INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 2 ISSUE 6 (2016) PAGES 1392 - 1396
RECEIVED : 16/11/2016. PUBLISHED: 28/11/2016
November 28, 2016

PARALLEL MINING OF FREQUENT ITEMSETS USING FIMN ON NEO4J

[1]M .Jayashree, [2]Dr. M. Barathi,
[1]PG Scholar, Dept Of CSE, Ganadipathy Tulsi's Jain Engineering College, Vellore,
[2]Prof& HOD, Dept Of CSE, Ganadipathy Tulsi's Jain Engineering College, Vellore.

Abstract

In parallel mining algorithms for frequent itemsets multiple mechanismare used (for eg.load balancing, data distribution, automatic parallelization, and fault tolerance on large clusters). For solution to this problem, we propose a new parallel frequent itemsets mining algorithm called FiDoop using the MapReduce programming model. FiDoop incorporates the frequent items ultrametric tree for achieving reduces storage and avoids building conditional pattern bases, rather than conventional FP trees. In FiDoop, we used three MapReduce Jobs are implemented to complete the mining task. In third MapReduce job, mappers decompose itemsets independently and reducer constructing small ultrametric trees, mining of these trees separately. In this paper, we implement FiDoop on our inhouse Hadoop cluster. We show that FiDoop on the cluster is sensitive to data distribution and dimensions, because itemsets with different lengths have different decomposition and construction costs. For improvingFiDoop's performance and workload balance metric to measure load balance across the cluster's computing nodes, in this paper we developFiDoop-HD. FiDoop-HD helps to speed up the mining performance for high-dimensional data analysis. Extensive experiments using real-world celestial spectral data demonstrate that our proposed solution is efficient and scalable. In our proposed scheme, we will add various approaches to improving energy efficiency of FiDoop running on Hadoop clusters.

Keywords: MapReduce, Energy efficiency, frequent itemsets, Frequent Items Ultrametric Tree (FIU-tree), Hadoop cluster, Load balance.

1. INTRODUCTION

An elementary necessity for mining for mining association rules is mining frequent itemsets. Numerous algorithms exist for frequent itemset mining. Apriori and FP-Growth are the traditional method. Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by recognizing the frequent individual items in the database and widening them to larger item sets providing those item sets appear adequately often in the database. It works with Candidate Generation and Test Approach.Fp-Growth is used to overcome the problem of candidate generation. FP-growth is a program to find frequent item sets with the FP-growth algorithm, which corresponds to the transaction database as a prefix tree which is enhanced with links that organize the nodes into lists referring to the same item. The search is carried out by prognostic the prefix tree, working recursively on the result, and trimming the original tree. The implementation also supports sifting for closed and maximal item sets with conditional item set repositories, although the approach used in the program differs in as far as it used top-down prefix trees rather than FP-trees. FP-growth condense a large database into a compact, Frequent-Pattern tree (FP-tree) structure with highly reduced, but complete for frequent pattern mining and avoid costly database scans. It develops an efficient, FP-tree-based frequent pattern mining method with a divide-and-conquer methodology which decomposes mining tasks into smaller ones and avoids candidate generation. The disadvantage

of this algorithm consists in the TID_set being too long, taking considerable memory space as well as computation time for intersecting the long sets. Incremental data mining is not hold by this algorithm.FREQUENT itemsets mining (FIM) is a center issue in association rule mining (ARM), grouping mining, and so forth. Accelerating the procedure of FIM is basic and essential, on the grounds that FIM utilization represents a huge part of mining time because of its high calculation andinput/output (I/O) force. At the point when datasets in present day information mining applications turn out to be too much substantial, consecutive FIM calculations running on a solitary machine experience the ill effects of execution decay. To address this issue, we research how to perform FIM utilizing MapReduce—a generally embraced programming model for preparing enormous datasets by misusing the parallelism among registering hubs of a bunch. We demonstrate to appropriate an extensive dataset over the group to adjust load over all bunch hubs, accordingly improving the execution of parallel FIM. Big information for the most part incorporates information set with sizes past the capacity of generally utilized programming devices to catch, oversee and handle information inside a fair passed time. Its size is continually moving focus starting 2012 going from a couple of Dozen of terabyte to numerous petabytes of information "greatly parallel programming running on tens, hundreds, or even a large number of servers".

