RESEARCHERID
THOMSON REUTERS

[ICESTM-2018]

ISSN 2348 – 8034
Impact Factor- 5.070

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

DATA PARTITIONING AND MINING STRATEGIES USING BY HADOOP CLUSTERS

V. Naveen Kumar[*1] & A. Ravi Kishore[2]
[*1&2]Asst. Professor, Dept. of CSE, BEC, Bapatla

ABSTRACT

Traditional parallel algorithms for mining frequent itemsets. This aims to balance load by equally partitioning data among a group of computing nodes. We study by the problem of the existing parallel Frequent Itemset Mining(FIM)algorithms. In this they given alargedataset, datapartitioning strategies in the existing solutions. Itsuffers high communication and mining transactions transmitted among computing nodes. We address this problem by developing a data partitioning approach called FiDoop-DP using the MapReduce programming model.It is the boost performance of parallel Frequent Item set Mining on Hadoop clusters. data partition to improve locality without creating an excessive number of redundant transactions. We implement FiDoop-DP on a 24-node Hadoop cluster. Experimental results that FiDoop-DP is to reducing network and computing loads by the virtue of eliminating redundant transactions of hadoop nodes.

Keywords: *Frequent itemset mining,parallel data mining, Data partitioning,Map Reduce programming model,Hadoop cluster.*

I. INTRODUCTION

Parallel Frequent Itemset Mining(FIM) techniques are focused on load balancing data.This data equally partitioned and distributed among computing nodes of a cluster. The absence of data collocation is increases the datashuffling costs and the network. It reduces the effectiveness of data partitioning. Thedata partitioning in FIM affects not only network traffic it also computing loads. It shows that data partitioning algorithms should pay attention to network and computing loads in addition to the issue of load balancing.We propose a parallel FIM approach called FiDoop-DP using the Map Reduce programming model.It implements the data partition anddistribute a large dataset across data nodes ofa Hadoop cluster to reduce network andcomputing loads induced by makingredundant transactions on remote nodes.FiDoop-DP is conducive to speeding up theperformance of parallel FIM on clusters.

II. RELATED WORK:

The following three observations motivate us to develop the performance of FIM on high-performance clusters.

- There is a pressing need for the development of parallel FIM techniques.
- The Map Reduce programming model is an ideal data centric mode to address the rapid growth of big-data mining.
- Data partitioning in Hadoop clusters play a critical role in optimizing the performance of applications processing large datasets.

Data Partitioning in Map Reduce: The existing approaches typically produce possible ranges or hash partitions, which are then evaluated using heuristics and cost models.

High scalability is one of the most important design goals for Map Reduce applications. the partitioning techniques in existing MapReduce platforms (e.g., Hadoop) are in their infancy, leading to serious performance problems.a data redistribution algorithm in HDFS to address the data-skew issue imposed by dynamic data insertions and deletions.
CoHadoop [33] is a Hadoop's lightweight extension, which is designed to identify related data files followed by a modified data placement policy to co-locate copies of those related files in the same server. CoHadoop considers the

relevance among files; that is, CoHadoop is an optimization of HaDoop for multiple files. A key assumption of the MapReduce programming model is that mappers are completely independent of one another.
Graph and hypergraph partitioning have been used to guide data partitioning in parallel computing. Graph-based partitioning schemes capture data relationships. For example, Ke et al. applied a graphic-execution-plan graph (EPG) to perform cost estimation and optimization by analyzing various properties of both data and computation [35]. Their estimation module coupled with the cost model estimate the runtime cost of each vertex in an EPG, which represents the overall runtime cost; a data partitioning plan is determined by a cost optimization module.

Application-Aware Data Partitioning:
Various efficient data partitioning strategies have been proposed to improve the performance of parallel computing systems. Traditional term-based partitioning has limited scalability due to the existence of very skewed frequency distributions among terms. Load-balanced distributed clustering across networks and local clustering are introduced to improve the chance that triples with a same key are collocated. These self-organizing approaches need no data analysis or upfront parameter adjustments in a priori. In Lu's study, objects are divided into partitions using a Voronoi diagram with carefully selected pivots. Then, data partitions (i.e., Voronoi cells) are clustered into groups only if distances between them are restricted by a specific bound. In this way, their approach can answer the k-nearest-neighbour join queries by simply checking object pairs within each group.

FIM for data-intensive applications over computing clusters has received a growing attention; efficient data partitioning strategies have been proposed to improve the performance of parallel FIM algorithms. A MapReducebased Apriori algorithm is designed to incorporate a new dynamic partitioning and distributing data method to improve mining performance [39]. This method divides input data into relatively small splits to provide flexibility for improved load-balance performance.the master node doesn't distribute all the data once.

