Bigdata Analytics Using RESTful Web Services

[1]Vimal Lalani [2]Arvind Meniya
[1]ME student, Department of Information Technology, SSEC, Bhavnagar, Gujarat, India
[2] Assistant Professor, Department of Information Technology, SSEC, Bhavnagar, Gujarat,India

Abstract- **We have entered an era of Big Data. Through better analysis of the large volumes of data that are becoming available, there is the potential for making faster advances in many scientific disciplines and improving the profitability and success of many enterprises. There are many challenges include not just the obvious issues of scale, but also heterogeneity, lack of structure, error-handling, privacy, timeliness, provenance, and visualization, at all stages of the analysis pipeline from data acquisition to result interpretation. These technical challenges are common across a large variety of application domains, and therefore not cost-effective to address in the context of one domain alone.**

Hadoop provides solution for above problem but by analyzing the solution we identify some issues. These issues are defined in this report. In this report solution of the issue and better solution with restful web service is purposed. One of the advantage of RestFul Web services is heterogeneity and scalability with various platform is utilized in this paper. Moreover improving MapReduce technique in shuffle phase leads to desirable results. In short in this paper we define architecture for Big Data analytics using RestFul Web Services

***Index Terms*— Big Data Analytic architecture , restful service with big data , improve shuffling**

I. Introduction

Human beings now create 2.5 quintillion bytes of data per day. The rate of data creation has increased so much that 90% of the data in the world today has been created in the last two years alone.[2] This acceleration in the production of information has created a need for new technologies to analyze massive data sets. The urgency for collaborative research on Big Data topics is underscored by the U.S. federal government's recent $200 million funding initiative to support Big Data research.[3]

The term *Big Data* refers to large-scale information management and analysis technologies that exceed the capability of traditional data processing technologies.[1] Big Data is differentiated from traditional technologies in three ways: the amount of data (volume), the rate of data generation and transmission (velocity), and the types of structured and unstructured data (variety) (Laney, 2001)

Variety: Variety makes big data really big. Big data comes from a great variety of sources and generally has in three types: structured, semi structured and unstructured. Structured data inserts a data warehouse already tagged and easily sorted but unstructured data is random and difficult to analyze. Semi-structured data does not conform to fixed fields but contains tags to separate data elements [7, 9]. *Volume:* Volume or the size of data now is larger than terabytes and petabytes. The grand scale and rise of data outstrips traditional store and analysis techniques [4, 5]. *Velocity :* Velocity is required not only for big data, but also all processes. For time limited processes, big data should be used as it streams into the organization in order to maximize its value [9,10].

II. Background: Big Data Analytic with Hadoop

Hadoop's implementation of MapReduce closely resembles Google's[2] . There is a single *master* managing a number of *slaves*. The input file, which resides on a distributed filesystem throughout the cluster, is split into even-sized *chunks* replicated for fault-tolerance. Hadoop divides each MapReduce job into a set of *tasks*. Each chunk of input is first processed by a *map* task, which outputs a list of key-value pairs generated by a user-defined map function. Map outputs are split into buckets based on key. When all maps have finished, *reduce* tasks apply a reduce function to the list of map outputs with each key. Figure 1 illustrates a MapReduce computation..

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks.[1]

