Implementation of Map-Reduce on Hadoop on Public Cloud

Shahbaj khan[1] Ms. Rachna Dasondhi[2]
[1]Research Scholar [2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1,2]Jawaharlal Institute of Technology, Borawan M.P. India

Abstract— As a result of the rapid development in cloud computing, it's fundamental to investigate the performance of extraordinary Hadoop MapReduce purposes and to realize the performance bottleneck in a cloud cluster that contributes to higher or diminish performance. It is usually primary to research the underlying hardware in cloud cluster servers to permit the optimization of program and hardware to achieve the highest performance feasible. Hadoop is founded on MapReduce, which is among the most popular programming items for huge knowledge analysis in a parallel computing environment. In this paper, we reward a particular efficiency analysis, characterization, and evaluation of Hadoop MapReduce WordCount utility.

Key words: Performance Analysis, Cloud Computing, Hadoop Word count

I. Introduction

Cloud computing is based on five attributes: multi-tenancy (shared resources), massive scalability, elasticity, pay as you go, and self-provisioning of resources, it makes new advances in processors, Virtualization technology, disk storage, broadband Internet connection, and fast, inexpensive servers have combined to make the cloud a more compelling solution.

The main attributes of cloud computing are illustrated as follows [4]:

A. *Multi-Tenancy (Shared Resources)*

Cloud computing is based on a business model in which resources are shared (i.e., multiple users use the same resource) at the network level, host level, and application level.

1) Massive scalability: Cloud computing provides the ability to scale to tens of thousands of systems, as well as the ability to massively scale bandwidth and storage space
2) Elasticity: Users can rapidly increase and decrease their computing resources as needed.
3) Pay as you used: Users to pay for only the resources they actually use and for only the time they require them.
4) Self-provisioning of resources: Users self-provision resources, such as additional systems (processing capability, software, storage) and network resources.
5) Cloud computing can be confused with distributed system, grid computing, utility computing, service oriented architecture, web application, web 2.0, broadband network, the browser as a platform, Virtualization, and free/open software

Hadoop is an open source framework from Apache and is used to store process and analyze data, which are very huge in volume. Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel with others. In short, Hadoop is used to develop applications that could perform complete statistical analysis on huge amounts of data.

B. *Hadoop Architecture*

At its core, Hadoop has two major layers namely:

– Processing/Computation layer (MapReduce),
– Storage layer (Hadoop Distributed File System)

C. *MapReduce*

To take the advantage of parallel processing of Hadoop, the query must be in MapReduce form. The MapReduce is a paradigm, which has two phases, the mapper phase and the reducer phase. In the Mapper the input is given in the form of key value pair. The output of the mapper is fed to the reducer as input. The reducer runs only after the mapper is over. The reducer too takes input in key value format and the output of reducer is final output.

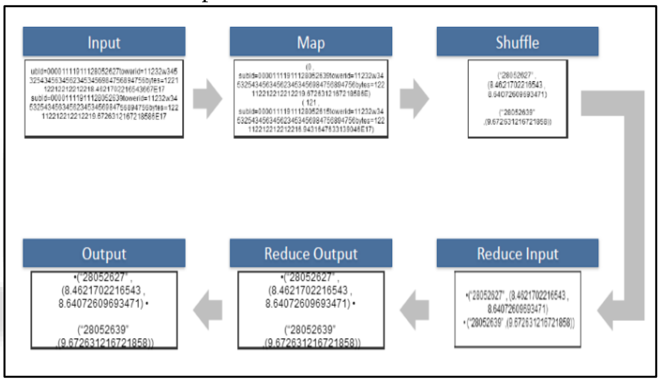


Fig. 1: Map Reduce Flow Diagram

II. Proposed Work

Word count is typical examples where Hadoop map reduce developers start their hands on. This sample map reduce is intended to count the no of occurrences of each word in the provided input files.