III. LITERATURE SURVEY

In Hadoop clusters, the amount of transferred data during the shuffling phase heavily depends on localities and balance of intermediate results. when a data partitioningschemepartitionstheintermediateresults,data locality and balance are completely ignored. In the existing Hadoop-based FIM applications [7][8][11], the traditional data partitioning schemes impose a major performance problem due to the following reasons: Conventional wisdoms in data partitioning aim to yield balanced partitions using either a hash function or a set of equally spaced range keys [12][13]. Interestingly, we discover that excessive computation and network loads are likely to be caused by inappropriate data partitions in parallel FIM. The traditional grouping strategy evenly groups the items into two groups by descending frequency .this grouping decision forces all the transactions to be transmitted to the two partitions prior to being processed. We argue that such a high transaction-transfer overhead can be reduced by making a good tradeoff between cross-node network traffic and load balancing.

IV. PROBLEM DEFINITION

Baseline Methods and Problems:
The most existing parallel FP-Growth algorithms basically followed the workflow plotted , where the second MapReduce job is the most performance critical and time-consuming among the four steps.

Experiment results reported in suggest that local FP-Growth cost accounts for more than 50% of the overall mining time and the grouping strategy plays the most importantroleinaffectingsubsequentdatapartitioningand local FP-Growth performance.

Design Goals:
FiDoop-DP aims to partition input transactions (1) to reduce the amount of data transferred through the network during the shuffle phase and (2) to minimize local mining load. Recall that high shuffling cost and local mining load are incurred by redundant transactions. In what follows, we formally state the design goal of Fidoop-DP.

ISSN 2348 – 8034
Impact Factor- 5.070

$$pi = \begin{cases} 1, & S(G_i) \neq T(G_i) \\ 0, & \text{otherwise.} \end{cases} \quad \_\_(1)$$

$$\text{Minimize:} \sum_{i=1}^{m} D_i \times p_i$$

V. PROPOSED APPROACH

This goals is achieved in FiDoop-Dpby reducing network and computing loads through the elimination of redundant transactions on multiple nodes. To alleviate the excessive network load problem.

The implementation details of LSH-based FiDoop-DP running on Hadoop clusters. which consists of four steps (i.e., one sequential-computing step and three parallel MapReduce jobs). Specifically, before launching the FiDoop-DP process, a preprocessing phase is performed in a master node to select a set of (k) pivots which serve as an input of the second MapReduce job that is responsible for the Voronoi diagrambased partitioning .In the first MapReduce job, each map per sequentially reads each transaction from its local input split on a data node to generate local 1-itemsets. Next, global 1-itemsets are produced by a specific reducer, which merges local 1itemsets sharing the same key (i.e., item name). The output of these reducers include the global frequent 1-itemsets along with their counts.
The second step sorts these global frequent 1-itemsets in a decreasing order of frequency; the sorted frequent 1-itemsets are saved in a cache named FList, which becomes an input of the second MapReduce job in FiDoop-DP. The second MapReduce job applies a second-round scanning on the database to repartition database to form a complete dataset for item groups in the map phase. Each reducer conducts local FP-Growth based on the partitions to generate all frequent patterns. The last MapReduce job aggregates the second MapReduce job's output (i.e., all the frequent patterns) to generate the final frequent patterns for each item.

VI. PROPOSED METHODOLOGY:

The data partitioning issues in parallel FIM.We focused on MapReduce-based parallel FP-tree algorithms; in particular, we studied how to partition and distribute a large dataset across data nodes of a Hadoop cluster to reduce network and computing loads.the communication and synchronization cost induce adverse impacts on the performance of parallel mining algorithms. The basic idea of Fidoop-DP - grouping highlyrelevanttransactionsintoapartition-allowstheparallel algorithms to exploit correlations among transactions in database to cut communication and synchronization overhead among Hadoop nodes.

VII. ALGORITHM

Algorithm 1: LSH-Fpgrowth
Input:*FList*, *k* pivots, DB*i*;
Output: transactions corresponding to eachGid;
1: function MAP(key offset, values DBi)
2: load FList, k pivots;