2. LITERATURE SURVEY

Simplified Data Processingon Large Clusters and Execution Overview Large-Scale Indexing.MapReduce is a programming model and an associated implementation for processingand generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day. In this proposed system efficient generation for large itemsets by hash method (2) effective reduction on itemsets scan required by the division approach and (3) the option of reducing the number of database scans required Our proposed hash and division-based techniques.We propose efficient use of Hadoop on heterogeneous clusters as well as on virtual/cloud infrastructure, both of which violate the peer-similarity assumption. To this end, we have implemented and here present preliminary results of an approach for automatically diagnosing the health of nodes in the cluster, as well as the resource requirements of incoming MapReduce jobs. We show that the approach can be used to identify abnormally performing cluster nodes and to diagnose the kind of fault occurring on the node in terms of the system resource affected by the fault (e.g., CPU contention, disk I/O contention). We also describe our future plans for using this approach to increase the efficiency of Hadoop on heterogeneous and virtual clusters,with or without faults.

3. MAPREDUCE-BASED FIDOOP

In this section, we present the design issues of FiDoop built on the MapReduce framework. Depicts the working flow of FiDoop consisting of three MapReduce jobs. Recall that the intermediate results provided by the mappers in the third MapReduce job are used to construct FIU trees (see Algorithm 4). We intentionally keep such intermediate key-value pairs output to make the process flow concise.From the aforementioned description of the processes, we show that frequent one-itemsets are

INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 2 ISSUE 6 (2016) PAGES 1392 - 1396
RECEIVED : 16/11/2016. PUBLISHED: 28/11/2016
November 28, 2016

directly generated by scanning a database. FiDoop carries out a two-stage process to construct a k-FIU-tree ($2 \leq k \leq M$) from k-itemsets. In the first stage, k-itemsets are obtained by pruning infrequent items of each transaction in the second scan for database. The second stage is the combination of k-itemsets generated by decomposing all h-itemsets ($k < h$). Please note that the two-stage process is similar to that of the FIUT algorithm, which ensures the correctness of our algorithm. The following preliminary findings motivate us to address a pressing issue pertinent to balancing load in FiDoop: 1) large itemsets give rise to high-decomposition overhead and 2) and small decomposed itemsets lead to a large number of itemsets. To achieve good load balancing performance, we incorporate constraints in the shuffling phase of the MapReduce jobs in FiDoop, thereby balancing the number of itemsets across reducers (see Section V-A for details on load balancing).multiple input files stored by the HDFS across data nodes of a Hadoop cluster. Each mapper sequentially reads each transaction from its local input split, where each transaction is stored in the format of pair. Then, mappers compute the frequencies of items and generate local one-itemsets. Next, these one-itemsets with the same key emitted by different mappers are sorted and merged in a specific reducer, which further produces global oneitemsets. Finally, infrequent items are pruned by applying the minsupport; and consequently, global frequent one-itemsets are generated and written in the form of pair as the output from the first MapReduce job. Importantly, frequent one-itemsets along with their counts are stored in a local file named F-list, which becomes the input of the second MapReduce job in FiDoop.

4. IMPLEMENTATION

Now, we discuss the implementation details of FiDoop. We pay particular attention to the last MapReduce job in FiDoop, because the last job is computationally expensive. We show how to optimize the performance of the third MapReduce job in two approaches.The decompose() function of the third MapReduce job accomplishes the decomposition process. If the length of an itemset is m, the time complexity of decomposing the itemset is O(2m).

