" EXMR: An Enhanced way for Incremental Data Processing based on Mapreduce "

Mr. Kurhade Nilesh V. [1] Prof. Kahate S. A.[2]

[1] *Lecturer, Computer Dept, SPIET,Otur, Pune(India)*

[2] *Asst. Prof., Computer Dept, SPCOE,Otur, Pune(India)*

ABSTRACT

Map Reduce 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 in this paper 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 flow and data dependence in a MapReduce computation as a bipartitegraph, 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:- *Mapreduce, MRBGraph, Incremental Processing*

INTRODUCTION:-

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. These platforms lack built-in support for iterative programs, which arise naturally in many applications including data mining, web ranking, graph analysis, model fitting, and so on MapReduce [4]. Is a well-known framework for programming. commodity computer clusters to perform large-scale data processing in a single pass. A MapReduce cluster can scale to thousands of nodes in a fault-tolerant manner. Although parallel database systems may also serve these data analysis applications[2]. MapReduce programming model has simplified the implementation of many data parallel applications. The simplicity of the programming model and the quality of services provided by many implementations of MapReduce attract a lot of enthusiasm among distributed computing communities[3].

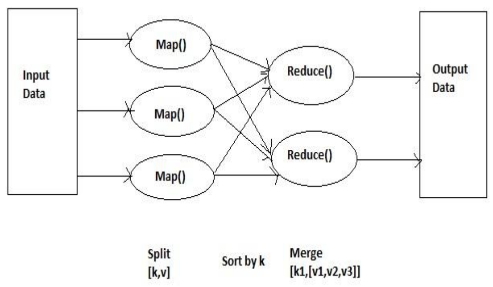


Figure:- Map Reduce Computation

MapReduce Background Procedure:-

Map (K1,V1) ->[(K2,V2)] // Map Function

Reduce (K2,{V2}) ->[<K3,V3>] // Reduce Function

Cluster computing frameworks like MapReduce and Dryad have been widely adopted for large-scale data analytics. These systems let users write parallel computations using a set of high-level operators, without having to worry about work distribution and fault tolerance. Although current frameworks provide numerous abstractions for accessing a cluster's computational resources, they lack abstractions for leveraging distributed memory. This makes them inefficient for an important class of emerging applications: those that reuse intermediate results across multiple computations. Data reuse is common in many iterative machine learning and graph algorithms, including Page Rank, K-means clustering, and logistic regression. Another compelling use case is interactive data mining, where a user runs multiple adhoc queries on the same subset of the data[2]. It is difficult to parallelize in-memory computation across many machines. As the entire computation is divided among multiple threads running on different machines, one needs to coordinate these threads and share intermediate results among them[3].

These cloud platforms share many implementation techniques with parallel DBMSs, yet they make different trade-offs that yield several benefits: scale-up to many (possibly heterogeneous) nodes, easier integration with user-defined code to support specialized algorithms, and transparent handling of failures (which frequently occur in large clusters)[5].

Parallel dataflow systems are an increasingly popular solution for analyzing large data volumes. They offer a simple programming abstraction based on directed acyclic graphs, and relieve the programmer from dealing with the complicated tasks of scheduling computation, transferring intermediate results, and dealing with failures. Most importantly, they allow dataflow programs to be distributed across large numbers of machines, which is imperative when dealing with today's data volumes. Besides parallel databases, MapReduce is the best known representative, popular for its applicability beyond relational data[4].

SYSTEM ARCHITECTURE:-

EXISTING SYSTEM:-

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce.

1) Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

2)Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial.

MAPREDUCE PROGRAMMING MODEL:-

MapReduce is a distributed programming technique proposed by Google for large-scale data processing in distributed computing environments. According to Jeffrey Dean and Sanjay Ghemawat, the input for MapReduce computation is a list of (key,value) pairs and each map function produces zero or more intermediate (key,value) pairs by consuming one input (key,value) pair. The runtime groups the intermediate (key,value) pairs based on some mechanism like hashing into buckets representing reduce tasks. The reduce tasks take an intermediate key and a list of values as input and produce zero or more output results [1].

PROPOSED SYSTEM:-

The propose i2MapReduce, an extension to MapReduce that supports fine-grain incremental processing for both onestep and iterative computation. Compared to previous solutions, i2MapReduce incorporates the following three novel.

features:

1) Fine-grain incremental processing using MRBG-Store. Unlike Incoop, i2MapReduce supports kv-pair level fine-grain incremental processing in order to minimize the amount of recomputationasmuch as possible. The model the kv-pair level data flow and data dependence in a MapReduce computation as a bipartite graph, called MRBGraph.