MapReduce process consist of following phase

Splitting
Mapping
Shuffling
Sorting
Reducing

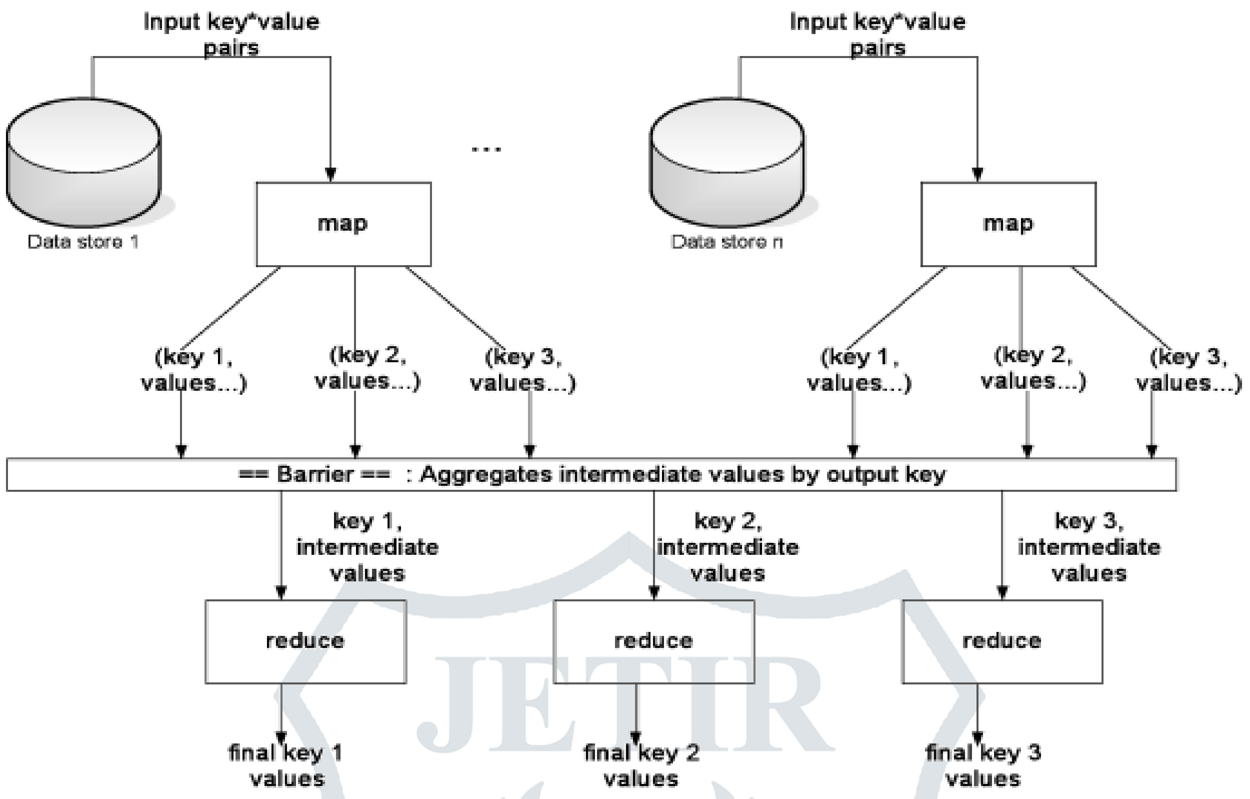

Figure 1: A MapReduce computation. Image from[3]

The mapper does not directly write to the disk rather it takes advantage of buffering the writes. Each mapper has a circular memory buffer with a default size of 100MB which can be tweaked by changing the io.sort.mb property. It does the flush in a very smart manner. When the buffer is filled up to a certain threshold (default of 80%- can be changed by tweaking the io.sort.spill.percent property) , a separate thread gets triggered which spills the content in the buffer to the disk. Before the spill happens to the disk, the thread (which is entrusted with the task of performing the spill) partitions the data according to the reducers it needs to go to and a background thread performs an in-memory sort within the partition based on the key. If a combiner is present, it consumes the output of the in-memory sort. There may be several spill files which get generated as a part of the above process, hence at the end of the map phase an on-disk merge is performed to form bigger partitions (bigger in size and less in number - number depends on the number of reducers) and the sorting order is taken care of during the merge process. Now, the output of the several map tasks is sitting on different nodes and it needs to get copied over to the node on which the reducer is going to run in order to consume the output of the map tasks. If the data from the map tasks is able to fit inside the reducer's task tracker memory, then an in-memory merge is performed of the sorted map output files coming from different nodes. As soon as a threshold is reached, the merged output is written onto the disk and the process repeated till all the map tasks have been accounted for this reducer's partition. Then, an on-disk merge is performed in groups of files and a final group of files is directly feeded into the reducer performing an in-memory merge while feeding (thus saving an extra trip to the disk). From the final merge (which was a mixture of an in-memory and on-disk merge) , the data is fed to the reduce phase which may optionally perform some further processing and finally the data is written to HDFS.

III. Big Data Analytics Using RESTful Web Services

Apache Hadoop provides a high performance native protocol for accessing HDFS. While this is great for Hadoop applications running inside a Hadoop cluster, users often want to connect to HDFS from the outside. For examples, some applications have to load data in and out of the cluster, or to interact with the data stored in HDFS from the outside. Of course they can do this using the native HDFS protocol but that means installing Hadoop and a Java binding with those applications. To address this we have developed a concept for Big Data analytics using an industry standard RESTful mechanism.
Hadoop MapReduce framework uses shuffling phase which needs to merge and then short all the generated keys at one place which cause problem if the generated keys are large. So merging and shorting process should not conducted at single place.
It is also difficult and costly to add data-nodes of various platform.
In purpose “Big Data Analytics using RESTful Services” method we are going to provide such architecture which is helpful to overcome above two problems.

