A System for Large-Scale Graph Processing on the Concept of Map-Reduce and MRBG

Mr. Nilesh Vasant Kurhade  
PG Student  
Department of Computer Engineering  
SPCOE, Otur, Pune.

Prof. Snadip Kahate  
Assistant Professor  
Department of Computer Engineering  
SPCOE, Otur, Pune.

Abstract

A time-line based framework for topic MapReduce programming model is widely used for large scale and one-time data intensive distributed computing, but lacks flexibility and efficiency of processing small incremental data. Incremental Mapreduce framework is proposed for incrementally processing new data of a large data set, which takes state as implicit input and combines it with new data. Map tasks are created according to new splits instead of entire splits while reduce tasks fetch their inputs including the state and the intermediate results of new map tasks from Preserved and latest generated result. The preserved states really producing the promising result and significantly reduce the run time for refreshing big data mining results compared to re-calculating on both simple and multi stage MapReduce. The intermediate states are saved in the form of kv-pair level data and data dependence in a MapReduce computation as a bipartite graph, called MRBGraph. A MRBG-Store is designed to preserve the fine-grain states in the MRBGraph and support efficient queries to retrieve fine-grain states for incremental processing.

Keywords: CAN Map, Reduce, key value, MRBG

I. INTRODUCTION

Now a day’s large amount of data has been generated in number of areas like e-commerce, social networks, education to process such a large amount of data there are number of frameworks are get designed and used for analysis. The Hadoop MapReduce is one the finest framework widely used to carry out such a cumbersome task. In another case there is always problem of refreshing such large amount of data to keep result up to date and accurate. Incremental approach is really the good solution to keep result fresh and accurate. The main target of the incremental approach is to avoid the recalculation hence increase system performance. The task having the same processing or the calculation are getting carryout only once and get stored for further use. Proposed work actually provides the extension to the MapReduce by providing the fine grain approach and incremental processing.

Fig. 1: Simple Map Reduce processing

A. Map Reduce Bipartite graph Abstraction and storage Phase and Storage Phase:

The desired system is really promising and one of the most important reasons is intermediate storage phase, MapReduce Bipartite graph is used to carry out this task due to its storing and processing capacity. Each vertex act as map phase and edge between the vertices are the logic behind the reduction. Mapping is done on each instance \( \{k, v\} \) pair and generated output are act as new \( \{k1, v1\} \) pair for upcoming map phase and reduction is done on \( \langle v3, \{K3\} \rangle \) pair generated in third phase of mapping.
and so on. The derived edges of the graph are stored as fine grain states and used as preserved MRB Graph. The states are getting stored in the format of i) source side map phase ii) destination side reduces phase and the iii) value of edge. The storage phase is responsible to store the fine grain states which must support incremental mechanism so that it will consider the incremental results and intermediate data provided as input.

B. The desired system has the some important feature as given below:

1) **KV-pair Level Fine Grain Incremental Processing:**
   Here the important part is Mapreduce bipartite graph store concept, stores the intermediate result as well as fine grain states generally used in incremental processing.

2) **Enhanced Iterative Processing:**
The central theme of the desired system is to freedom to add and delete any intermediate state so that any intermediate change are get covered and system become stable irrespective to any sudden change in input. In that part Mapreduce bipartite graph perform the important task.

II. **LITERATURE SURVEY**

Yanfeng Zhang, Shimin Chen, Qiang Wang, and Ge Yu, "i2MapReduce: Incremental MapReduce for Mining Evolving Big Data", VOL. 27, NO. 7, JULY 2015.

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for, in-memory cluster computing. A web service is experiencing errors and an operator wants to search terabytes of logs in the Hadoop filesystem (HDFS) to find the cause. Using Spark, the operator can load just the error messages from the logs into RAM across a set of nodes and query them interactively [2].