2) General-purpose iterative computation with modest extension to MapReduce API. Our previous work proposed iMapReduce to efficiently support iterative computation on the MapReduce platform. However, it targets types of iterative computation where there is a one-to-one/all-to-one correspondence from Reduce output to Map input. In comparison, our current proposal provides general-purpose support, including not only one-to-one, but also one-to-many,many-to-one, and many-to-many correspondence. Enhance the Map API to allow users to easily express loop-invariant structure data, and propose a Project API function to express the correspondence from Reduce to Map. While users need to slightly modify their algorithms in order to take full advantage of i2MapReduce.

3) Incremental processing for iterative computation. Incremental iterative processing is substantially more challenging than incremental one-step processing because even a small number of updates may propagate to affect a large portion of intermediate states after a number of iterations. To address this problem, propose to reuse the converged state from the previous computation and employ a change propagation control (CPC) mechanism. Also enhance the MRBG-Store to better support the access patterns in incremental iterative processing.

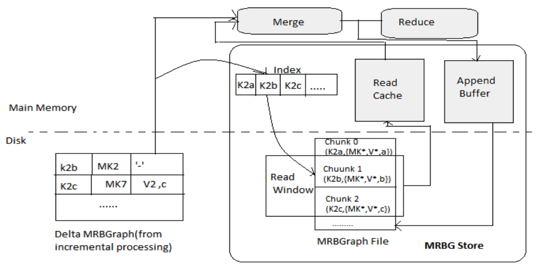


Fig.Structure of MRBG-Store

Figure: MRBG Store

MRBG ARCHITECTURE:-

1)Bipertite Graph:-a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are each independent sets) such that every edge connects a vertex in U to one in V. Vertex set U and V are often denoted as partite sets. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles.

The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors, as is required in the graph coloring problem. One often writes $G = (U, V, E)$ to denote a bipartite graph whose partition has the parts U and V, with E denoting the edges of the graph. If a bipartite graph is not connected, it may have more than one bipartition; in this case, the (U, V, E) notation is helpful in specifying one particular bipartition that may be of importance in an application. If $|U| = |V|$, that is, if the two subsets have equalcardinality, then G is called a balanced bipartite graph. If all vertices on the same side of the bipartition have the same degree, then G is called biregular.

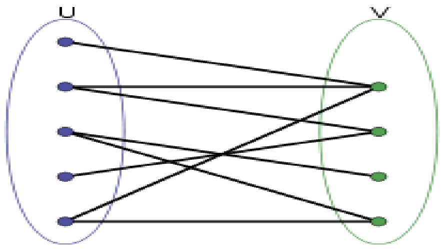
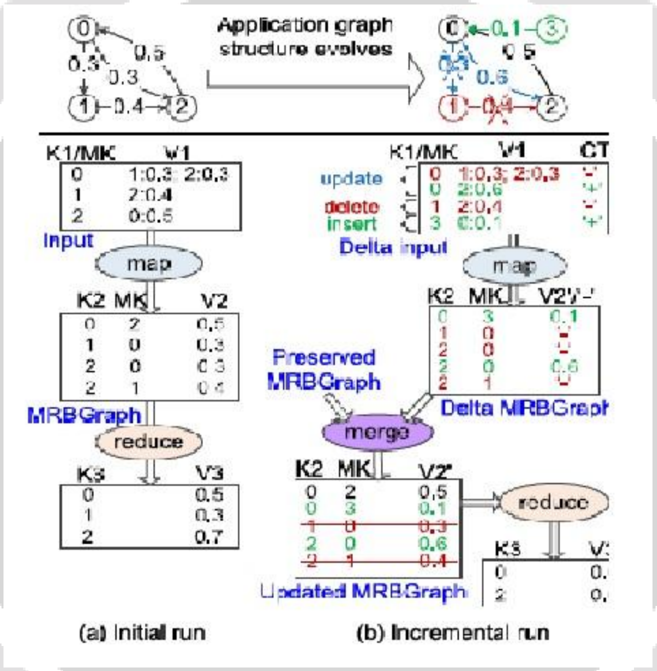


