IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 4, April 2017

Use of MapReduce in Distributed Systems

Nishant Saxena[1], Shikha Sharma Sarkar[2]

B.Tech Scholar, Poornima Group of Institution, Jaipur[1]

Dept of Computer Science, Poornima Group of Institution, Jaipur[2]

Abstract: MapReduce is a programming model or software framework which is associated with the implementation of generating large data sets and their processing to a broad variety of real world task. Programmers computes in terms of a map and a reduce function. There are various programs written in the style that automatically functions parallel and are executed on large clusters of commodity computers. MapReduce jobs are executed on commodity computers every day, processing a total in petabytes of data per day.

Keywords: Big data, Data science, Map Reduce and Distributed Systems.

I. INTRODUCTION

Prior to the development of MapReduce there were many special purpose computations that results in processing of large amount of raw data, such as web documents, inverted indices, web requests log etc. Some common applications of distributed computing are Webpage crawling, grid computing, distributed databases, wireless sensing networks, etc. However, the input data is so large and processing should made on distributed machines within a smaller interval of time. These issues of parallelizing computation, processing over distributed data, producing results in lesser time causes the complexity in computation. As a result of complexity we focus on using some simple computation techniques and hiding the complex details of parallelization, fault tolerance and data distribution. Fault tolerance may be defined as the condition in which some of the hardware components of a system are failed but the system still results in the correct output without any loss in data. The main functionality of fault tolerance is to remove frequently occurring failures that disturbs the normal processing of the system. In this more digitized world, where the data is increasing with such a great rate that to monitor it is really a big task. To monitor this problem or to analyze this Big Data from different social networking sites, Ecommerce websites, clinical and public databases, MapReduce comes in play. MapReduce is a programming model which is associated with processing of large data sets. In this the run-time systems takes care of portioning the input data, scheduling of program execution, etc. A typical MapReduce processes many terabytes of data on different machines

II. LITERATURE REVIEW

"MapReduce: Simplified Data Processing on large Clusters [1]"
In this research paper the authors have discussed about the basic functionality of Map and Reduce functions. In addition to this they have also mentioned about some refinements in MapReduce and some of the improvements for better performance of the distributed system.
"MapReduce for Machine Learning on Multicore[2]"
In this research paper the authors have discussed about what the key value pairs are in a Map and Reduce function, what are the values will map and reduces functions takes as input and what are the outputs generated by them.
Fault Tolerance in MapReduce and what is its proper functioning with HDFS (Hadoop Distributed File System).
"Gray, J.sort benchmark home page [3]"
In this link the authors of the paper discussed about the proper functioning and execution of Grep and Sort in MapReduce i.e. how thousands of petabytes of data is scanned according to implemented filters using Grep and how that scanned data is further sorted for processing over distributed systems using Sort.

III. PROGRAMMING MODEL OF MAPREDUCE

The processing of Map Reduce takes a set of input key/value pairs and produces a set some output key/value pairs. The computation in MapReduce is basically done with the help of two functions:
p ():
It is written by the user, it takes an input pair and produces a set of intermediate key/value pairs.
Reduce ():
This is also written by the user, it takes intermediate key and set of values for that key and merges these values to produce smaller set of values.

Example:
Consider a problem of counting the number of each word in large collection of documents.
The user will write a code as:

```
map (String key, String value):
// key: it is document name
// value: it is document contents for each word w in value.
EmitIntermediate (w,"1");
reduce (String key, Iterator values):
// key: it is a word
// values: it is list of counts
int result=0;
for each v in values:
result + = ParseInt(v);
Emit (As String (result));
```

The map function emits each word plus an associated count of occurrence.
The reduce function sums together all counts emitted for a particular word. [2, 5]

IV. IMPLEMENTATION

Various different implementation of MapReduce are possible, the right choice of any implementation depends on the environment. As one of the implementation may be suitable for a large collection of networked machines.

Example:

```
SELECT age, Avg(Contacts)
FROM info.person
GROUP BY age
ORDER BY age

function Map is
  input: integer K between 1 and 1100, representing a batch of 1 million info.person records
  for each info.person record in the K batch do
    let Y be the person's age
   let N be the number of contacts the person    has
    produce one output record (Y,(N,1))
  repeat
end function

function Reduce is
  input: age (in years) Y
  for each input record (Y,(N,C)) do
    Accumulate in S the sum of N*C
    Accumulate in $C_1$ the sum of C
  repeat
  let A be S/$C_1$
  produce one output record (Y,(A,Cnew))
end function
```

V. EXECUTION OVERVIEW

- The MapReduce in user program splits into n input files, typically 16 to 64 MB.
- It then starts up with various copies of the program on cluster machines.
- One of the copies of program (master copy) is special. The rest of the copies are assign tasks by master.
- A worker machine is assigned a task of map, reads the content, parse key/value pair to user defined map function.
- These buffered pairs are written to local disk regularly.
- Whenever a reduce worker is notified by the master about the storage location, it uses remote procedure call (RPC) to read data from the local disk.
- The reduce worker iterates over stored intermediate data and for each intermediate unique key, it passes it to corresponding intermediate value to user's reduce function.
- The output reduce function is appended to final output file of reduce partition.

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 4, April 2017

- When all map and reduce task have been completed, the master wakes up the user program. At this point MapReduce call in the user program returns back to user code.

VI. MASTER DATA STRUCTURE

It keeps a proper track of various data structures. For every map and reduce task it keeps track of state i.e. ideal, in progress or completed, and for non- ideal tasks it keeps track of identity of worker machine.

