HADOOP AND ITS ARCHITECTURE

Hadoop is a java based framework that is well-organized for processing great data sets in a dispersed computing environment. Hadoop is sponsored by Apache Software Foundation. The maker of Hadoop was Doug Cutting and he named the framework after his child’s ballooned toy elephant. Applications are ready run on systems with thousands of nodes making use of thousands of terabytes via Hadoop. Distributed file system in Hadoop facilitates fast data shift among nodes and allows continuous operations of the system even if node failure occurs. This concept lowers the risk of disastrous system failure even if multiple nodes become inoperative. The inspiration behind working of Hadoop is Google’s Map reduce which is a software framework in which application under consideration is broken down into number of small parts. In the cluster, any of these fragments can run on any node. The apparatus involved in Hadoop system are Hadoop kernel, Map-Reduce, Hadoop distributed file system and loads of correlated projects like Apache hive, HBase and Zookeeper.

The major players like Yahoo, IBM and Google are uses Hadoop framework. Ultimate operating systems for Hadoop are Windows and Linux as well as worked with BSD and OS X. In previous days, the equipment implicated was developed by Google in order to index all precious textural and structural in order collected by them. To provide meaningful result to the user all this was done. Later this innovation of google was included into Nutch which was an open source project and Hadoop was spun off from it. The most outstanding in developing Hadoop for enterprise applications was Yahoo. Hadoop platform is favored to solve problems where data is vast and does not robustproperly in tables. It is used to run analytics that involves widecalculation. There are many sectors where Hadoop finds it application. It is used in finance to achievecorrectcollection evaluation and risk analysis. In online retail, it helps in providing superior answers to the clients to increase the possibility of their buying the things.
Hadoop runs on enormous number of machines that do not share any memory space. Alike to buying a whole group of product servers slap them in a frame and run Hadoop software on all of them. Suppose one decides to load all the data of their connection in Hadoop, the software busts that data into pieces and extend them across different servers. Hadoop keeps path of where the whole data resides. Also mentionable is that if data on server goes offline, it can be replicated repeatedly from a known good copy. One big disk that is linked to four, eight or sixteen processors in centralized database systems. But in gap, each of these servers has two, four or eight CPUs in Hadoop. You can run your indexing job by distributed your scheme to each of the dozens of servers in your group, and each server operates on its own little piece of the data. The result is then delivered back together. This is what is known as map - reduce. In all these servers Map reducing technique involves mapping the operations out and after that these results are condensed back into a single result set.

As Map reduces an algorithm, it can be written in any programming language. Hadoop map decrease works in three stages:

- **First Stage: mapping:** A list of elements is provided to a ‘mapper’ function to get it transferred into pairs in this stage. The input data does not alter by ‘mapper’ function, although merely proceeds a fresh output list.

- **Intermediate stages: Shuffling and Sorting:** After the mapping stage, the midway outputs interactions by the program from the mapping stage to different ‘reducers’. This procedure is called shuffling.

- **Final Stage: Reducing:** In the final reducing stage, each key is called by an example of a user-provided symbols in the divider assigned to a reducer. In particular, we have one output file per executed diminish task.
4.1 COMPONENTS OF HADOOP

Hadoop is an agenda which comprised of six mechanisms. Each mechanism is assigned an testing job to be performed. To identify with it suppose whole arrangement as a Hadoop zoo as shown in Fig.

Fig. 1 Working of Map Reduce
• **HDFS** – HDFS are sprinkled cages where each single one animal breathe i.e. where data resides in a spotted layout. HDFS enables the center storage space for the Hadoop bunch. It divides the records keen on small parts and distributes it across the different servers/nodes.

• **Apache H Base** – It is a graceful and massive catalog. H Base is a column-oriented database management scheme that sits on peak of HDFS. It uses a non-SQL approach.

• **Zookeeper**- Zookeeper is the person liable for managing animals play. Zookeeper allows a centralized infrastructure with various services, providing harmonization across a bunch of servers. Big data analytics applications use these services to coordinate parallel processing.

