ISSN: 0974-2115
www.jchps.com Journal of Chemical and Pharmaceutical Sciences

Dynamic Processing for large data sets using Bipartite Graph

Praveen T*, Priyatharsini B, Preethi MN, Sujatha R
Department of Computer Science & Engineering, Vel Tech High Tech Dr.Rangarajan Dr.Sakunthala Engineering College, Chennai, India
***Corresponding author:E-Mail:praveent738@gmail.com**

ABSTRACT

As huge amount of data is being accreted with persistent arrival of new updates in various fields like social networks, e-commerce, finance and education, the retrieval of data becomes complex requiring efficient algorithms to manage the tasks. Map reduce which is most widely used framework involves task-level computation in analysis of big data. This involves re-computation which results in long time processing and less performance. Hence, this framework seems to be less efficient. We employ Bipartite graph to perform Map reduce for efficient and fast processing. Also we focus on reducing time complexity to make the system competent. Inorder to show significant performance improvements, the large data sets are processed using Bipartite graph.

KEYWORDS: Bigdata, Hadoop, MapReduce, HDFS, Bipartite Graph

1. INTRODUCTION

Recent updates in technology made organizations to collect and store extremely large amounts of data resulting in complex data mining. “Big Data” management and processing has been one of the biggest challenges of our era. Big Data refers to datasets grow so large and complex that it is difficult to capture, store, manage, share, analyze and visualize within current computational architecture Big Data cannot be processed using traditional computing techniques. For example, social networks require large volume of data to be collected and manage on a daily basis, can fall under the category of Big data. Big Data is not only about scale and volume, it also involves one or more of the following aspects − Velocity, Variety, Volume, and veracity. We have large number of computing frameworks that have been developed for big data analysis. One among those frameworks is Map Reduce (with its open-source implementation) is the most widely used in production. Hadoop supports running applications on large clusters of commodity hardware and provide fast and reliable analysis of both structured and unstructured data. It uses simple programming model and can scale from single servers to thousands of machines, each offering local computation and storage. Map reduce supports task level computation which takes long time to process and makes the whole system less efficient. We propose Bipartite graph which supports state level processing that is, it saves as states and in case of data change the affected chunk alone is recomputed by saving processing time thus makes the system more efficient.

Map reduce background: MapReduce is the programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. MapReduce operates in parallel across massive cluster sizes. Because cluster size doesn't affect a processing job's final results, jobs can be split across almost any number of servers. MapReduce serves two essential functions: It parcels out work to various nodes within the cluster or map, and it organizes and reduces the results from each node into a cohesive answer for a query by user.

MapReduce also consists of several components, including:

- JobTracker -- the master node that manages all jobs and resources in a cluster
- TaskTrackers -- agents deployed to each machine in the cluster to run the map and reduce tasks
- JobHistoryServer -- a component that tracks completed jobs, and is typically deployed as a separate function or with JobTracker

The basic architecture of map reduce is as shown in fig.1.1. The term MapReduce actually refers to two distinct jobs that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. Both these functions can be prototyped as follows:

$Map(K_1,V_1)\rightarrow[(K_2,V_2)]$.

$Reduce(K_2,\{V_2\})\rightarrow[(K_3,V_3)]$.

When a file is divided into equal sized blocks (64MB-128MB) and each block is assigned to a cluster, the job tracker starts a map task for each data block and typically assigns it to the task tracker on the machine. Each data block will have a mapper. Each mapper will have a corresponding reducer. The mapper performs map function and we get the intermediate results. This intermediate results enters the reduce phase.The reducer performs reduce function. The results of all task tracker are combined to get the output.

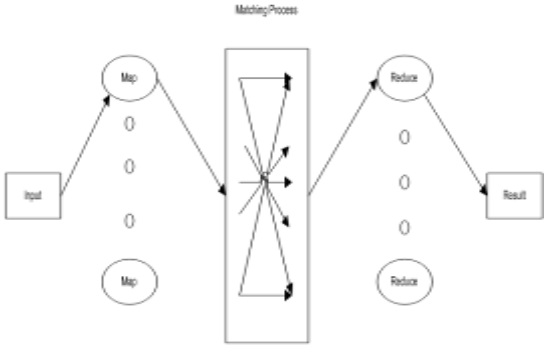

Figure.1.MatchingProcess

Though Map Reduce framework seems to be more efficient for processing multi-petabytes of data, it also possess some limitations. Some of the limitations are: Map reduce seems to be useless when it comes to the case of computation depending on previously computed values. Thus hadoop map reduce is not iterative. It follows Task level processing. This kind of processing is not suitable for dynamic data.

Proposed work: We begin with the idea of integrating bipartite graph on hadoop. The map reduce task which processes the entire task considering it as new data even when the input data has some modifications. This increases overhead in the entire framework. We aim at designing the system that works on the input data considering as states rather than task, and it is saved as states for future reference. This is termed as state level processing. Unlike basic map reduce, this design supports iterative processing with the aim of reducing re-computations as much as possible. In map reduce, Even on small number of updates may propagate to affect a large portion processing. To address this problem, we propose to reuse the preserved state from the previous computation. We present main idea to perform iterative processing by preserving the states in bipartite graph store. We also propose the method to identify the chunk on which data change has taken place.