R. Power and J. Li, Piccolo: Building fast, distributed programs with partitioned tables, with the increased availability of data centers and cloud platforms, programmers from different problem domains face the task of writing parallel applications that run across many nodes. These application range from machine learning problems (k-means clustering, neural networks training), graph algorithms (PageRank), scientific computation etc. Many of these applications extensively access and mutate shared intermediate state stored in memory [3].

S. R. Mihaylov, Z. G. Ives, and S. Guha, Rex: Recursive, deltabased datacentric computation, Web and social network environments, query workloads include ad hoc and OLAP queries, as well as iterative algorithms that analyze data relationships (e.g., link analysis, clustering, learning). Modern DBMSs support ad hoc and OLAP queries, but most are not robust enough to scale to large clusters. Conversely, cloud platforms like MapReduce execute chains of batch tasks across clusters in an fault tolerant way, but have too much -overhead to support ad hoc queries [5].

S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, Spinning fast iterative data flows, A method to integrate incremental iterations, a form of workset iterations, with parallel dataflows. After showing how to integrate bulk iterations into a dataflow system and its optimizer, presenting an extension to the programming model for incremental iterations. The extension alleviates for the lack of mutable state in dataflow and allows for exploiting the sparse computational dependencies inherent in many iterative algorithms. The evaluation of a prototypical implementation shows that those aspects lead to up to two orders of magnitude speedup in algorithm runtime, when exploited [7].

Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, Haloop: Efficient iterative data processing on large clusters, the growing demand for large-scale data mining and data analysis applications has led both industry and academia to design new types of highly scalable data-intensive computing platforms. We evaluated HaLoop on real queries and real datasets. Compared with Hadoop, on average, HaLoop reduces query runtimes by 1.85, and shuffles only 4 percent of the data between mappers and reducers [8].

J. Ekanayake, H. Li, B. Zhang, T. Gunarathe, S.-H. Bae, J. Qiu, and G. Fox, Twister: A runtime for iterative mapreduce, MapReduce programming model has simplified the implementation of many data parallel applications. the years of experience in applying MapReduce to various scientific applications we identified a set of extensions to the programming model and improvements to its architecture that will expand the applicability of MapReduce to more classes of applications [9].

D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, Naiad: A timely dataflow system, A new computational model, timely dataflow, underlies Naiad and captures opportunities for parallelism across a wide class of algorithms. This model enriches dataflow computation with timestamps that represent logical points in the computation and provide the basis for an efficient, lightweight coordination mechanism [9].

III. **EXISTING SYSTEM**
The normal MapReduce framework is stateless and single stage in processing nature. Due to the same the result are become stale. the whole work is get divided into the two steps like, In map the single large task are get divided into the smaller tasks and distributed for the processing .(based on key/value pair).
IV. Motivation

Current technology, architecture, management and analysis approaches are not fully able to cope with the flood of data, and organizations will need to change the angle of view about, plan, govern, manage, process and report on data to realize the potential of big data. Since the motivation behind the project is that manage big heterogeneous, coming from various autonomous as well as private sources and having critical and evolving relationships data. Fast and updated retrieval of big data which is clustered amongst various clusters has been done by managing and merging the attributes of big data and keep result up to date and accurate.

V. Proposed System Work

The Proposed system stores the intermediate result, and used for further processing. It involve Starting run and MapReduce bipartite Graph preserving Normal MapReduce is got performed and intermediate result get stored in the form of graph.

Delta input the type of input given to the system which is absolutely new and need to process.

Incremental map computation to obtain the delta MapReduce Bipartite Graph. The newly calculated result will get combined with the existing one and finally result will get finalized.

Incremental reduce computational state. In the last stage the reduce function are get applied to optimize the reduced finalized result.

VI. Problem Statement

Big data is continuously and diversely growing field, as the input data vary the input to the processing are get changed and hence result become absolute. To avoid such a situation it is desirable to keep result up to date with the help of incremental iterative processing.