VII. REFINEMENTS

Generally, the basic functions provided by the map and reduce functions is sufficient for full filling needs.
But some useful extensions also exists as:
1. User specified portioning function, it determines mapping of intermediate key values to the R reduce shards.
2. Ordering guarantees, its implementation guarantees that within each of R reduce partitions, the intermediate key value pairs are processed in increasing key order.
3. User specified combiner functions for doing partial combination of generated intermediate values with the same key within the same map task (to reduce the amount of intermediate data that must be transferred across the network).
4. Custom input and output types, for reading new input formats and producing new output formats.
5. A mode for execution on a single machine for simplifying debugging a small scale testing. [1]

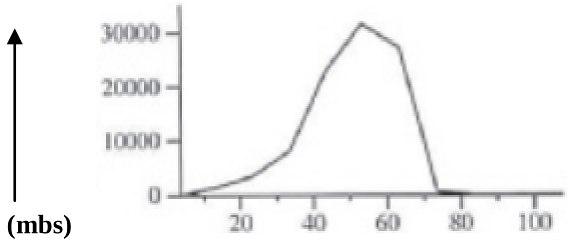
VIII.PERFORMANCE

Here we are measuring the performance of MapReduce based on two computations: Searching a data of one terabyte and searching a particular pattern. The other computation is about sorting of this one terabyte data.
The performance can be estimated on focusing these two:
- **Clustering Configuration**:

All programmers are executed on a cluster that consists of approximately 1800 machines. Each machine has two 2 GHz Intell Xeon processor with Hyper-Threading enabled, 4 GB of memory, two 160 GB IDE disks, and a gigabit Ethernet link. These machines were arranged in two level tree shaped switched network with approximately 100-200 Gbps of aggregate bandwidth available at the root.

1. **Grep** :

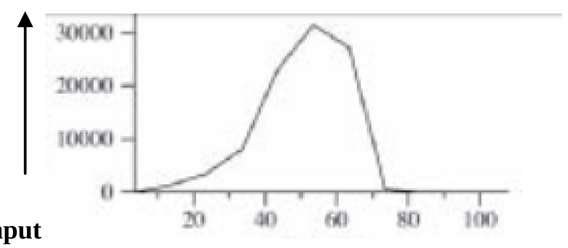
The grep program scans through 10^{10} 100 bytes records, searching for a relatively rare three character pattern (pattern occurs in 92,337 records).
The input is split into approximately 64 MB pieces (M=1500) and the entire output is placed on one file (R=1).



Input (mbs)
Seconds
Data transfer rate over time (mr grep)

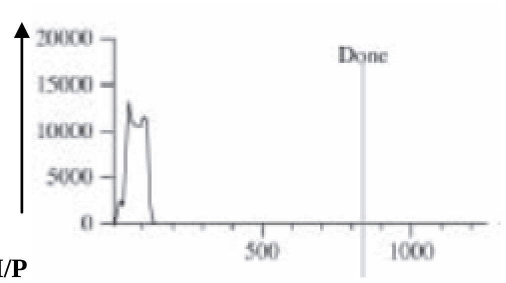
2. **Sort**:

The sort program sorts 10^{10} 100 byte records (approximately 1 terabyte data). The program is modeled after the Terasort benchmark. [3]

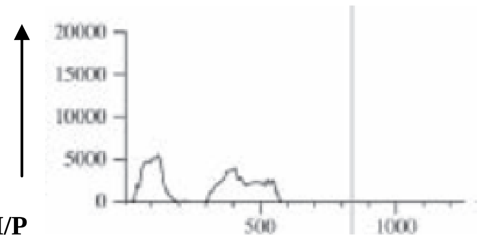


Input
Seconds
Data transfer rate over time

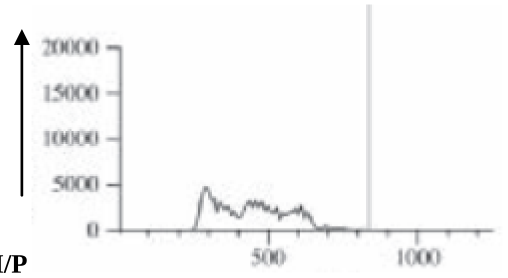
The sorting program consist of less than 50 lines of user code. The final stored output is written to a set of 2- way replicated GFS file (i.e. 2 terabytes are written as the output of the program).



I/P
Seconds



I/P
Seconds



I/P
Seconds

IX. CONCLUSION

The MapReduce programming model has been successfully used for solving the big data problems over distributed network consisting of 1000's of machines.
We attribute this success to several reasons:
First, the model is very easy to use and implement, even by the programmers without the knowledge and experience of parallel and distributed system.
Second, different varieties of problems are easily expressible as MapReduce computations.
Third, MapReduce can also be used for different types of analysis in industries which belongs from different backgrounds ex. Social networking, ecommerce, medical, telecom etc.

REFERENCES

1. Dean, J. and Ghemawat, S.2004. MapReduce simplified data processing on large clusters. In Proceedings of Operating Systems Design and Implementation (OSDI), San Francisco, CA 137-150.
2. Chv, C-T, Kim, S.K, Lin, Y.A, Yu,Y.,Bradski, G, Ng A., and Olukotun k 2006. MapReduce for machine learning on multicore. In Proceedings of Neural Information Processing System Conference (NIPS). Vancouver, Canada.
3. Gray, J.sort benchmark home page. http:// research.microsoft.com/bare/sortbenchmarks/
4. Hadoop In Real World (Hadoop Starter Kit).
5. Ranger, C., Raghuraman, R., Penmesta,A., Bradski, G., and Kozyrakis, C.2007. Evaluating MapReduce for multicore and multiprocessor systems. In Proceedings of 13th International Symposium on High-Performance computer Architecture (HPCA). Phoenix, AZ.
6. Dean, J. and Ghemawat, S.2004. MapReduce simplified data processing on large clusters. In Proceedings of Operating Systems Design and Implementation (OSDI), San Francisco, CA 137-150.