Big Data - Map Reduce Framework and Programming Model

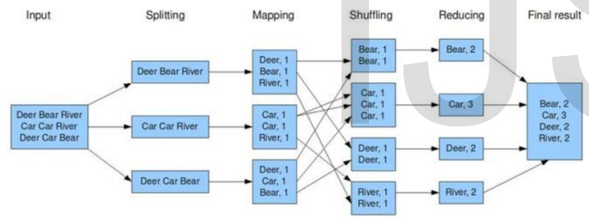
Pankaj Singh, Ankur Chaudhary

Abstract - We live in a world of Big Data. This year alone, over a trillion gigabytes of new data will be created globally. According to a survey, by year 2020, 45 zettabytes of data will be stored in the world. Twitter itself generates 7 TB of data every day whereas Facebook generates 10 TB of data every day. Big Data presents a big challenge – but also exciting new opportunities for enterprises to rise above the competition. The volume of the data generated in an organization is on an increase while the percent of the data we can analyze is on decline. Hence organizations require an efficient and scalable storage system to manage data growth. Scale-out storage, paired with powerful analytics tools that can derive valuable insight from oceans of content, are the right combination for making the most of big data.

Index Terms—Big Data, Map Reduce, Framework, Programming Model, Reducer, Mapper, Fault Tolerance.

1. Introduction

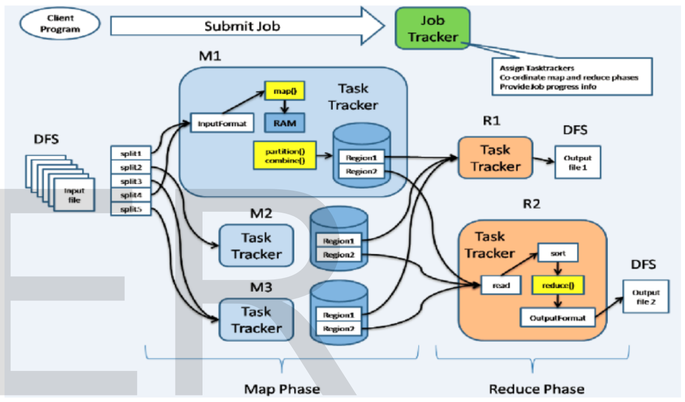
MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.



MapReduce has become a dominant parallel computing paradigm for big data, i.e., colossal datasets at the scale of tera-bytes or higher. Ideally, a MapReduce system should achieve a high degree of load balancing among the participating machines, and minimize the space usage, CPU and I/O time, and network transfer at each machine.

2. Framework

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the job's component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.



DFS (Distributed File System) is a distributed file system designed to run on many commodity hardware.DFS is fault tolerant and designed to be deployed on low cost hardware.

Job Tracker is a job configuration which specifies the map (M1, M2, M3 etc), combine and reduce function, as well as the input and output path of the data. JobConf is the primary interface for a user to describe a MapReduce job for execution. The framework tries to faithfully execute the job as described by JobConf. The JobTracker will first determine the number of splits (each split is configurable, ~16-64MB) from the input path, and select some TaskTracker based on their network proximity to the data sources, then the JobTracker send the task requests to those selected TaskTrackers.

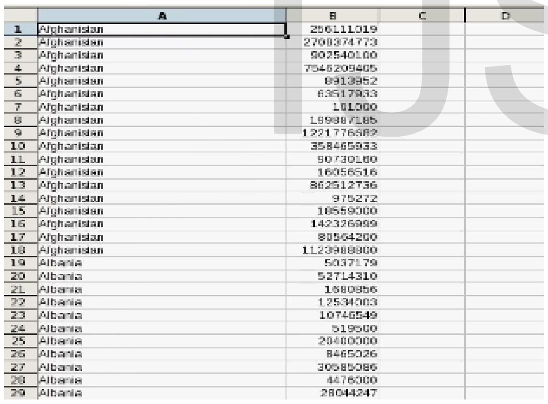
Task Tracker will initiate the assigned mapper by extracting the input data from split, as a child process in a separate java virtual machine.

Input format is a chunk of the input that is processed by single map. Each map processes a single split. Each split is divided into records and the map processes each record – a key-value pair in turn.

