

---

be more time consuming and not an efficient way of using resources [7].

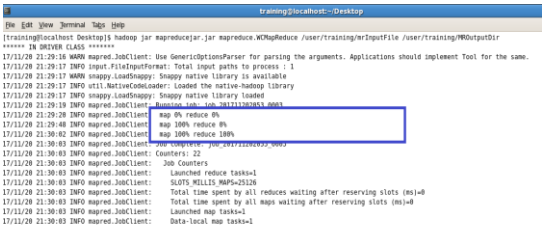

Fig 9.Mapper and reducer execution process

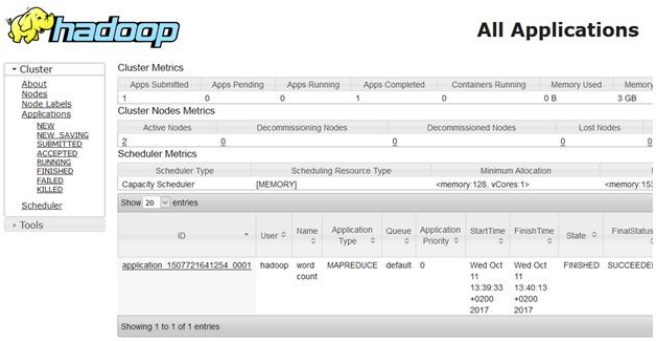

Fig 10: Yarn User interface

IX. SUBMIT MAPREDUCE JOBS TO YARN

Yarn jobs are packaged into jar files and submitted to YARN for execution with the command yarn jar. The Hadoop installation package provides sample applications that can be run to test your cluster. You'll use them to run a word count on the three books previously uploaded to HDFS.

SUBMIT A JOB WITH THE SAMPLE JAR TO YARN. ON NODE-MASTER, RUN

YARNjar~/Hadoop/share/Hadoop/mapreduce/Hadoop-mapreduce-examples-2.8.1.jar wordcount "books/*" output
The last argument is where the output of the job will be saved - in HDFS. After the job is finished, you can get the result by querying HDFS with hdfs dfs -ls output. In case of a success, the output will resemble:
Found 2 items
-rw-r--r-- 1 Hadoop supergroup 0 2017-10-11 14:09 output/_SUCCESS
-rw-r--r-- 1 Hadoop supergroup 269158 2017-10-11 14:09 output/part-r-00000
Print the result with:
hdfs dfs -cat output/part-r-00000

X. PROCESSING STOCKS DATA

For processing data a subset of a stock dataset taken with information of stock symbols. In each line in the dataset, it has information about stock symbol for a day. Information like the opening price, closing price, high, low,

volume, etc, if consider one line in the dataset, each line indicates a record. So, the first one is the exchange name, the next is the symbol, the date, the opening price, the closing price, high, low for the day and the volume. Using this dataset, we would like to find the maximum closing price for each stock symbol across several days. Let's have a look at our dataset.

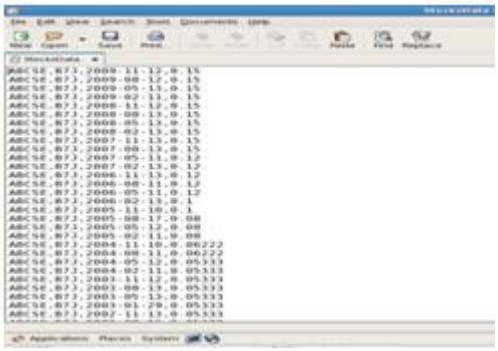

Fig 11: stack dataset

To implement the solution to stock processing data, by using a Single-node cluster developed by Cloudera. It will also be using Eclipse IDE to write the Java Classes for Mapper and reducer.

i. Mapper Class :

Create a Java class with name (let's say) StockMapper and extend the class Mapper from *org.apache.Hadoop.mapreduce.Mapper* package and override the map method to implement the mapper logic in this class. So, let's work on the logic here, it has an input data which has lines of Text data. But, it doesn't have any datatype which can store Text Type. So, we will be converting it into String datatype. Once you convert into the String type, now using split method to differentiate each line from the record. Now from each line or record, need two fields i.e., stock symbol and its closing price. So, assign stock symbol data and closing price to two new variables. The logic here will look something similar to this.

```
String line = value.toString();
String[] items = line.split(",");
String stockSymbol = items[1];
Float closePrice =
Float.parseFloat(items[3]);
```

