An Enhanced Map Reduce Framework for Improving the Performance of Massively Scalable Private Clouds

Ashutosh Rajan
Amrita School of Arts and Sciences,Kochi,
Amrita Vishwa Vidyapeetham,

M. V. Judy, Ph.D
Amrita School of Arts and Sciences, Kochi,
Amrita Vishwa Vidyapeetham

ABSTRACT
Cloud Computing systems provide access to large amount of data and other resources through a large number of interfaces. Apache Hadoop is a framework that allows distributed processing of large sets of data across cluster of computers. It is a powerful abstraction proposed for making scalable and fault tolerant applications. In this paper we have suggested an enhanced framework for MapReduce which increased the performance of the Private Clouds in distributed environment. In this framework a separate thread is maintained for each and every Mapper and a single buffer is used for retrieving all threads. A single Buffer retrieves all records. At this instance a separate thread can search for all the records with same key in the buffer and pass it on to the Reduce function which can executed. A multimap is used to access partial result while maintain key ordering. Our analysis shows that better performance can be achieved using this enhanced Map Reduce framework than using the traditional MapReduce framework. The results show the reduction in job completion time when compared with existing one.

1. INTRODUCTION
The growth of internet has pushed researchers from all disciplines to deal with volumes of information where the only viable path is to utilize data intensive frameworks [4]. Cloud Computing is a successful paradigm of service oriented architecture. It has revolutionized the infrastructure which is abstracted and used. Elasticity, pay-per-use, low time to market and transfer of risk are the major factors which makes cloud suitable for deploying applications which are infeasible in traditional enterprise architecture. Hadoop [3] is used to improve the performance of data intensive systems.

With the increase in power of processors every year, the data to be processed is also increasing with a faster rate. In such a situation rather than trying to improve the complexity of algorithms which has got a saturation point or upgrading the processor each and every time, the performance can be improved by enhancing the parallel processing framework.

In this paper the Map Reduce algorithm is used to solve problems with large size on a private cloud and its scalability demonstrated. The enhanced Map Reduce removes the barriers of the traditional Map Reduce and its performance improvement is shown.

2. PRIVATE CLOUD
Cloud computing [6] is a computing model where resources such as computation power, storage, network and software are abstracted and provided over internet in remotely accessible fashion. Cloud has got following service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud can be divided into following types: Public Cloud, Community, Private Cloud [1] and Hybrid Cloud.

Private Cloud is an infrastructure which operates solely for a private organization which is managed internally.

3. MAPREDUCE
Inspired from map and reduce of functional programming Hadoop proposed open source implementation of Google MapReduce [2]. MapReduce is an abstraction which provides the user with capability to build large scale distributed applications easily. MapReduce easily parallelizes the computations as invocation of map function is independent and uses re-execution as primary mechanism of fault-tolerance.

In this input is given as a set of key/value pair and also produces a key/value pair as its output. The user of MapReduce library expresses the computation as two functions: Map and Reduce. Map function takes as input a set of key/value pair and generates a set of intermediate key/value pair as its output. MapReduce library groups together all values for a particular key. Until whole data from Map stage is transferred to appropriate machine Reduce function waits. The Reduce function accepts this key and set of values corresponding to that particular key as its inputs. It is the job of reducer to merge these values and provide a smaller set of values. The intermediate values are supplied to user via an Iterator. It allows model to handle large list of values that are too large to fit in main memory.

Logically map and reduce function executes in following manner:

Map (k1, v1) $\rightarrow$ list (k2, v2)

Reduce (k2, list (v2)) $\rightarrow$ list (k3, v3)

MapReduce library splits the input file into M pieces and many copies of programs are started up in cluster of machines and these input splits can be processed in parallel by different machines. Shuffling and sorting takes place. Reduce invocations are distributed by partitioning the intermediate key spaces into R pieces which *hash(key)%R* according to Hadoop configuration.

4. ENHANCED MAPREDUCE FRAMEWORK
MapReduce program is divided into two stages Map and Reduce. Map stage writes the output locally and Reducers aggregates the output by remotely reading from the Mappers. This process of transferring the data is called as Shuffling.

In this Non-conventional MapReduce we device a technique by which we bypass the sorting mechanism and modify the invocation of reduce function so that it can be called with a small set of records. Reducers no longer needs to wait for remotely read from Mappers and then to be grouped. Due to this performance gets improved as now Reducers need not to wait till the Mappers complete their whole work and shuffling gets completed.