• **Pig** – Pig allows playing with data from HDFS cages. Pig programming language is configured to incorporate all types of data (structured/unstructured, etc.). It is comprised of two key modules: the language itself, called Pig Latin, and the runtime version in which the Pig Latin code is executed.

• **Hive**- Hive allows data analysts play with HDFS and makes use of SQL. Hive is a runtime Hadoop hold up architecture that leverages Structure Query Language (SQL) with the Hadoop platform. It permits SQL programmers to enlarge Hive Query Language (HQL) statements akin to typical SQL statements.

• **HCatalog** helps to upload the database file and automatically make table for the user.

### 4.2 MAP REDUCING FUNCTIONING

Map Reduce is a framework initially developed at Google that allows for simple large scale distributed computing across a digit of domains. The Apache Hadoop software library is a structure that allows for the dispersed processing of large data sets across clusters of computers using simple programming models. It is planned to scale up from single servers to thousands of machines, each contributing local computation and storage. Hadoop Map Reduce includes several stages, each with a significant set of operations helping to get to your goal of getting the
answers you need from big data. The procedure starts with a user demand to run a Map Reduce program and continues until the results are written back to the HDFS.

Fig. 3 Working of Map Reduce Technology

Map Reduce is an architectural model for parallel processing of responsibilities on a distributed computing system. This algorithm was first described in a paper "Map Reduce Simplified Data Processing on Large Clusters," by Jeffery Dean and Sanjay Ghemwat from Google. This algorithm allows splitting of a single calculation task to multiple nodes or computers for distributed processing.

As a solitary task can be broken down into multiple subparts, each handled by a detach node, the number of nodes determines the processing power of the system. There are various profitable and open-source technologies that employ the Map Reduce algorithm as a part of their internal architecture. A popular implementation of Map Reduce is the Apache Hadoop, which is used for data processing in a dispersed computing environment. As Map Reduce is an algorithm, it can be written in any programming language.

The initial part of the algorithm is used to divide and 'map' the sub tasks to computing nodes. The 'reduce' part takes the results of individual computations and combines them to get the final result. In the Map Reduce algorithm, the mapping function reads the input data and generates a set of halfway records for the computation. These halfway records generated by the map function.
take the shape of a (key, data) pair. As a part of mapping function, these proceedings are distributed to different computing nodes using a hashing function. Individual nodes then carry out the computing operation and revisit the results to decrease function. The reduce function collects the individual consequences of the computation to make a final output.

4.3 FLOWCHART OF MAP REDUCING

The flowchart depicting the working of Map-Reduce technology is shown in Fig. 4.
Fig. 3. Flowchart showing working of Map-Reduce technology

### 4.4 ALGORITHM OF MAP REDUCING

- The incoming data can be estranged into n number of modules which depends upon the quantity of input data and processing power of the individual unit.

- All these disjointed modules are then approved over to mapper function where these modules undergo simultaneous parallel processing.

- Thereafter, shuffling is conducted in order to get together similar looking patterns.

- Finally reducer purpose is called which is conscientious for getting the ultimate output in a reduced form.

- Moreover, this technique is scalable and depending upon augment in the data to be processed, the processing units can be further extended.

The Map Reduce algorithm contains two significant tasks, namely Map and Reduce.

- The map task is done by way of Mapper Class
- The diminish task is done by means of Reducer Class.

Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.
Map Reduce implements various mathematical algorithms to split a job into small parts and allocate them to multiple systems. In technical terms, Map Reduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.

These mathematical algorithms may include the following –

- Sorting
- Searching
- Indexing
- TF-IDF

**Sorting**

Sorting is one of the fundamental Map Reduce algorithms to route and analyze data. Map Reduce equipment sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys.

- Sorting methods are implemented in the mapper class itself.

- In the Shuffle and Sort phase, after tokenizing the values in the mapper class, the **Context** class (user-defined class) collects the matching valued keys as a collection.

- To collect similar key-value pairs (intermediate keys), the Mapper class takes the help of **RawComparator** class to sort the key-value pairs.

- The set of middle key-value pairs for a given Reducer is involuntarily sorted by Hadoop to form key-values (K2, {V2, V2, …}) before they are presented to the Reducer.