The word count operation takes place in two stages a mapper phase and a reducer phase. In mapper phase first the test is tokenized into words then we form a key value pair with these words where the key being the word itself and value ‘1’. For example consider the sentence “tringtring the phone rings”

In map phase the sentence would be split as words and form the initial key value pair as

<tring,1>
<tring,1>
<the,1>
<phone,1>
<rings,1>

In the reduce phase the keys are grouped together and the values for similar keys are added. So here there are only one pair of similar keys ‘tring’ the values for these keys would be added so the out put key value pairs would be

<tring,2>

<the,1>
<phone,1>
<rings,1>

This would give the number of occurrence of each word in the input. Thus reduce forms an aggregation phase for keys.

The point to be noted here is that first the mapper class executes completely on the entire data set splitting the words and forming the initial key value pairs. Only after this entire process is completed the reducer starts. Say if we have a total of 10 lines in our input files combined together, first the 10 lines are tokenized and key value pairs are formed in parallel, only after this the aggregation/ reducer would start its operation.

The Fig. below would throw more light to your understanding

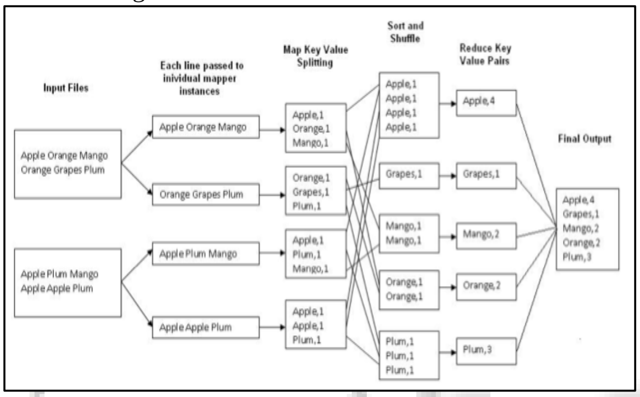


Fig. 2. Proposed System using Map Reduce

MapReduce Algorithm uses the following three main steps:

- Map Function
- Shuffle Function
- Reduce Function

A. *Map Function*

Map Function is the first step in MapReduce Algorithm. It takes input tasks and divides them into smaller sub-tasks. Then perform required computation on each sub-task in parallel.

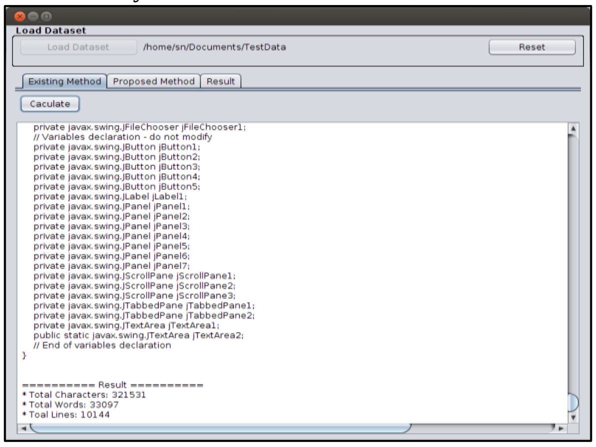
This step performs the following two sub-steps:

- Splitting
- Mapping

```
Algorithm Mapper()
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
    Text outputKey = new Text(word.toUpperCase().trim());
IntWritableoutputValue = new IntWritable(1);
con.write(outputKey, outputValue);
}
}
Algorithm Reducer()
{
int sum = 0;
for(IntWritable value : values)
  {
sum += value.get();
  }
con.write(word, new IntWritable(sum));
}
}
```