In Non-conventional Map Reduce we overcome the following overheads:

1. Waiting time between Remote reading of the first and the last records.
2. Time taken for sorting of whole records.
3. Maintaining a local buffer for each Mapper.

In this Non-conventional Map Reduce the intermediate results are not stored as a whole for each and every key. And Reduce function works at one record at a time.

In traditional Map Reduce shuffle stage is designated to work in an efficient and asynchronous manner by providing a thread for each and every Mapper which reads the data from Mappers and this data is stored in their local Buffers which is then merge sorted. Then the key and corresponding values are passed on to the Reduce function.

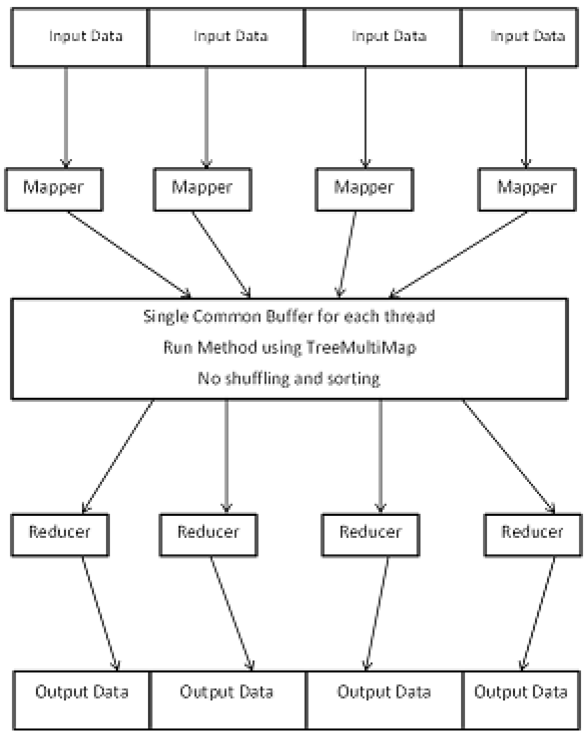

Figure 2: Enhanced Map Reduce Framework

5. ENHANCED MAP REDUCE WORD COUNT ALGORITHM

In this section we modify the traditional Map Reduce to a Non –Conventional Map Reduce.

Algorithm 4: Non-Conventional Map Reduce Mapper

```
map ( key, value )
// key: document name
// value: document contents
for each word in value
        EmitIntermediate (word,1)
End for
```

Algorithm 5: Non-Conventional Map Reduce reducer function

```
reduce (key, value, context)
//key: word
//value: a list of counts
result =0
for each v in value
        result=result+v
End for
Insert(key,value) in TreeMultiMap
```

Algorithm 6: Non-Conventional Map Reduce Run function

```
run ()
create a new TreeMultiMap
while context has more keys
        key= current key from context
        value=current values from context

        if TreeMultiMap doesn't contain a
        particular key then

        Insert (key,0)
        End if
reduce(key, value, context)
/* After all reduce invocations have been done*/
```

6. RESULTS

Word Count Problem is a simple problem where frequency of occurrences of each word is counted by the algorithm in "n" documents. We implemented this simple problem on Hadoop (0.20) and executed it on Intel core i3 machines with 4 GB RAM and 500 GB Hard disk. Private Cloud was formed with three machines and Hadoop was configured on this Private Cloud.

Then we implemented the same Word Count problem with Non-Conventional Map Reduce and again counted the performance of the algorithm we found that Non-Conventional Map Reduce performed better than Conventional one as it reduced the effort which was there while shuffling was done. Detailed description on Private Cloud setup can be found from [1]. Apache Hadoop [2][3][10] setup description can be found from [10][11].

Scenario 1

In first scenario we tested the word count algorithm and its performance on the Private Cloud using Conventional Map Reduce and Non-Conventional Map Reduce showing that performance of Non-Conventional Map Reduce was better than Conventional Map Reduce. Performance Increase was of nearby 20%.

Total word count = Total number of words/Total time taken for execution

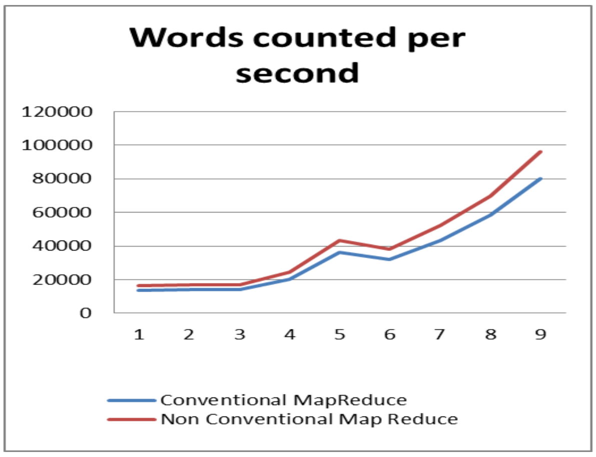

Figure 2 Total number of words counted in a given time period.