3. Algorithm/Programming Model

In the MapReduce, a distributed file system (DFS) initially partitions data in multiple machines and data is represented as (key, value) pairs. The computation is carried out using two user defined functions: map and reduce functions. Both map and reduce functions take a key-value pair as input and may output key-value pairs. The map function defined by a user is first called on different partitions of input data in parallel. The key-value pairs output by each map function are next grouped and merged by each distinct key. Finally, a reduce function is invoked for each distinct key with the list of all values sharing the key. The output of each reduce function is written to a distributed file in the DFS.

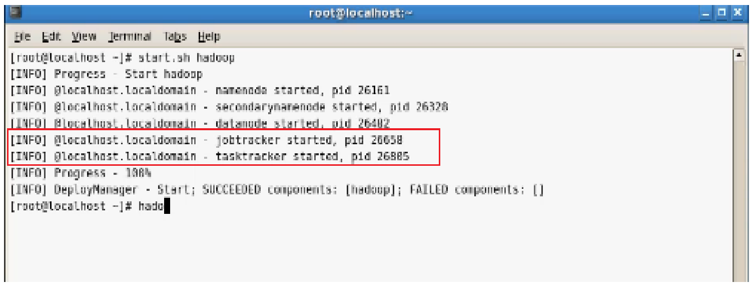
Now we will look into an algorithm to summarize the input metric as per the country.



	A	B	C	D
1	Afghanistan	256111019		
2	Afghanistan	2708374773		
3	Afghanistan	902540100		
4	Afghanistan	7546209405		
5	Afghanistan	8913952		
6	Afghanistan	63517933		
7	Afghanistan	101000		
8	Afghanistan	189887185		
9	Afghanistan	1221776682		
10	Afghanistan	358465933		
11	Afghanistan	90730100		
12	Afghanistan	16056516		
13	Afghanistan	862512736		
14	Afghanistan	975272		
15	Afghanistan	10559000		
16	Afghanistan	142326999		
17	Afghanistan	80564200		
18	Afghanistan	1123988800		
19	Albania	5037179		
20	Albania	52714310		
21	Albania	1680856		
22	Albania	12534003		
23	Albania	10746549		
24	Albania	519500		
25	Albania	20400000		
26	Albania	8465026		
27	Albania	30585086		
28	Albania	4476000		
29	Albania	28044247		

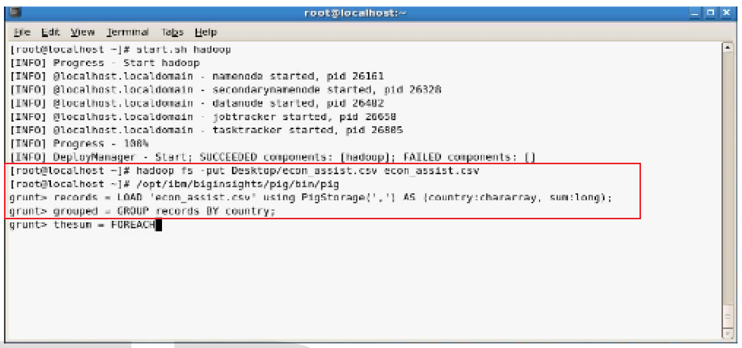
As we can see that the input file has two columns –country, revenue. Our objective is to summarize all revenue data so that we can have only one row per country with its total revenue data.

Our next step is to submit the job and initialize Job tracker and henceforth Task Tracker.



```
[root@localhost ~]# start.sh hadoop
[INFO] Progress - Start hadoop
[INFO] @localhost.localdomain - namenode started, pid 26161
[INFO] @localhost.localdomain - secondarynamenode started, pid 26328
[INFO] @localhost.localdomain - datanode started, pid 26402
[INFO] @localhost.localdomain - jobtracker started, pid 26658
[INFO] @localhost.localdomain - tasktracker started, pid 26885
[INFO] Progress - 100%
[INFO] DeployManager - Start; SUCCEEDED components: [hadoop]; FAILED components: []
[root@localhost ~]# hado
```

The Task Tracker now attaches itself with the input file/split. Below is the command for that in hadoop