ii. Reducer Class

Create a Java class with name (let's say) StockReducer and extend the reducer class from org.apache.Hadoop.mapreduce.Reducer package and override reduce method to implement reducer logic in the class. From mapper class, we already have separate stock symbols and their respective closing price values. Now using foreach loop we will be finding the maximum closing price value by comparing with every stock symbol in the dataset. The logic here will look something like this.

```
Float maxClosePrice =Float.MIN_VALUE;
for(FloatWritable value : values)
{
maxClosePrice=
Math.max(maxClosePrice, value.get());
}
```

iii. Driver Class

This is the actual class where execution of the code starts from the main method. In this class, it import many class libraries like Job, FileInputFormat, FileOutputFormat, TextInputFormat, TextOutputFormat and Path Class. This is also the class, where it assign the mapper and reducer classes for execution of the logic. FileInputFormat is used for adding the input path and FileOutputFormat is used to set the output path to store the output in the prescribed location. Once, written our Java Classes, the very next thing we do is creating a Jar File out of these classes by exporting the class. The above screenshot will give you an understanding of how to create a Jar File.

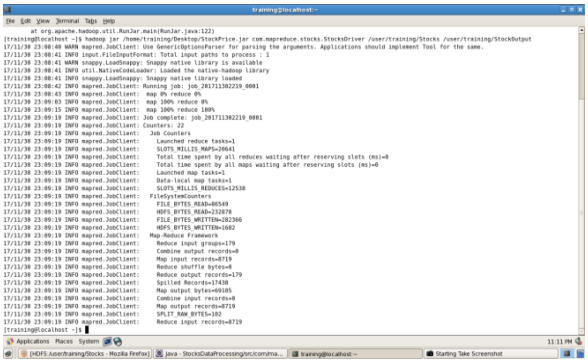

Fig12. Jar files loading

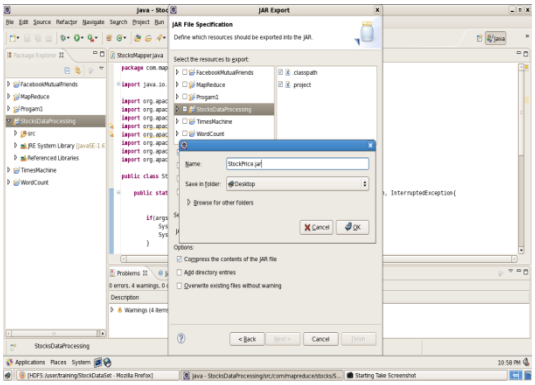

Fig 13: Jar file creation

In this example, created a jar file with name StockPrice.jar . To execute the program in Hadoop, it need to upload the data into HDFS. The command that used to insert data into HDFS is *Hadoop fs -put /home/training/Desktop/StocksData /user/training/Stocks*

Here the first path is the input data file with path and second path is the Output File Path. Let us execute our logic now in Hadoop.

Hadoopjar /home/training/Desktop/StockPrice.jar (Jar File) com.mapreduce.stocks.StocksDriver (Driver Class File) /user/training/Stocks (Input File Path in HDFS) /user/training/StockOutput (Output Folder Path)

Let us now see the results of our logic that implemented to find the maximum closing price of each respective stock symbol.

The fig11 and fig12 screenshot displays the complete mapreduce process happening during the execution time. Now, lets see the our output. We can view the output in HDFS Browser as well as from terminal.

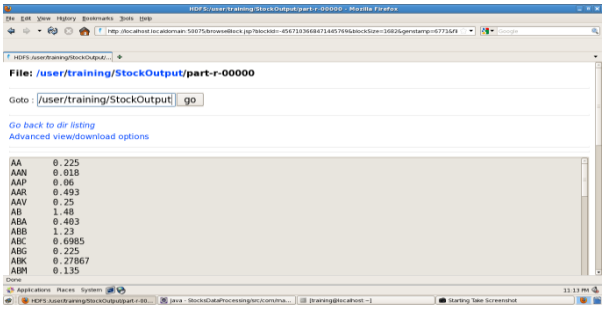

Fig 14: stock price

The above screenshot displays the maximum closing price of each stock symbol after MapReduce processing.

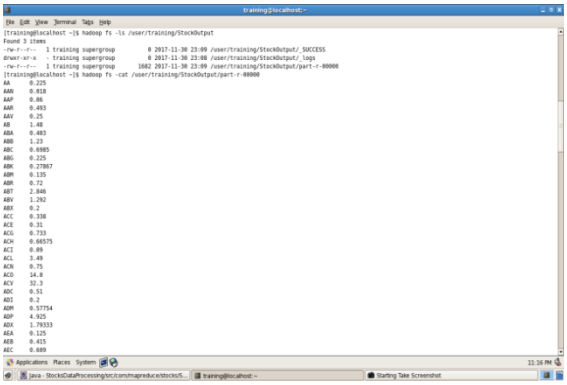

Fig 15: Output window

This is another way to view the output from terminal using the ls command in Hadoop.