Scenario 2:
In second case we kept the number of words constant and time taken for counting frequency of all words in all documents given as input was measured. This test was also performed using both Conventional and Non-Conventional Map Reduce. We tested this with about 8 GB of data.

Our results showed that Non-Conventional Map Reduce performed better than Conventional Map Reduce.

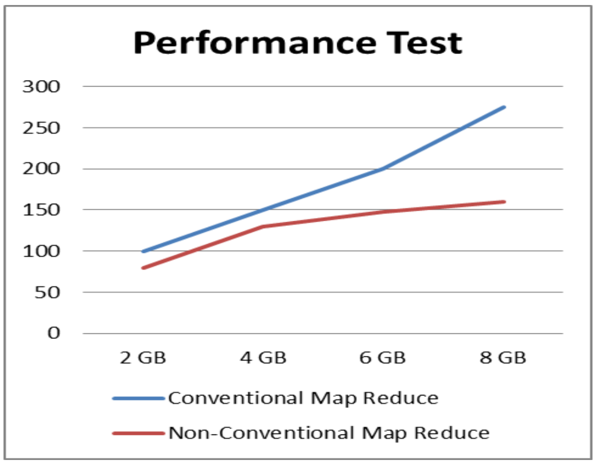

Figure 3 Total time taken for counting words.

7. DISCUSSION OF RELATED WORKS

MapReduce [8] is a programming model that enables users to easily develop large-scale distributed applications. Apache Hadoop is the open source implementation of MapReduce model whereas Google MapReduce is a proprietary version of MapReduce. We tried to change the Conventional Map Reduce to improve the performance which we called Non-Conventional Map Reduce. Several different implementations of Hadoop for multicores like Phoenix [12]. CGL-MapReduce [13] for streaming applications are available.

8. CONCLUSION AND FUTURE WORK

In this paper, we have mainly used MapReduce model to show that performance increases when resources are increased. We described how we can design a parallel algorithm using MapReduce model on Hadoop and then we modified the original Map Reduce model to non-Conventional Map Reduce and its performance is analyzed. Performance of the algorithm on Non-Conventional Map Reduce was near about 20% better. It also depicts that on adding more resources we would be able to solve larger problems and Non-Conventional Map Reduce is better than the original Map Reduce. Memory management may be a problem which arises for large sets of data as partial results obtained after the Map stage is stored in memory only. This could result in overflow. A solution suggested to this problem is moving the contents which is least recently used into files. A Hash table is used to keep the track of files which have been moved on to the file and for faster access. We can try to implement Non-Conventional MapReduce model on GPUs that is our compute intensive tasks Map on GPUs and Reduce on CPUs.

9. REFERENCES

[1] Private Cloud setup http://www.akashsharma.me/private-cloud-setup-using-eucalyptus-and-xen/

[2] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107–113,2008.

[3] The Apache Hadoop Project. http://hadoop.apache.org/core/, 2009.

[4] Data Intensive applications http://en.wikipedia.org/wiki/Data_Intensive_Computing#MapReduce

[5] Fundamentals of cloud computing http://www.cse.fau.edu/~borko/HandbookofCloudComputing.

[6] Cloud computing http://en.wikipedia.org/wiki/Cloud_computing

[7] Eucalyptus Beginner's Guide UEC Edition by Johnson D, Kiran Murari, Murthy Raju, Suseendran RB, Yogesh Girikumar.

[8] Apache Hadoop Map Reduce http://hadoop.apache.org/common/docs/current/mapred_tutorial.html#MapReduce+- +User+Interfaces

[9] Apache Hadoop Distributed File System http://hadoop.apache.org/common/docs/current/hdfs_design.html

[10] Running Hadoop on single node environment http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[11] Running Hadoop on multi node environment http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

[12] R. Raghu Raman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. Proceedings of the 2007 IEEE 13th International Symposium

[13] Breaking the Map Reduce stage Barriers

[14] Abhishek Verma, Nicolas Zea, Brian Cho, Indranil Gupta, Roy H. Campbell University of Illinois at Urbana-Champaign