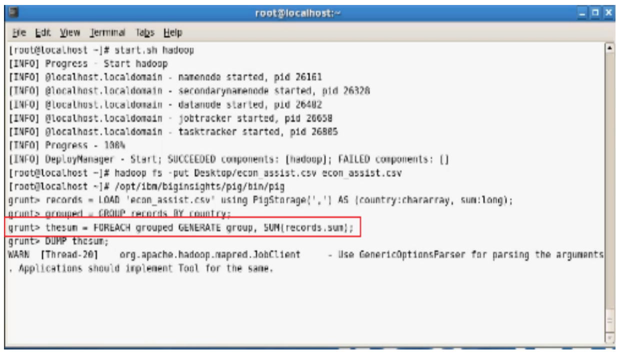


```
[root@localhost ~]# start.sh hadoop
[INFO] Progress - Start hadoop
[INFO] @localhost.localdomain - namenode started, pid 26161
[INFO] @localhost.localdomain - secondarynamenode started, pid 26328
[INFO] @localhost.localdomain - datanode started, pid 26402
[INFO] @localhost.localdomain - jobtracker started, pid 26658
[INFO] @localhost.localdomain - tasktracker started, pid 26885
[INFO] Progress - 100%
[INFO] DeployManager - Start; SUCCEEDED components: [hadoop]; FAILED components: []
[root@localhost ~]# hadoop fs -put Desktop/econ_assist.csv econ_assist.csv
[root@localhost ~]# /opt/ibm/biginsights/pig/bin/pig
grunt> records = LOAD 'econ_assist.csv' using PigStorage(',') AS (country:chararray, sum:long);
grunt> grouped = GROUP records BY country;
grunt> thesum = FOREACH
```

Once data gets loaded by the task tracker on mapper, it process the data and provide result as per the below mentioned algorithm form.

$$\text{map} \quad (\mathtt{k1,v1}) \quad \rightarrow \mathtt{list(k2,v2)}$$

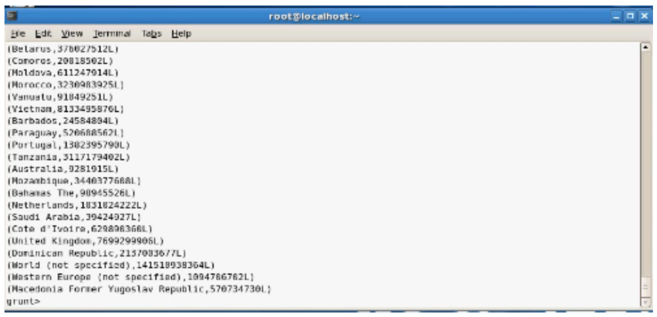
The output from the mapper is fed as the input to the reducer.



```
[root@localhost ~]# start.sh hadoop
[INFO] Progress - Start hadoop
[INFO] @localhost.localdomain - namenode started, pid 26161
[INFO] @localhost.localdomain - secondarynamenode started, pid 26328
[INFO] @localhost.localdomain - datanode started, pid 26402
[INFO] @localhost.localdomain - jobtracker started, pid 26658
[INFO] @localhost.localdomain - tasktracker started, pid 26885
[INFO] Progress - 100%
[INFO] DeployManager - Start; SUCCEEDED components: [hadoop]; FAILED components: []
[root@localhost ~]# hadoop fs -put Desktop/econ_assist.csv econ_assist.csv
[root@localhost ~]# /opt/ibm/biginsights/pig/bin/pig
grunt> records = LOAD 'econ_assist.csv' using PigStorage(',') AS (country:chararray, sum:long);
grunt> grouped = GROUP records BY country;
grunt> thesum = FOREACH grouped GENERATE group, SUM(records.sum);
grunt> DUMP thesum;
WARN  [Thread-20]    org.apache.hadoop.mapred.JobClient     - Use GenericOptionsParser for parsing the arguments
. Applications should implement Tool for the same.
```

The reducer processes the data and gives us the desired output as per the algorithm

```
reduce    (k2,list(v2))   → list(v2)
```



```
(Belarus,376027512L)
(Comoros,20818502L)
(Moldova,611247914L)
(Morocco,3230983925L)
(Vanuatu,91849251L)
(Vietnam,8133495876L)
(Barbados,24584894L)
(Paraguay,5206885621L)
(Portugal,1302395790L)
(Tanzania,3117179402L)
(Australia,9281915L)
(Mozambique,3440377668L)
(Bahamas The,90945526L)
(Netherlands,1831824222L)
(Saudi Arabia,39424927L)
(Cote d'Ivoire,6298983668L)
(United Kingdom,7699299906L)
(Dominican Republic,2157083677L)
(World (not specified),141518938364L)
(Western Europe (not specified),1094786782L)
(Macedonia Former Yugoslav Republic,570734730L)
grunt>
```