Figure: Bipertite Graph

MRBG DATAFLOW:-



K1/MK	V1
0	1:0.3; 2:0.3
1	2:0.4
2	0:0.5

K2	MK	V2
0	2	0.5
1	0	0.3
2	0	0.3
2	1	0.4

K3	V3
0	0.5
1	0.3
2	0.7

K1/MK	V1	CT
0	1:0.3; 2:0.3	'-'
0	2:0.6	'+'
1	2:0.4	'-'
3	0:0.1	'+'

K2	MK	V2'
0	3	0.1
1	0	-
2	0	-
2	0	0.6
2	1	-

K2	MK	V2*
0	2	0.5
0	3	0.1
~~1~~	~~0~~	~~0.3~~
2	0	0.6
~~2~~	~~1~~	~~0.4~~

K3	V3
0	0.
2	0.

Figure: Data Flow of MRBGraph.

MRBGraph edges are the fine-grain states M that would like to preserve for incremental processing. The edge can be contains the three pieces of the information:

(i) the source Map instance,
(ii) the destination Reduce instance,
(iii) the edge value.

Therefore, i2MapReduce will preserve (K2, MK, V 2) for each MRBGraph edge.

Timely dataflow is a computational model based on a directed graph in which stateful vertices send and receive logically time stamped messages along directed edges. The dataflow graph may contain nested cycles, and the timestamps reflect this structure in order to distinguish data that arise in different input epochs and loop iterations[14]. MapReduce is a well-known framework for programming commodity computer clusters to perform large-scale data processing in a single pass. A MapReduce cluster can scale to thousands of nodes in a faulttolerant manner. Although parallel database systems may also serve these data analysis applications, they can be expensive, difficult to administer, and lack fault-tolerance for longrunning queries. Hadoop, an open-source MapReduce implementation, has been adopted by Yahoo!, Facebook, and other companies for large-scale data analysis. With the

MapReduce framework, programmers can parallelize their applications simply by implementing a map function and a reduce function to transform and aggregate their data, respectively. Many algorithms naturally fit into the MapReduce model, such as word counting, equijoin queries, and inverted list construction[4].

ALGORITHMS:-

Algorithm 1.

Query Algorithm in MRBG-Store

```
Input queried key: k;
the list of queried keys: L
Output chunk k
1: if !readcache.contains(k) then
2: gap <-0, w<- 0
3: i k's index inL //That is, Li = k
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
11: return read cache.get_chunk(k)
```

Algorithm 2.

PageRank in MapReduce

```
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=∑i Ri,j+(1-d)
```

```
7: output < j, Nj|Rj>
```

```
Algorithm 3.Kmeans in MapReduce
Map Phase input: <pid, pval|{cid; cval}>
1: cid find the nearest centroid of pval in {cid; cval}
2: output <cid, pval>
Reduce Phase input: <cid, fpvalg>
3: cval compute the average of fpvalg
4: output <cid, cval>
```

FUTURE SCOPE:-

Though the proposed framework are able to improve the performance of Mapreduce hence it's System applications, still there is lots of scope to work, like its performance suffers when input and intermediate data cannot fit into memory, so In-memory Processing would be implantable along with this Bulk Synchronous Processing are the interesting future work to exploit the similar ideas in this work to proceed.

CONCLUSION:-

In this paper, a general approach to iterations in dataflows. As it leads to a huge speedup. To exploit computational dependencies in dataflow systems, suggested an abstraction for incremental iterations. an incremental data processing model which is compatible with the MapReduce model and its runtime. It supports MapReduce-based applications without any modification. Future research will be focused on how to optimize the state management because the size and location of the state data have significant impact on the efficiency and network bandwidth. Our future research includes state acquisition, preservation, transmission, and updating. Another point is how to utilize the latest state to optimize the sort and merge process in the reduce phase.

BIBLOGRAPHY:

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in Proc. 6th Conf. Symp. Opear. Syst. Des. Implementation, 2004, p. 10.
[2] 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," in Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation, 2012, p. 2.
[3] R. Power and J. Li, "Piccolo: Building fast, distributed programs with partitioned tables," in Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation, 2010, pp. 1–14.
[4] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 135–146.
[5] S. R. Mihaylov, Z. G. Ives, and S. Guha, "Rex: Recursive, deltabased data-centric computation," in Proc. VLDB Endowment, 2012, vol. 5, no. 11, pp. 1280–1291.