III. Result Analysis

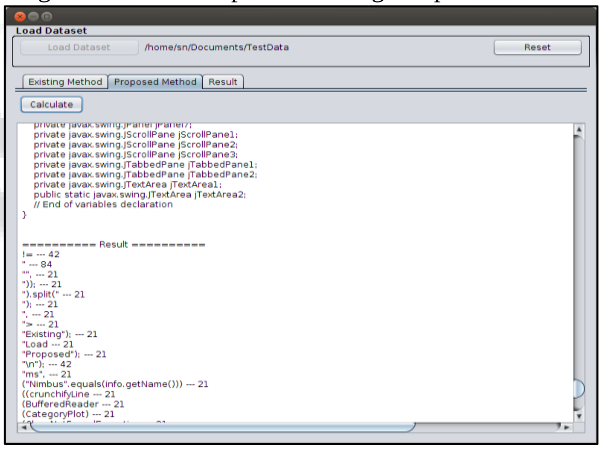
Existing and proposed system implemented on Ubuntu 14.10 Server edition. First install and conFig. jdk1.8 on machine. After that install Hadoop 2.7 and conFig. it. NetBeans 8.0 used as editor and creates Graphical User Interface for project. Compare existing and proposed on the basis of computation time. Below Fig.s show GUI and comparison between both systems.



```
    private javax.swing.JFileChooser jFileChooser1;
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JPanel jPanel4;
    private javax.swing.JPanel jPanel5;
    private javax.swing.JPanel jPanel6;
    private javax.swing.JPanel jPanel7;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTabbedPane jTabbedPane2;
    private javax.swing.JTextArea jTextArea1;
    public static javax.swing.JTextArea jTextArea2;
    // End of variables declaration
}


========== Result ==========
* Total Characters: 321531
* Total Words: 33097
* Toal Lines: 10144
```

Fig. 3: Word Count problem using Simple Java Code



```
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTabbedPane jTabbedPane2;
    private javax.swing.JTextArea jTextArea1;
    public static javax.swing.JTextArea jTextArea2;
    // End of variables declaration
}


========== Result ==========
!= --- 42
" --- 84
"", --- 21
")); --- 21
").split(" --- 21
"); --- 21
", --- 21
"> --- 21
"Existing"); --- 21
"Load --- 21
"Proposed"); --- 21
"\n"); --- 42
"ms", --- 21
("Nimbus".equals(info.getName())) --- 21
((crunchifyLine --- 21
(BufferedReader --- 21
(CategoryPlot) --- 21
```

Fig. 4: Word Count problem using Map Reduce

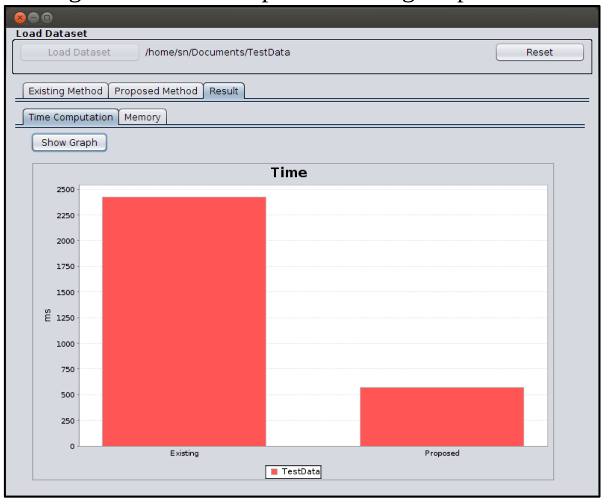


Fig. 5. Computation time chart for existing and proposed system.

We evaluate our proposed system on different parameters, which describe below:

– Execution
– Time Memory

	Existing System	Proposed System
Execution Time (ms)	2280	630
Memory (MB)	130	107

Table 1: Parameter Comparisons

1) Execution Time

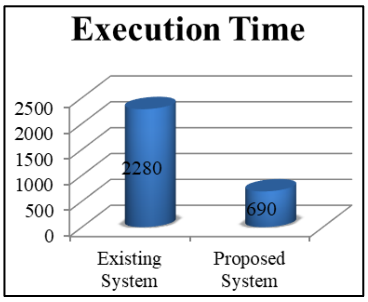


Fig. 6. Execution time graph for Exiting System and Proposed System

We calculate Execution Time value for all algorithms Exiting System and Proposed System. And results shown with help of diagram. We find that Proposed system shows less time compare to Existing System.