Another Example – Pesudo code for the word count algorithm

```
class MAPPER
    method MAP(docid $a$, doc $d$)
        for all term $t \in$ doc $d$ do
            EMIT(term $t$, count 1)
class REDUCER
    method REDUCE(term $t$, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ do
            $sum \leftarrow sum + c$
        EMIT(term $t$, count $sum$)
```

Input:

- Key-value pairs: (docid, doc) stored on the distributed filesystem
- docid: unique identifier of a document
- doc: is the text of the document itself

Mapper:

- Takes an input key-value pair, tokenize the document
- Emits intermediate key-value pairs: the word is the key and the integer is the value

The Framework:

- Guarantees all values associated with the same key (the word) are brought to the same reducer.

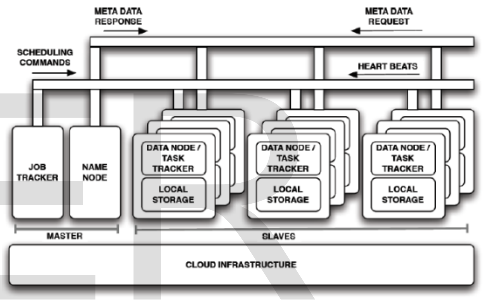
The Reducer

- Receives all values associated to some keys
- Sums the values and writes output key-value pairs: the key is the word and the value is the number of occurrences

4. FAULT TOLERANCE

Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully.

One of the main components of Hadoop is the JobTracker, which is executed as a daemon process that executes on a master node. The JobTracker is the scheduler and the main coordinator of tasks. It is in charge of distributing the MapReduce tasks between the available computing nodes. Each time the JobTracker receives a new job to execute, it contacts a set of TaskTracker processes, which are daemons that execute on the working nodes (one TaskTracker exists for each worker in the infrastructure). MapReduce tasks are then assigned to those working nodes for which their TaskTracker daemons report that they have available slots for computation (several tasks assigned to the same worker are handled by a single TaskTracker daemon).



The fault tolerance mechanisms implemented in Hadoop are limited to reassign tasks when a given execution fails. In this situation, two scenarios are supported:

1. In case a task assigned to a given TaskTracker fails, a communication via the Heartbeat is used to notify the JobTracker, which will reassign the task to another node if possible.

2. If a TaskTracker fails, the JobTracker will notice the faulty situation because it will not receive the Heartbeats from that TaskTracker. Then, the JobTracker will assign the tasks the TaskTracker had to another TaskTracker.

There is also a single point of failure in the JobTracker, since if it fails, the whole execution fails. The main benefits of the standard approach for fault tolerance implemented in Hadoop consists on its simplicity and that it seems to work well in local clusters.

5. CONCLUSIONS

The MapReduce programming model has been successfully used at Google,yahoo, facebook for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. We have learned several things from this work. First, restricting the programming model makes it easy to parallelize and distribute computations and to make such computations fault-tolerant. Second, network bandwidth is a scarce resource. A number of optimizations in our system are therefore targeted at reducing the amount of data sent across the network: the locality optimization allows us to read data from local disks, and writing a single copy of the intermediate data to local disk saves network bandwidth. Third, redundant execution can be used to reduce the impact of slow machines, and to handle machine failures and data loss.

REFERENCES

[1] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, pages 137–150, 2004.
[2] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In SIGMOD, pages 973–984, 2011.
[3] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. Mrshare: Sharing across multiple queries in mapreduce. PVLDB, 3(1):494–505, 2010.
[4] B. Panda, J. Herbach, S. Basu, and R. J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. PVLDB, 2(2):1426–1437, 2009.
[5] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. PVLDB, 4(11):1135–1145, 2011.
[6] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In SODA, pages 938–948, 2010.
[7] Communications between the TaskTrackers and the JobTracker in Hadoop, Kadirvel and Fortes, 2013.
[8] Apache Hadoop, 2013 and Patil and Soni,2013
[9] R. Chaiken, B. Jenkins, P. ake Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. PVLDB, 1(2):1265–1276, 2008.