IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 3, Ver. IV (May-Jun. 2016), PP 125-129
www.iosrjournals.org

I++Mapreduce: Incremental Mapreduce for Mining the Big Data

Prof. Kanchan Dhomse B.[1]
[1](IT, METS BKC IOE, NASHIK, India)

Abstract: *Data mining is an interdisciplinary area of computer engineering. Incremental processing is a challenging approach to refreshing mining results and it uses saved states to avoid the cost of re computation from damage. Here, I propose i[2]MapReduce is a novel incremental processing expansion to MapReduce in data mining. As compared with the state-of-the-art work on Incoop, instead of using task level re-computation the i[2]MapReduce executes the key-value pair level incremental processing . It helps not only one-step computation but also more sophisticated iterative computation. Then it incorporates a set of novel techniques to decrease input output level for accessing dedicated fine-grain computation states in the data mining. The final results on Amazon EC2 describe the significant performance development of i[2]MapReduce as compared to iterative and plain MapReduce performing re computation.*

Keywords: *Big data, Incremental Processing, Mapreduce, MRBGraph, Recompilation.*

I. Introduction

Recently, huge amount of digital data is being assembled in many important areas like social network, e-commerce, education, finance, health care and environment. It has become increasingly popular to mine such big data in order to obtain outputs to help business decisions or to give better personalized and higher quality services in data mining. Recently a huge number of computing frameworks [1-10] have been developed for big data analysis in data mining. So, among these frameworks the MapReduce [1] with its open-source implementations, such as Hadoop is the mostly used in production due to its simplicity and generality and maturity. Here, I focus on improving MapReduce performance. Actually, Big data is constantly increasing new updates are being collected and the input data of a big data mining algorithm will slowely change. Generally, it is desirable to periodically updates the mining computation in order to keep the mining results accurate. Here, some are examples like the PageRank algorithm [11] evaluates ranking scores of web pages based on the web graph structure for supporting web search in data mining. Here, I explain the nature of the problem and previous work, purpose, and the contribution of the paper which shows the structure of mapreduce. The incremental processing utilize the fact that the input data of two subsequent computations are similar . Only a very small fraction of the input data has changed. Here, system observe and find the realization of this principle in the context of the MapReduce computing framework in data mining.

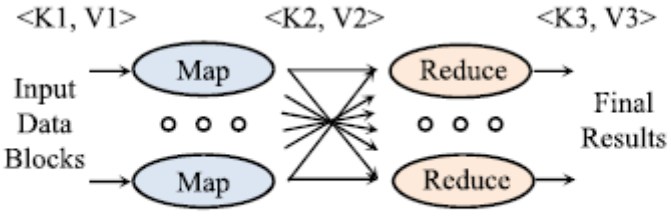

Fig.1: computation of Map Reduce

II. Implementation

1. Mapreduce Background

Generally, a MapReduce program is made up of a Map function and a Reduce function [1], as shown in Fig. 1. Their APIs are as shown below:

Map(k1,v1)-> (<k2,v2>)

Reduce(k2,v2)->(<k3,v3>)

Here, the Map function enters a kv-pair <K1, V1> as input and evaluate zero or more middle kv-pairs <K2,V2> in mapreduce function. Now all <K2, V2> are grouped by K2 and The Reduce function inserts a K2 and a list of V 2 as input and computes the final output kv-pairs <K3,V3> in mapreduce. So, a MapReduce

normally reads the input data of the MapReduce evaluation from and writes the final results to a distributed file system which distributes a file into equal-sized i.e. 64 MB blocks and collects the blocks across a cluster of machines in data mining. The mapreduce system executes a JobTracker process on a master node to concentrate the job progress and a set of Task Tracker processes on worker nodes to implement the actual Map and Reduce tasks. Then, the Job Tracker starts a Map task per data block and typically assigns it to the TaskTracker on the machine that holds the corresponding data block in order to decrease communication. At each Map task calls the map function for every input <K1, V1> pair and stores the middle kv-pairs <K2, V2> on local disks in mapreduce function. The intermediate results are shuffled to reduce tasks according to a divides the function on K2 in pair. So, after a reduce task gets and merges middle pair's results from all map tasks then it invokes the reduce function on every <K2, V 2> pair to create the final output kv-pairs <K3, V3>among all pairs.

2 Fine- Grain Incremental Processing For One-Step Computation

Here, we start by describing the basic idea of fine-grain incremental processing in the mapreduce function . it shows the main design including the MRBGraph abstraction and the incremental processing engine in mapreduce process. Here we divide it into two aspects of the design. i.e. the mechanism that maintain the fine-grain states and the handling of a special case where the Reduce function performs collection function.