System architecture:

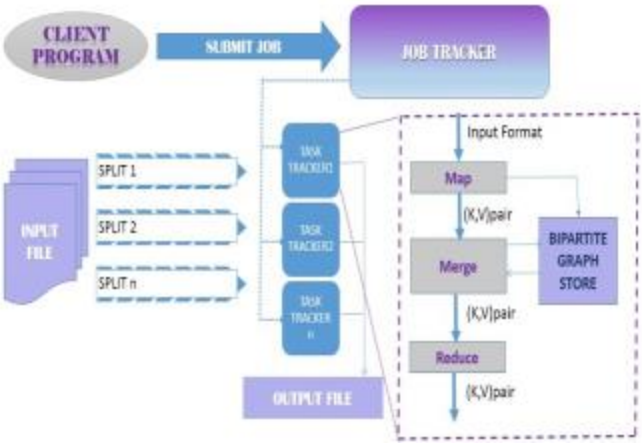

Figure.1.System Architecture

In this section, we present the working architecture of processing of map reduce framework with the implementation of bipartite graph. The client program are the queries raised by the client in order to get the refined results that requires processing the huge volume of data.

The related volume of data serves as input data which is voluminous that it cannot process on single node. Thus the input data is split into various blocks each called as input splits. Each input split is given to a task tracker which performs the actual map reduce job and is saved as preserved Bipartite graph. For dynamic data, the changed data block (or) chunk which is also called as delta data alone undergoes the map reduce process and is saved as delta Bipartite graph. By comparing both these preserved and delta bipartite graph, we get an updated Bipartite graph and is stored in the output file.

Bipartite graph store: Bipartite Graph Store is the central idea of the system. The Bipartite Graph is a graph whose vertex-set can be split into two sets in such a way that each edge of the graph joins a vertex in first set to a vertex in second set that is, every edge connects a vertex in one set to one in other set. The input is in the form of (Key, Value) pair and it can be represented as (K1, V1),(K2,V2),(K3,V3) etc. Each vertex in the Map task represents an individual Map function call instance on a pair of (K1, V 1). Each vertex in the Reduce task represents an individual Reduce function call instance on a group of (K2,{V2}). An edge from a Map instance to a Reduce instance means that the Map instance generates a (K2, V2) that is shuffled to become part of the input to the Reduce instance.

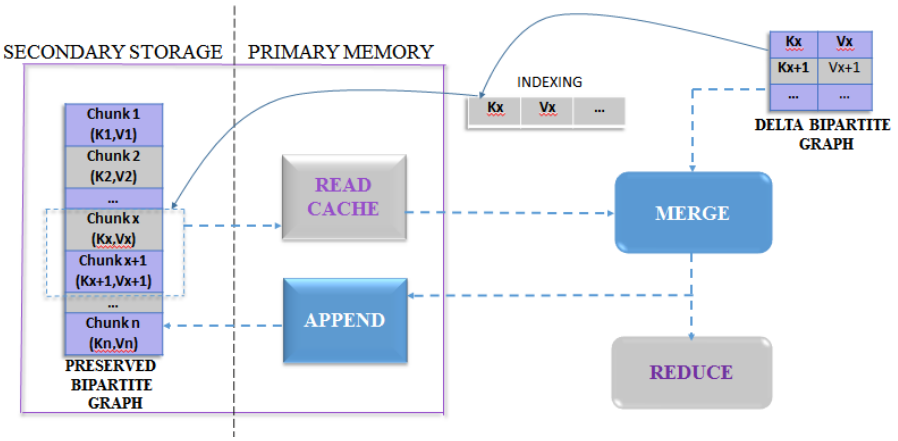

Figure.2.Bipartite Graph Store

The Figure 2 shows the detailed processing of our system in terms of data changes that is, modified data. The modified data is identified which is in the form of (Key, Value) pair. This delta data (changes made in input) undergoes map phase,where the mapper splits the data in equal sized blocks e.g.: if the block size is 128 MB it splits the data into equal sized 64MB.In each block the data remains the same and this can be used in case of data loss during transmission or reception etc. After the mapper, using indexing algorithm like SHA1 we retrieve the data of affective chunk with the help of Key and Value pair which is stored in Delta Bipartite Graph store. Then the delta Bipartite Graph and the preserved Bipartite Graph is combined or merged (merge phase).One copy of the updated result enters the reduce phase and the other copy is appended to save the changes that occured to the preserved Bipartite Graph.Thus the reduce phase eliminates the duplicacy in the data and produces the final output.