**Searching**

Searching plays an important role in Map Reduce algorithm. It helps in the combiner phase (optional) and in the Reducer phase. Let us try to understand how Searching works with the help of an example.
4.5 EXAMPLES OF MAP REDUCING

4.5.1 Example 1

The popular social networking website Facebook also makes use of map reduce technology. We will illustrate this via an example.

Facebook has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine). They also have lots of disk space and they serve hundreds of millions of requests every day. They've decided to pre-compute calculations when they can to reduce the processing time of requests. One common processing request is the "I and Gagan have 230 friends in common" feature. When you visit someone's profile, you see a list of friends that you have in common. This list doesn't change frequently so it'd be wasteful to recalculate it every time you visited the profile. We're going to use map reduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

Assume the friends are stored as Person \(\rightarrow\) [List of Friends], our friends list is then:

A \(\rightarrow\) B C D

B \(\rightarrow\) A C D E

C \(\rightarrow\) A B D E

D \(\rightarrow\) A B C E

E \(\rightarrow\) B C D

Each line will be an argument to a mapper. For every friend in the list of friends, the mapper will output a key-value pair. The key will be a friend along with the person. The value will be the list of friends. The key will be sorted so that the friends are in order, causing all pairs of friends to go to the same reducer. This is hard to explain with text, so let's just do it and see if you can see the pattern. After all the mappers are done running, you'll have a list like this:

For map (A\(\rightarrow\)B C D):
(A B) \(\rightarrow\) B C D

(A C) \(\rightarrow\) B C D

(A D) \(\rightarrow\) B C D

For map (B \(\rightarrow\) A C D E): (Note that A comes before B in the key)

(A B) \(\rightarrow\) A C D E

(B C) \(\rightarrow\) A C D E

(B D) \(\rightarrow\) A C D E

(B E) \(\rightarrow\) A C D E

For map(C \(\rightarrow\) A B D E):

(A C) \(\rightarrow\) A B D E

(B C) \(\rightarrow\) A B D E

(C D) \(\rightarrow\) A B D E

(C E) \(\rightarrow\) A B D E

For map (D \(\rightarrow\) A B C E):

(A D) \(\rightarrow\) A B C E

(B D) \(\rightarrow\) A B C E

(C D) \(\rightarrow\) A B C E

(D E) \(\rightarrow\) A B C E

And finally for map (E \(\rightarrow\) B C D):

(B E) \(\rightarrow\) B C D
Before we send these key-value pairs to the reducers, we group them by their keys and get:

- \((A B) \rightarrow (A C D E) (B C D)\)
- \((A C) \rightarrow (A B D E) (B C D)\)
- \((A D) \rightarrow (A B C E) (B C D)\)
- \((B C) \rightarrow (A B D E) (A C D E)\)
- \((B D) \rightarrow (A B C E) (A C D E)\)
- \((B E) \rightarrow (A C D E) (B C D)\)
- \((C D) \rightarrow (A B C E) (A B D E)\)
- \((C E) \rightarrow (A B D E) (B C D)\)
- \((D E) \rightarrow (A B C E) (B C D)\)

Each line will be passed as an argument to a reducer. The reduce function will simply intersect the lists of values and output the same key with the result of the intersection. For example, reduce \(((A B) \rightarrow (A C D E) (B C D))\) will output \((A B): (C D)\) and means that friends A and B have C and D as common friends.

The result after reduction is:

- \((A B) \rightarrow (C D)\)
- \((A C) \rightarrow (B D)\)
- \((A D) \rightarrow (B C)\)
- \((B C) \rightarrow (A D E)\)
Now when D visits B’s profile, we can quickly look up (B D) and see that they have three friends in common, (A C E).

This is how Face book analyses millions of user accounts created on it and finds out that to which what people should be shown in there “people you may know section”.

4.5.2 Example 2

The following example shows how MapReduce employs Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset.

- Let us suppose we have employee data in four different files – A, B, C, and D. Let us also assume there are duplicate employee records in all four files because of importing the employee data from all database tables frequently. See the following figure.