A. Algorithm Used:

Map Phase input: <i, Ni|Ri>
1) output <i, Ni >
2) for all j in Ni do
3) Ri;j= Ri/|Ni|
4) output < j, Ri;j>
5) end for
Reduce Phase input: < j, {Ri;j;Nj }>
6) Rj = d∑iRi;j +( 1 - d )
7) output < j, Nj |Rj>

B. Algorithm: State Storage Algorithm:

Input queried key: k; the list of queried keys: L
Output chunk k
1) if !read_cache.contains (k) then
2) gap <- 0, w <- 0
3) I <- k’s index in L
4) while gap < T and w + gap + length (Li) < read cache size do
5) w<- w + gap + length (Li)
6) gap< pos (Li+1) – pos (Li ) - length(Li )
7) i< i + 1
8) end while
9) starting from pos(k), read w bytes into read_cache
10) end if

VII. EXPERIMENTAL RESULTS

For performing experiments Hadoop 0.20.3 is modified to include incremental processing, such that map-reduce programmers can take advantage of this framework. The Hadoop processing architecture consists of 3 data nodes and 1 namenode. The data node is also responsible in processing in this setup. All the machines having 2.4 GHz of CPU and 4 GB RAM and 250 GB HDD. This system uses HDFS for storage, ith 256 MB block size. The machines have a 100Mbps per second Ethernet connection to a shared switch fabric.

The dataset downloaded we generate the web link graph for PageRank based on the statistics of a web graph of ClueWeb consist of 20,000,000 pages and 365,684,186 links and having overall size 36.4GB. For each run the cache is cleared to get cleared result. The data loading time over data nodes is 20 minutes and the running time of algorithm for plane PageRank algorithm over given platform is given as shown in given table.

<table>
<thead>
<tr>
<th>Sr.no.</th>
<th>Algorithm</th>
<th>Time taken on Plane Hadoop</th>
<th>Incremental Map Reduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PageRank</td>
<td>20 minutes</td>
<td>20 minutes</td>
</tr>
</tbody>
</table>

![Time taken in minutes for Normal MapReduce](image1)

Fig. 1: Time taken in minutes for normal mapreduce

After 10% changes in given dataset the time taken by both frameworks is given in table given below,

<table>
<thead>
<tr>
<th>Sr.no.</th>
<th>Algorithm</th>
<th>Time taken on Plane Hadoop</th>
<th>Incremental MapReduce with state storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PageRank</td>
<td>20 minutes</td>
<td>4 minutes</td>
</tr>
</tbody>
</table>

For iterative PageRank algorithm, the delta input is generated by randomly changing 10 percent of the input data. To make the comparison as fair as possible, we start incremental iterative processing from the previously converged states for all the four solutions. To get exact knowledge running time the experiments were run for 2, 5, 10 and 20% changes and the results obtained for incremental MapReduce are shown below,

![Time taken in minutes for inc.MapReduce](image2)

Fig. 2: Time taken in minutes for inc.Mapreduce
VIII. CONCLUSION AND SCOPE

Incremental data processing model which is compatible with the MapReduce. The framework combine the incremental and iterative engine to enhance the performance of simple MapReduce. The preserved states really producing the promising result and significantly reduce the run time for refreshing big data mining results compared to re-calculating on both simple and multi stage MapReduce.

The scope of system is wide since it covers all the major and minor considerations and problems that are comes during the simple Mapreduce processing. As big data increasing rapidly according to time here is a demand of system that works robustly in the sense of large and huge data. The proposed framework is promising to generate fine and expected result within time bound by avoiding recomputation. This concept provide the absolute extension to the simple Mapreduce and put one step forward to tackle the problem of refreshing the result, so the outcome become up-to-date.

ACKNOWLEDGMENT

For everything we achieved, the credit goes to all those who had really helped us to complete this work successfully. We are extremely thankful to P. G. Coordinator "Prof. S. A. Kahate” for guidance and review of this paper. I would also like to thanks the all faculty members of "SharadchandraPawar College of Engineering”

REFERENCES