Directed Acyclic graph representation:

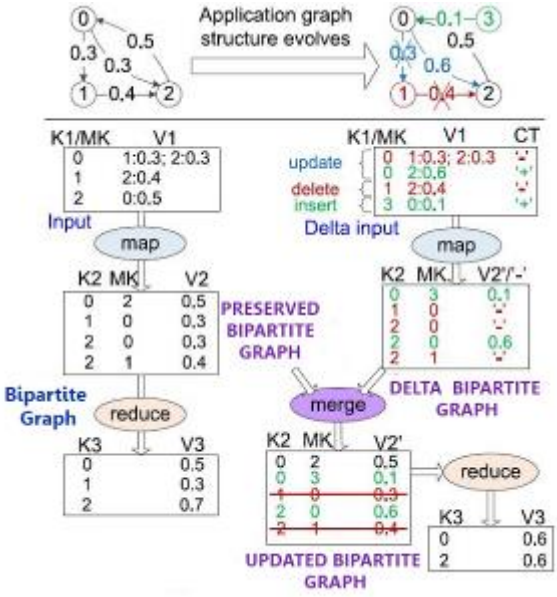

Figure.3.Dynamic processing for an application that computes the sum of in-edge weights for each vertex in graph

A directed acyclic graph, is a directed graph with no directed cycles. That is, it is formed by a collection of vertices and directed edges, each vertex is connected to every other vertex by edges.

This graphical representation not only follows one-one, one-many, many-one but also many-many mapping. DAGs may be used to model many different kinds of information. A partial order can be formed by using reachability in a directed acyclic graph. A collection of tasks that must be ordered into a sequence, subject to constraints that certain tasks must be performed earlier than others, may be represented as a DAG with a vertex for each task and an edge for each constraint. DAGs may be used as a space-efficient representation of a collection of sequences with overlapping sub-sequences. DAGs have been used to represent systems of events or potential events and the causal relationships between them. The fig.3 illustrates the dynamic processing for dynamic data. With the initial run performing normal map reduce our system also sees for delta data i.e, newly inserted, deleted or updated data. For this delta data it performs map reduce with modified kv-pairs. From initial run we will get a preserved Bipartite graph and wit the next run we get a Delta Bipartite graph. The Engine merges both preserved Bipartite graph and delta Bipartite graph to get an updated Bipartite graph.

2. CONCLUSION

Thus, we integrate Bipartite graph on hadoop for dynamic and iterative processing. We employ Bipartite graph to perform map reduce thereby creating an effective processing tool for Big data. This helps in efficient and faster processing. Hence runtime performance is achieved. The system is made much more competent by reducing time complexity. Performance improvements are shown by processing the large data sets using Bipartite graph.

Future Enhancements: In future, we are trying to identify the data changes whenever there is a dynamic updation using FP algorithm. Though retrieval of data becomes easier with map reduce, the interdependency of map and reduce tasks requires more fault tolerance. So, we are focusing on good fault tolerance solution by analysing and experimenting with various computing frameworks like pagerank, k-meansGIM-V etc.

REFERENCE

Dean J and Ghemawat S, Mapreduce: Simplified data processing on large clusters, in Proc. 6th Conf. Symp. Opear. Syst.Des.Implementation, 2004, 10.

Ewen S, Tzoumas K, Kaufmann M and Markl V, Spinning fast iterative data flows, in Proc. VLDB Endowment, 5(11), 2012, 1268–1279.

Logothetis D, Olston C, Reed B, Webb K.C and Yocum K, Stateful bulk processing for incremental analytics, in Proc. 1st ACM Symp. Cloud Comput, 2010, 51–62.

Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A and Hellerstein JM, Distributed graphlab, A framework for machine learning and data mining in the cloud, in Proc. VLDB Endowment, 5(8), 2012, 716–727.

Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N and Czajkowski G, Pregel, A system for large-scale graph processing, in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, 135–146.

Mihaylov S.R, Ives Z.G and Guha S, Rex: Recursive, delta based data-centric computation, in Proc. VLDB Endowment, 5(11), 2012, 1280–1291.

Peng D and Dabek F, Large-scale incremental processing usingdistributed transactions and notifications, in Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation, 2010, 1–15.

Power R and Li J, Piccolo, Building fast, distributed programs with partitioned tables, in Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation, 2010, 1–14.

Yanfeng Zhang, Shimin Chen, Qiang Wang, and Ge Yu, i2MapReduce: Incremental MapReduce for Mining Evolving Big Data, in IEEE transactions on knowledge and data engineering, 27(7), 2015.

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin M.J, Shenker S and Stoica I, Resilient distributed datasets, A fault-tolerant abstraction for, in-memory cluster computing, in Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation, 2012, 2.

Zhang Y, Gao Q, Gao L and Wang C, Accelerate large-scale iterative computation through asynchronous accumulative updates, in Proc. 3rd Workshop Sci. Cloud Comput. Date, 2012, 13–22.

Zhang Y, Gao Q, Gao L and Wang C, Imapreduce: A distributedcomputing framework for iterative computation, J. Grid Comput, vol. 10, no. 1, 2012, 47–68,