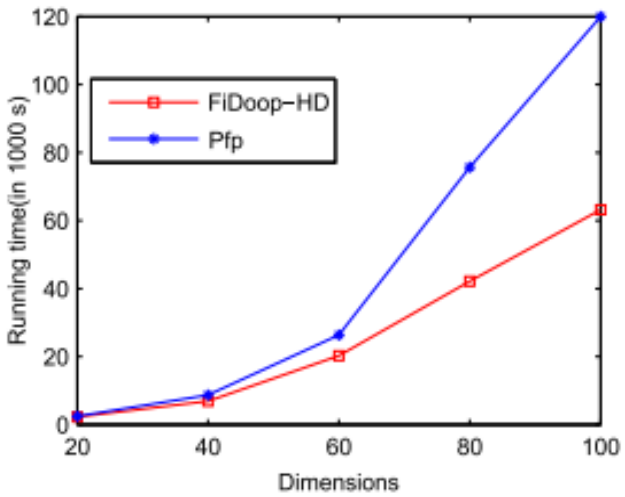

Fig.1.Effect of FiDoop-HD and Pfp

INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 2 ISSUE 6 (2016) PAGES 1392 - 1396
RECEIVED : 16/11/2016. PUBLISHED: 28/11/2016
November 28, 2016

```
Input: to be decomposed string s;
Output: the decomposed results l;
 1: function DECOMPOSE(s, l, de-result)
 2:     /* s is the string to be decomposed, l is the length of the itemset
    required to be decomposed, de-result stores the results.
 3:     for all ( i is from l to s.length) do
 4:         decompose(s, i, result, resultend);
 5:         de-result ← i+resultend;
 6:     end for
 7: end function
 8: function DECOMPOSE(s, m, result, resultend)
 9:     if (m == 0) then
10:         resultend.addAll(result);//resultend is a list storing all the i-item
    set
11:         return;
12:     end if
13:     if (s is not null) then
14:         result.add(s[0]+null);
15:         for all (j is from the second value to the last value of s) do
16:             s1[j − 1] ← s[j];
17:         end for
18:         decompose(s1, m - 1, result, resultend); //when selecting the first
    item
19:         result.remove(result.size() - 1);
20:         desompose(s1, m, result, resultend); //when the first item is not
    selected
21:     end if
22: end function
```

Fig.2.Algorithm for Decomposed string

Thus, the decomposition cost is exponentially proportional to the itemset's length. In other words, when the itemset length is going up, the decomposition overhead willdramatically enlarged. The data skewness problem is mainly induced by the decomposition operation, which in turn has a significant performance impact on FiDoop. The first step toward balancing load among data nodes of a Hadoop cluster is to quantitatively measure the total computing load of processing local itemsets. We achieve this first step by developing a workload-balance metric to quantify load balance among the data nodes. In Proposed System In base system having dimension reduction issue, in proposed we need to focus eliminate such problems. The system also focuses on SQL injection and prevention as well as data collusion attacks. Develop the system in HDFS 2.0 with MongoDB with 16 cluster node environment. Proposed system use HDFS framework with R package called R-hadoop. The proposed system can extends up to node cluster. We also use transaction management system base on ACID properties which will help for avoid data inconsistency.

CONCLUSION

To solve the scalability and load balancing challenges in the existing parallel mining algorithms for frequent itemsets, we applied the MapReduce programming model to develop a parallel frequent itemsets mining algorithm called FiDoop. FiDoop incorporates the frequent items ultrametric tree or FIU-tree rather than conventional FP trees, thereby achieving compressed storage and avoiding the necessity to build conditional pattern bases. FiDoop seamlessly integrates threeMapReduce jobs to accomplish parallel mining of frequent itemsets. The third MapReduce job plays an important role in parallel mining; its mappers independently decompose itemsets whereas its reducers construct small ultrametric trees to be separately mined. We improve the performance of FiDoop by balancing I/O load across data nodes of a cluster.

REFERENCES

[1] Yi Yao, Jiayin Wang, Bo Sheng, Chiu C. Tan, NingfangMi, "Self-Adjusting Slot Configurations for Homogeneous and Heterogeneous HadoopClusters " 2168-7161 (c) 2015 IEEE.

[2] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters "COMMUNICATIONS OF THE ACM January 2008/Vol. 51, No. 1.

[3] R. Agrawal, T. Imieli´nski, and A. Swami, "Mining association rules between sets of items in large databases," ACM SIGMOD Rec., vol. 22, no. 2, pp. 207–216, 1993.

[4]A. Schuster and R. Wolff, "Communication-efficient distributed mining of association rules," Data Min. Knowl. Disc., vol. 8, no. 2, pp. 171–196, 2004.

[5] Jong Soo Park; Ming-Syan Chen and Philip S. Yu, "Efficient Parallel Data Mining for Association Rules "

[6] Sandy Moens, EminAksehirli and Bart Goethals, "Frequent Itemset Mining for Big Data ", 2013 IEEE International Conference on Big Data, 978-1-4799-1293-3/13/$31.00 ©2013 IEEE

[7] TruptiKenekar, A. R. Dani, "A Study of Differentially Private Frequent Itemset Mining" International Journal of Science and Research (IJSR), Volume 4 Issue 10, October 2015

[8] Wei. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using MapReduce," Proc. VLDB Endow., vol. 5, no. 10, pp. 1016–1027, 2012.

[9] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," Data Min. Knowl. Disc., vol. 8, no. 1, pp. 53–87, 2004.

[10] Shekhar Gupta, Christian Fritz, Johan de Kleer, and CeesWitteveen, "Diagnosing Heterogeneous Hadoop Clusters " 23rd International Workshop on Principles of Diagnosis.