<table>
<thead>
<tr>
<th>Name, salary</th>
<th>name, salary</th>
<th>name, salary</th>
<th>name, salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suresh, 26000</td>
<td>Hitesh, 5000</td>
<td>Suresh, 26000</td>
<td>Suresh, 26000</td>
</tr>
<tr>
<td>Mahesh, 25000</td>
<td>Mahesh, 25000</td>
<td>Mohan, 45000</td>
<td>Mahesh, 25000</td>
</tr>
<tr>
<td>Ram, 15000</td>
<td>Ram, 15000</td>
<td>Ram, 15000</td>
<td>Tanu, 45000</td>
</tr>
<tr>
<td>Sham, 1000</td>
<td>Sham, 1000</td>
<td>Sham, 1000</td>
<td>Sham, 1000</td>
</tr>
</tbody>
</table>

- The Map phase processes each input file and provides the employee data in key-value pairs (<k, v>:<emp name, salary>). See the following figure.
< Name, salary >< Name, salary >< Name, salary >< Name, salary >

< Suresh, 26000 >< Hitesh, 5000 >< Suresh, 26000 >< Suresh, 26000 >

< Mahesh, 25000 >< Mahesh, 25000 >< Mohan, 45000 >< Mahesh, 25000 >

< Ram, 15000 >< Ram, 15000 >< Ram, 15000 >< Tanu, 45000 >

< Sham, 1000 >< Sham, 1000 >< Sham, 1000 >< Sham, 1000 >

- **The combiner phase** (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary. Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file. See the following snippet:

```
<k: employeeName, v: salary>
Max= the salary of a first employee. Treated as max salary

if(v(second employee).salary > Max)
{
    Max = v(salary);
}
else
{
    Continue checking;
}
```

The expected result is as follows –

< suresh,26000>    <Hitesh,50000>    <mohan,45000>    <tanu,45000>
- **Reducer phase** – Form each file, you will find the highest salaried employee. To avoid redundancy, check all the \(k, v\) pairs and eliminate duplicate entries, if any. The same algorithm is used in between the four \(k, v\) pairs, which are coming from four input files. The final output should be as follows –

| <Hitesh, 50000> |

**Indexing**

Normally indexing is used to point to a particular data and its address. It performs batch indexing on the input files for a particular Mapper.

The indexing technique that is normally used in MapReduce is known as **inverted index**. Search engines like Google and Bing use inverted indexing technique. Let us try to understand how Indexing works with the help of a simple example.

**Example**

The following text is the input for inverted indexing. Here \(T[0], T[1], \text{and } t[2]\) are the file names and their content are in double quotes.

\(T[0] = "\text{it is what it is}"\)

\(T[1] = "\text{what is it}"\)

\(T[2] = "\text{it is a banana}"\)

After applying the Indexing algorithm, we get the following output –

"a": \(\{2\}\)

"banana": \(\{2\}\)

"is": \(\{0, 1, 2\}\)

"it": \(\{0, 1, 2\}\)
"what": {0, 1}

Here "a": {2} implies the term "a" appears in the T[2] file. Similarly, "is": {0, 1, 2} implies the term "is" appears in the files T[0], T[1], and T[2].

TF-IDF

TF-IDF is a text processing algorithm which is short for Term Frequency – Inverse Document Frequency. It is one of the common web analysis algorithms. Here, the term 'frequency' refers to the number of times a term appears in a document.

Term Frequency (TF)

It measures how frequently a particular term occurs in a document. It is calculated by the number of times a word appears in a document divided by the total number of words in that document.

\[
\text{TF (the)} = \frac{\text{(Number of times term the ‘the’ appears in a document)}}{\text{(Total number of terms in the document)}}
\]

Inverse Document Frequency (IDF)

It measures the importance of a term. It is calculated by the number of documents in the text database divided by the number of documents where a specific term appears.

While computing TF, all the terms are considered equally important. That means, TF counts the term frequency for normal words like “is”, “a”, “what”, etc. Thus we need to know the frequent terms while scaling up the rare ones, by computing the following –

\[
\text{IDF (the)} = \log \left( \frac{\text{Total number of documents}}{\text{Number of documents with term ‘the’ in it}} \right)
\]