2.1 MRBGraph Abstraction

We use a MRBGraph i.e. Map Reduce Bipartite Graph is the abstraction to model the data flow in MapReduce function. As shown in Fig. 2(a) every vertex in the Map task represents an single Map function call instance on a pair of <K1,V1> nodes. At every vertex in the Reduce task represents an individual Reduce function call instance on a group of <K2; V 2>. The range from a Map instance to a Reduce instance means that the Map instance creates a <K2,V2> pair that is shuffled to become part of the input to the Reduce instance of pair. Here, for example the input of reduce instance getting from map instance 0, 2, and 4 respectively. Therefore, i^{2}MapReduce will maintain (K2, MK, V 2) for each MRBGraph edge in given function.

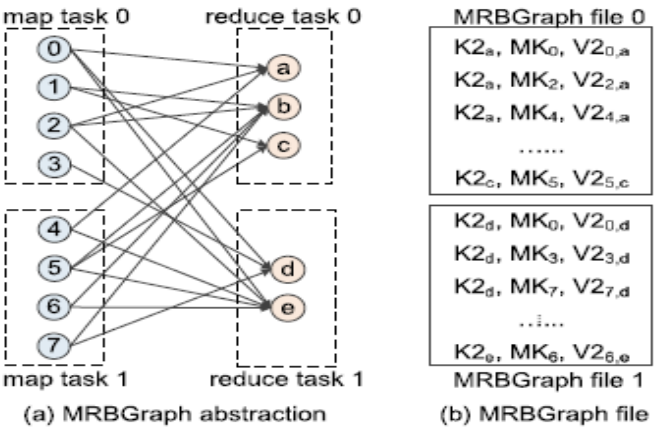

Fig. 2. MRB Graph

2.2 Fine-Grain Incremental Processing Engine

Here, figure shows the fine-grain incremental function engine with an example application which evaluates the addition of in-edge weights for each vertex in a graph of MRBG. So, as shown the input data i.e. the graph structure includes over time in running process. We present and explains that how the engine performs incremental processing to update the analysis results in mapreduce techniques.

III. Algorithms

In data mining and analysis the fundamental algorithms form field of data science which includes default methods to analyze patterns and models for all kinds of data with applications ranging from scientific discovery to business intelligence and analytics at the time of algorithm selection [8].

3 General-Purpose Support For Iterative Computation

3.1 Analyzing Iterative Computation :

PageRank is a iterative graph algorithm for ranking web pages in data mining. So, it evaluates a ranking score for each vertex in a graph in algorithm.

Algorithm 1: PageRank in MapReduce

Map Phase input: $< i, N_i | R_i >$

1: output $< i, N_i >$
2: **for all** j in N_i **do**
3: $R_{i,j} = \frac{R_i}{|N_i|}$
4: output $< j, R_{i,j} >$
5: **end for**

Reduce Phase input: $< j, \{R_{i,j}, N_j\} >$

6: $R_j = d \sum_i R_{i,j} + (1 - d)$
7: output $< j, N_j | R_j >$

Kmeans : The Kmean is a generally used clustering algorithm that partitions points into k clusters .

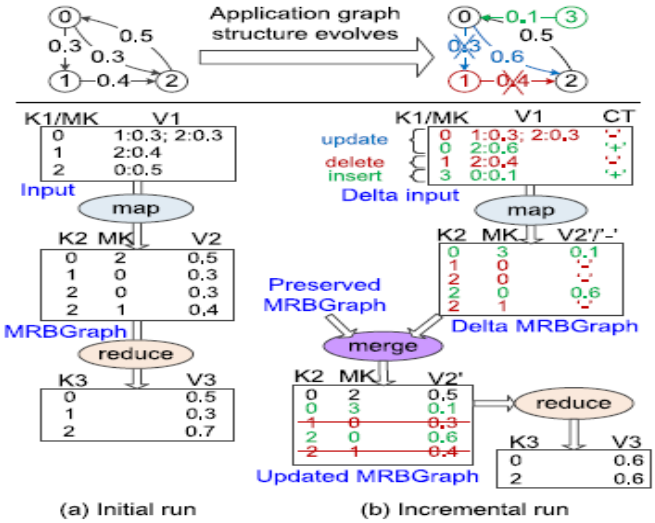

Fig. 3. Incremental processing for an application

Algorithm 2: kmeans in MapReduce

Map Phase input: $< pid, pval | \{cid, cval\} >$

1: $cid \leftarrow$ find the nearest centroid of $pval$ in $\{cid, cval\}$
2: output $< cid, pval >$

Reduce Phase input: $< cid, \{pval\} >$

3: $cval \leftarrow$ compute the average of $\{pval\}$
4: output $< cid, cval >$

4 Incremental Iterative Processing

Here, we shows the incremental processing techniques for iterative evaluation. But it is not sufficient to simply gather the above solutions for incremental one step processing and iterative computation given below we discuss 3 main aspects that we noticed in order to obtaine an effective design of i[2]mapreduce.