XI. CONCLUSION

The MapReduce programming model has been efficiently used at Google for plenty one-of-a-kind purposes. We attribute this achievement to several motives. First, the model is easy to use, even for programmers without revel in parallel and distributed systems, because it hides the info of parallelization, fault-tolerance, locality optimization, and cargo balancing.

Second, a big type of troubles are effortlessly expressible as MapReduce computations. For example, MapReduce is used for the generation of statistics for Google's manufacturing net search service, for sorting, for records mining, for gadget mastering, and many different structures.

Third, we have developed an implementation of MapReduce that scales to big clusters of machines comprising heaps of machines. The implementation makes green use of these system resources and therefore is appropriate for use on some of the huge computational problems encountered at Google. Redundant execution can be used to reduce the effect of gradual machines, and to deal with system disasters and data loss.

REFERENCES

[1]. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, H. A. Jacobsen, 2013 "BigBench: Towards an industry standard benchmark for big data analytics", Proc. ACM SIGCOMM USA, pp. 1197-1208.

[2]. D. Sun, G. Zhang, S. Yang, W. Zheng, S. U. Khan, K. Li, 2015 "Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments", Information Sciences., vol. 319, pp. 92-112.

[3]. Hideya Nakanishi; Masaki Ohsuna; Mamoru Kojima; Setsuo Imazu; Miki, 2016, "Real-Time Data Streaming and Storing Structure for the LHD's Fusion Plasma Experiments", ISSN: 1558-1578, Volume: 63, Issue: 1, pp: 222 – 227.

[4]. Jeongkyu Hong ; Soontae Kim, 2017, "Smart ECC Allocation Cache Utilizing Cache Data Space" ISSN: 0018-9340, Volume 66, Issue 2, pp: 368 – 374.

[5]. Miao Wang ; Guiling Wang ; Yujun Zhang ; Zhongcheng Li, 2016, "A High-reliability Multi-faceted Reputation Evaluation Mechanism for Online Services", ISSN: 1939-1374, Volume PP, Issue: 99, pp 1-1.

[6]. Neha Bharill ; Aruna Tiwari ; Aayushi Malviya, 2016, "Fuzzy Based Scalable Clustering Algorithms for Handling Big Data Using Apache Spark", ISSN: 2332-7790, Volume: 2, Issue 4, pp: 339-352.

[7]. Neng Zhang ; Jian Wang ; Yutao Ma, 2017, 7. "Mining Domain Knowledge on Service Goals From Textual Service Descriptions", ISSN: 1939-1374, Volume PP Issue 99, pp:1-1.

[8]. Yuji Ishizuka; Wuhui Chen; Incheon Paik, 2016, "Workflow Transformation for Real-Time Big Data Processing", ISSN: 978-1-5090-2622-7, Big Data (BigData Congress), 2016 IEEE International Congress on Wuhui Chen; Incheon Paik; Zhenni Li, 2017,

[9]. "Cost-Aware Streaming Workflow Allocation on Geo-Distributed Data Centers" ISSN: 0018-9340, Volume 66, Issue: 2, pp: 256-271.

Authors Profile

Ms. Bhavani Buthukuri, Research Scholar from SHRI JAGDISHPRASAD JHABARMAL TIBREWALA UNIVERSITY, received her Bachelor's degree in Computer Science & Information Technology in the year 2006 and received her Master's degree in Computer Science and Engineering in the year 2011. Pursuing Ph.D in Real Time Stream processing for Big Data. Research interests include Hadoop & Data Warehousing.

Dr M E Purushoththaman Professor from the department of Computer Science and Engineering, Hyderabad. A docotorate in Neural Networks(2007) and Cloud Computing(2016), achieved his Bachelor degree from the Royal Charter, The Institution of Engineers(India) on 2000 and his Masters on 2003 from the Punjabi University, Patiala, docotorated by the Fellow of IE(India) and also Chartered Engineer(India), in academics as Principal a decade and hold various positions earlier for seven years, and a product of prouded soldier from the great EME of Indian Army. He has published more than 20 research papers in reputed international journals and conferences including IEEE. His main research work focuses on Cloud Computing, Network Security, Cloud Security and Privacy, Big Data Analytics, Data Mining, He has 17 years of teaching experience including 14 years of Research Experience.