ISSN 2348 – 8034
Impact Factor- 5.070

```
3: Glists ← GenerateGlists(FList,kpivots);/* based on the correlation of each item in FList and k pivots */
4: for all (T in DBi) do
 5: items[] ← Split(eachT);
 6: for all (item in items[]) do
 7: if item is in FList then
 8: a[] ← item
 9: end if
 10: end for
 11: Add Generate-signature-matrix(a[]) into ArrarylistsigMatrix;
12: end for
13: for all (ci in sigMatrix ) do
14: divide ci into b bands with r rows;
 15: Hashbucket ← HashMap(each band of ci());
 16: end for
 17: if at least one band of ci and pivot pj is hashed into the same bucket then
18: Gid ← j;
 19: Output(Gid, new TransactionTree(a[i]));
20: end if
 21: for all each GListt(t 6= i) do
 22: if ci contains an item in GListt then
23: Gid ← t
24: Output(Gid, new TransactionTree(a[i]));
/* guarantee the data completeness for each GList */
 25: end if
26: end for
27: end function Input: transactions corresponding to each Gid; Output: frequent k-itemsets;
28: function REDUCE(key Gid, values DBGid)
29: Load GLists; 30: nowGroup ← GListGid
31: localFptree.clear;
 32: for all (Ti in DBGid) do
33: insert-build-fp-tree(localFptree, Ti);
 34: end for
35: for all (ai in nowGroup ) do
36: Define a max heap HP with size K;
 37: Call TopKFPGrowth(localFptree,ai,HP);
38: for all (vi in HP) do
39:Output(vi, support(vi));
40: end for
41: end for
```

Algorithm 2 : Generate-signature-matrix

```
Input: a[];
Output: signature matrix of a[];
1: function GENERATE-SIGNATURE-MATRIX(a[])
2: for (i=0;i<numHashFunctions;i++) do
3: minHashValues[i] =Integer.MAX V ALUE;
4: end for
5: for (i=0;i<numHashFunctions;i++) do
6: for all ele: a[] do
7: value ← Integer(ele);
 8: bytesToHash[0]=(byte)(value >> 24);
 9: bytesToHash[1]=(byte)(value >> 16);
```

10: bytesToHash[2]=(byte)(value >> 8);
11: bytesToHash[3]=(byte)value);
12: hashIndex← hashFunction[i].hash(bytesToHash);
13: if (minHashValues[i]) >hashIndex then
14: minHashValues[i]=hashIndex;
15: end if
16: end for
17: end for
18: end function

VIII. RESULTS

To mitigate high communication and reduce computing cost in MapReduce-based FIM algorithms, we developed FiDoop-DP, which exploits correlation among transactions to partition a large dataset across data nodes in a Hadoop cluster.

One of the features of FiDoop-DP lies in its capability of lowering network traffic and computing load through reducing the number of redundant transactions, which are transmitted among Hadoop nodes.

IX. CONCLUSION

Our experimental results reveal that FiDoop-DP significantly improves the FIM performance of the existing Pfp solution by up to 31% with an average of 18%. We introduced in this study a similarity metric to facilitate data-aware partitioning. As a future research direction, we will apply this metric to investigate advanced loadbalancing strategies on a heterogeneous Hadoop cluster.

X. ACKNOWLEDGMENT

The work in this paper was in part supported by the National Natural Science Foundation of P.R. China (No.61272263, No.61572343). Xiao Qin's work was supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER). The authors would also like to thank Mojen Lau for proof-reading

REFERENCES

[1] M. J. Zaki, "Parallel and distributed association mining: A survey," Concurrency, IEEE, vol. 7, no. 4, pp. 14–25, 1999. [2] I. Pramudiono and M. Kitsuregawa, "Fp-tax: Tree structure based generalized association rule mining," in Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. ACM, 2004, pp. 60–63. [3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[2] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "Pfp: parallel fp-growthforqueryrecommendation,"inProceedingsofthe2008ACM conference on Recommender systems. ACM, 2008, pp. 107–114.

[3] A. Stupar, S. Michel, and R. Schenkel, "Rankreduce–processing knearestneighbor queries on top of mapreduce," in Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval. Citeseer, 2010, pp. 13–18. [25] B. Bahmani, A. Goel, and R. Shinde, "Efficient distributed locality sensitive hashing," in Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, 2012, pp. 2174–2178. [26] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. ACM, 2006, pp. 1186–1195. [27] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Minwise independent permutations," Journal of Computer and System Sciences, vol. 60, no. 3, pp. 630–659, 2000. [28] L. Cristofor, "Artool," 2006.

ISSN 2348 – 8034
Impact Factor- 5.070

[4] *B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," Proceedings of the VLDB*

[5] *M. Y. Eltabakh, Y. Tian, F. ¨Ozcan, R. Gemulla, A. Krettek, and J. McPherson, "Cohadoop: flexible data placement and its exploitation in hadoop," Proceedings of the VLDB Endowment, vol. 4, no. 9, pp. 575–585, 2011. [34] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, "Adaptive mapreduce using situation-aware mappers," in Proceedings of the 15th International Conference on Extending Database Technology. ACM, 2012, pp. 420–431.*