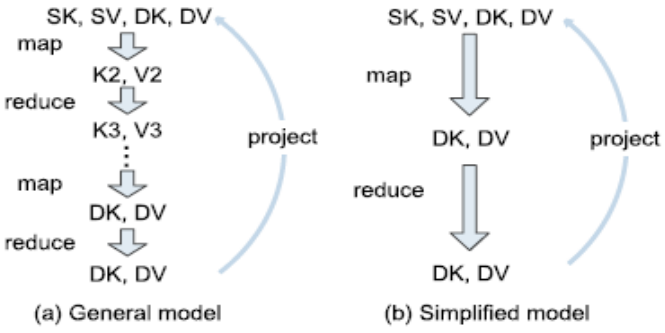

Fig. 6. Iterative model of i2MapReduce.

IV. Result Analysis

As we described all the algorithms which are helpful for i[2]mapreduce Solutions. Actually, our experiments compare four solutions initially PlainMR recomp re-computation on vanilla Hadoop then iterMR recomp and re-computation on Hadoop optimized for iterative evaluation. Secondly, hadoop re-computation on the iterative Mapreduce framework Hadoop which controls mapreduce by providing a structure data caching mechanism then in i[2]Mapreduce our proposed solution is to best of our knowledge the task-level coarse-grain incremental processing system. Here, Incoop is not normally present that's why we can't compare i[2]MapReduce with Incoop technique. So, the statistics describes that without careful data partition almost all tasks see changes in the experiments and making task-level incremental processing less important than the other.

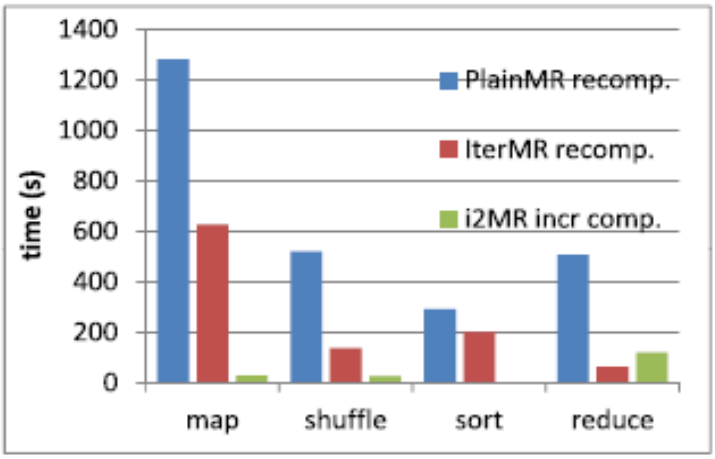

Fig. 7. Run time of individual stages in PageRank.

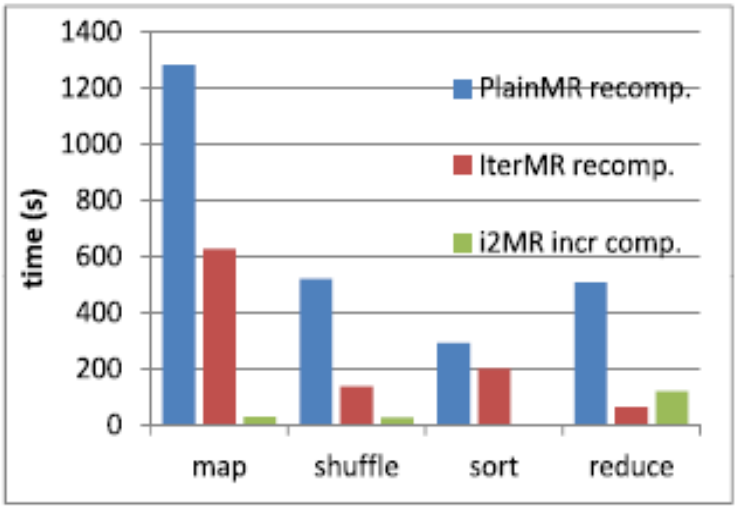

Fig. 8. Normalized runtime.

V. Conclusion

In data mining includes mapreduce function described i[2]MapReduce technique. A MapReduce based framework for incremental big data process is used for i[2]mapreduce function. I[2]MapReduce collects a fine-grain incremental engine and a general-purpose iterative model also a set of useful techniques for incremental iterative evaluation. The real-machine experiments describes that i[2]MapReduce can significantly decrease the run time for updating the big data mining results as compared to the recomputation on both plain and iterative MapReduce.

Acknowledgements

We especially thanks to our department and our Institute for the great support regarding paper and their all views. I really thankful to my all staffs and our H.O.D. Prof. N.R. kale. who showed me the way of successful journey of publishing paper.

References

Journal Papers

[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.

[6]. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine Learning and data mining in the cloud," in Proc. VLDB Endowment, 2012, vol. 5, no. 8, pp. 716–727.

Proceedings Papers

[7]. S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl, "Spinning fast iterative data flows," in Proc. VLDB Endowment, 2012, vol. 5,no. 11, pp. 1268–1279.

Books

[8]. Mohammed j. Zaki's and Wagner Meira jr.'s Data mining and Analysis (Cambridge university , 2014)