IV. Architecture of Big Data Analytic using RESTful Web Service

Combine Algorithm of NameNode and DataNode

1. BroadCast code message

```
        2. Execute Map method at each datanode simultaneously
        3. Waits until all finish Map
        4. For each datanode call
             BroadCast service of DataNode
        5. Execute Reduce and combine reduce to transfer client
```

Algorithm for client

```
    1. Ask namenode for writing
    2. Get the write service URL from namenode
    3. Perform write to datanode
        i)   Perform till data line available
        ii)  Get N number data line with max length of line < M , N is total no of available data node
        iii) Create N threads to call write service
    4. Put code for Map method and Reduce method
```

Detail Algorithm for DataNode and NameNode

```
Class DataLine {
String : dataline ;
Int: Index}
```

BroadCast Service implemented by DataNode

```
        1. Prepare Key-Node list
        2. For each dataNode call
             Send( ownNodeId, key-nodeList, keyList_value) ,{keyList value is for receiver node}
        3. Remove from Key-ValueList where key-value = key value of receiver
        4. Add key-list_values returned from service to key-list value
```

Send Service implemented by DataNode

```
        1. Update Key-Node_List
        2. Search keys for nodeId in key_List value
        3. Return key-list value from above step
        4. Add key-List value(third-parameter) to own key-list-valueList
```

Prepare key-Node List Method

```
        1. Tempkeys ← find those keys from key-list value which are not in key-node list
        2. For each tempkey in tempkeys
           Add key-Node (tempkey , DataNodeList.Next())
      3. tempkeys ← find keys for receiver
        4. key-list_value for receiver ← select key-value list where keys = tempkeys
```

Note: DataNodeList.Next() should maintain in circular manner

Reduce task is defined by client

```
        1. DataNode will get data in (key-valueList)List format
```

Reduce task is combined at one place after reduce call
Mapping task is defined by client
Prepare List of key

How above algorithm works ?

Above map-reduce process works on three main phase. In first phase client writes data on datanode. To write data client get the Rest web service url from main name node and call write services of all datanodes with multiple threads simultaneously. By this way multi channel writing can be possible. After wring data file and code to datanode , namenode call map service of all datanode. And waits for all complete their mapping work. In second phase ,after completion of all datanode , namenode call broadcast service which is responsible to generate key-valuelist pair and transfer them to other node by send service call, in sequential manner. In last phase reducer are ready to reducing work. Namenode call reduce service of all the datanode parallel.

Our focus was on second phase. After completion of mapping task by each node they converts their key-value pair in to key-valuelist pair. And then waits for the broadcast cal from the name node. When name node broadcast service call occurs then datanode divides its list of key-valueslist by total number of reducer(datanode which will performs the reducing tasks) and send equal number list element of list of key-valuelist to them. And same metadata information is sent to other datanode. Here we shoe the algorithm for send service call. We can show that after completion of first broadcast message we can able to send n^{th} output of mapper to reducer. Here we can not need to shart maps output as well as to merge them on single place. The subsequence call of send call from remaining datanode will dramatically reduce the cost of communication as in subsequence call datanode only need to distribute only unkown key-valuelist which are not fetched on pervious nodes. By n/2 call of send service call all reducer get their key-valuelist data.

Above arrangement uses JSON data to transfer across nodes. Here DataLine object is also mention to write data from client to datanode. This object contains Index property which helps to maintain recovery management schemes. Initial experiment shows that to mapping task takes 20 sec to maps 100 Kb file in single node with .NET environment. But the reducer task takes generally 200 ms. Overall performance can be improved by adopting faster approach of exacting data from file to main memory. Currently major time consume on getting data from txt file to main memory.

V. Conclusion

As more and more data are collected, the analysis of these data requires scalable, flexible, and high-performing tools to provide analysis and insight in a timely fashion. The outcome of this research is in the form of a framework for Big Data analytics using Restful web service. The main focus of this framework is to provide big data analytic tool which provides better scalability of component interaction, more generality of interfaces and independent deployment of component.

References

[1] http://gartner.com/it-glossary/big-data/

[2] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008

[3] Figure from slide deck on MapReduce from Google academic cluster, http://tinyurl.com/4zl6f5.Available under Creative Commons Attribution 2.5 License.

[4] http://www-01.ibm.com/software/data/bigdata

[5] http://www.nytimes.com/2012/03/29/technology/new-us-research-will-aim-at-flood-of digital-data.html

[6] Hadoop Map-Reduce flow, http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[7] C. Eaton, D. Deroos, T. Deutsch, G. Lapis and P.C.Zikopoulos, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, Mc Graw-Hill Companies, 978-0-07-179053-6, 2012.

[8] S. Madden,"From Databases to Big Data", IEEE Internet Computing, v.16, pp.4-6, June 2012.

[9] S. Singh and N. Singh, "Big Data Analytics", 2012 International Conference on Communication, Information & Computing Technology Mumbai India, IEEE, October 2011.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, 51 (1): 107-113, 2008