B. Memory:

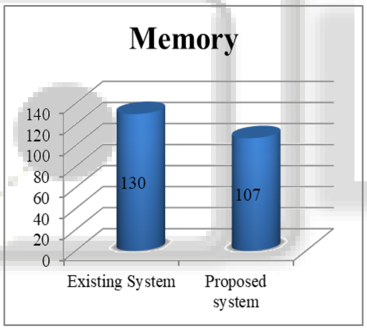


Fig. 7. Memory graph for Exiting System and Proposed System

We calculate Execution Memory value for all algorithms Exiting System and Proposed System. And results shown with help of diagram. We find that Proposed system shows less memory compare to Existing System.

IV. Conclusion

Map-Reduce has become an important platform for a variety of data processing applications. Word CountMechanisms in Map-Reduce frameworks such as Hadoop, suffer from performance degradations in the presence of faults. Word Count Map-Reduce, proposed in this paper provides an online, on-demand and closed-loop solution to managing these faults. The control loop in word count mitigates performance penalties through early detection of anomalous conditions on slave nodes. Anomaly detection is performed through a novel sparse-coding based method that achieves high true positive and true negative rates and can be trained using only normal class (or anomaly-free) data. The local, decentralized nature of the sparse-coding models ensures minimal computational overhead and enables usage in both homogeneous and heterogeneous Map-Reduce environments.

References

[1] Samneet Singh and Yan Liu,"A Cloud Service Architecture for Analyzing Big Monitoring Data",ISSNll1007-0214ll05/10llpp55-70 Volume 21, Number 1, February 2016
[2] JOSEPH A. ISSA, "Performance Evaluation and Estimation Model Using Regression Method for Hadoop WordCount", Received November 19, 2015, accepted December 12, 2015, date of publication December 18, 2015, date of current version December 29, 2015.
[3] Yaxiong Zhao, Jie Wu, and Cong Liu, "Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework", ISSNll10070214ll05/ 10llpp39-50 Volume 19, Number 1, February 2014
[4] Zhuoyao Zhang LudmilaCherkasova, "Benchmarking Approach for Designing a MapReduce Performance Model", ICPE'13, April 21-24, 2013
[5] NikzadBabaiiRizvandi, Albert Y. Zomaya, Ali JavadzadehBoloori, Javid Taheri1, "On Modeling Dependency between MapReduce Configuration Parameters and Total Execution Time", 2012
[6] NikzadBabaiiRizvandi, Javid Taheri1, Reza Moraveji, Albert Y. Zomaya, "On Modelling and Prediction of Total CPU Usage for Applications in MapReduce Enviornments", 2011
[7] A. Baratloo, M. Karaul, Z. Kedem, and P.Wyckoff, ``Charlotte: Meta computing on theWeb,'' in Proc. 9th Int. Conf. Parallel Distrib. Comput. Syst., 1996, pp. 1_13.
[8] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny, ``Explicit control in the batch-aware distributed _le system,'' in Proc. 1st USENIX Symp. Netw. Syst. Design Implement. (NSDI), Mar. 2004, pp. 365_378.
[9] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier,``Cluster-based scalable network services,'' in Proc. 16th ACMSymp. Oper.Syst. Principles, Saint-Malo, France, 1997, pp. 78_91.
[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, ``The Google _le system,'' in Proc. 19th Symp. Oper. Syst. Principles, New York, NY, USA, 2003, pp. 29_43.
[11] S. Ibrahim, H. Jin, L. Lu, L. Qi, S.Wu, and X. Shi, ``Evaluating MapReduce on virtual machines: The Hadoop case,'' in Proc. Int. Conf. Cloud Comput., vol. 5931. 2009, pp. 519_528.
[12] J. Issa and S. Figueira, ``Graphics performance analysis using Amdahl's law: IEEE/SCS SPECTS,'' in Proc. Int. Symp. Perform. Eval. Comput.Telecommun. Syst., Ottawa, ON, Canada, 2010, pp. 127_232.