Data Partitioning in Frequent Itemset Mining in Flink

S. R. Shaikh[1] and B. M. Patil[2]

[1,2]*Computer Science and Information Technology Department, MBES COE, Ambajogai, Maharashtra, India.*

ABSTRACT

Mining a frequent itemsets from databases is really hard task to perform. This can be done using a traditional parallel algorithms .Using these algorithms itemsets are equally partitioned among a group of computing nodes. But these algorithms gives a performance issues like high communication and mining overhead induced by redundant transactions transmitted among computing nodes, database has to scan multiple times and huge candidate keys needs to be exchanged between the processor, I/O and synchronization, impracticableness to construct in-memory FP trees to manage large databases. To solve all these problems we have implemented FiDoop a parallel frequent itemset mining algorithm. FiDoop develops frequent items ultrametric tree instead of conventional FP trees to increase the performance of algorithm. It uses three MapReduce jobs and develops a workload balance metric to measure load balance across the nodes of cluster. In this way FiDoop helps to improve the parallel mining of frequent data itemsets by incorporating frequent items ultrametric trees.

Keywords *– Fi-Doop, frequent itemset mining, parallel mining, MapReduce, Hadoop clusters.*

I. INTRODUCTION

FiDoop implements frequent items ultrametric tree instead of conventional FP trees which is constructed using correlations among transactions. It tries to improve locality by reducing the redundant transactions and by placing highly similar transactions into a data partition. The large data itemsets are then distributed across data nodes of a Hadoop cluster to reduce the size of a network and computing loads. As a result it becomes an efficient and performance increasing parallel frequent itemsets mining algorithm. FiDoop uses a MapReduce programming model to implement algorithm. The focus area of FiDoop is to integrate highly possible relevant frequent itemsets according to association rule into a data partition. By which the redundant frequent transactions of itemsets can be truncated from the database.

Frequent itemsets mining (FIM) is a core problem in association rule mining (ARM), sequence mining, and the like. Speeding up the process of FIM is very much necessary, because FIM consumes lots of time due to its high computation and input/ output (I/O) intensity. And the increasing nature of frequent itemsets in database due to daily business transactions it is becoming a excessively large databases to handle by mining algorithms. So the sequential FIM algorithms running on a single machine suffer from performance deterioration. To address this issue, we investigate how to perform FIM using MapReduce—a widely adopted programming model for processing big datasets by exploiting the parallelism among computing nodes of a cluster. We show how to distribute a large dataset over the cluster to balance load across all cluster nodes, thereby optimizing the performance of parallel FIM.[1]

There are two types of algorithms that can be used for mining the frequent itemsets first method is the candidate itemset generation approach and without candidate itemset generation algorithm for example *Apriori* and FP-growth algorithms respectively. Finding association rule for discovering interesting relations among variables and data itemsets in large databases is the core problem in data mining algorithms. Apriori algorithm generates large number of candidate itemsets; Apriori has to repeatedly scan an entire database [2]. In general datasets are stored on local databases at local computers which are then connected to a computer network. Hadoop—one of the most popular MapReduce implementations—is running on clusters where Hadoop distributed file system (HDFS) stores data to provide high aggregate I/O bandwidth.[1]

This paper gives a performance analysis of Java and Flink in FiDoop. In this paper section II givers an overview and important aspects of FiDoop, section III focuses on proposed work and implementation details, section IV gives the analysis report on the basis of observations done in practical, section V and section VI concludes the discussion of proposed work in detail.

II. OVERVIEW OF FIDOOP

The overall goal of FiDoop-DP is to boost the performance of parallel FIM applications running on Hadoop clusters. This goal is achieved in FiDoop-DP by reducing network and computing loads through the elimination of redundant transactions on multiple nodes.[3] We used data partitioning scheme into Hadoop-based frequent-pattern-tree (FP-tree) algorithms. Meanwhile the FP-growth algorithm uses tree structure to reduce the transaction, and keep the relationships between attributes among transactions. In addition to FP-tree algorithms (e.g., FP-Growth [4] and FUIT [5]), other FIM algorithms like Apriori [6] [7] can also benefit from this data partitioning scheme. FiDoop includes three MapReduce jobs for mining data itemsets which are described in detail.

Finding of all frequent data itemsets or frequent one itemsets is done first MapReduce job. The given database is taken as input to a Map tasks and the output produced as a frequent one-itemsets using Reduce tasks. In second MapReduce job database is scanned to generate *k*-itemsets by removing infrequent itemsets from each transaction. In last MapReduce job—the most complicated one of the three—constructs *k*-FIU-tree and mines all frequent *k*-itemsets. The FiDoop focuses on third MapReduce job which is the core to increase the performance of the FiDoop Algorithm.

In this algorithm the various terms considered and implemented are discussed as follows:

A. Association rules:

Association rule identifies relations between different itemsets and helps to predict the future occurrences of data itemsets. It is mostly used in medical research field and supermarkets to identify the disease according to symptoms and to predict the buyers view respectively. Association rule is meant to find frequent patterns, correlations, associations, or causal structures from data sets found in various kinds of databases such as relational databases, transactional databases, and other forms of data repositories.[8]

B. Frequent itemset ultrametric tree:

Frequent items ultrametric trees (FIUT), is one of the efficient method for mining frequent itemsets in a database and uses a a special frequent items ultrametric tree (FIU-tree) structure to enhance its efficiency in obtaining frequent itemsets. FIUT has four major advantages over existing methods. Firstly, it scans the database only twice to minimize I/O overhead. Second, it reduces search space by clustering transactions to partition database. Thirdly, for compressed storage only frequent items in each transaction are inserted as nodes into the FIU-tree. And the fourth advantage is all frequent itemsets are generated by checking the leaves of each FIU-tree, without traversing the tree recursively, which significantly reduces computing time. [9]

C. MapReduce Framework:

Map Reduce is a popular data processing technique used for efficient and fault tolerant workload distribution in large clusters of scientific analysis. It works in two phases namely, Map phase and Reduce phase. The Map phase splits an input data into a large number of fragments to Map tasks across the nodes of a cluster to process. Each Map phase generates a set of intermediate key-value pairs by taking in a key-value pair. The infrequent itemsets pairs are truncated by applying minimum support to local pairs. After the MapReduce runtime system groups and sorts all the intermediate values associated with the same intermediate key, the runtime system delivers the intermediate values to Reduce tasks. Each Reduce task takes in all intermediate pairs associated with a particular key and emits a final set of key value pairs. Both input pairs of Map and the output pairs of Reduce are managed by an underlying distributed file system. MapReduce greatly improves programmability by offering automatic data management, highly scalable, and transparent fault-tolerant processing. [10][11][1]

III. PROPOSED WORK

In proposed work we have used an Apache Flink open source stream processing framework for implementation of FiDoop and compared with Java. Apache Flink is a true streaming engine. It uses streams for all workloads: streaming, SQL, micro-batch and

batch. It provides native support for iterative processing and iterates data by using its streaming architecture. It can also be instructed to only process the parts of the data that have actually changed, thus significantly increases the performance of job. It process every record exactly once hence eliminates duplication. [12]

We have used an Apriori and MapReduce algorithms for finding frequent data itemsets. These algorithms are implemented in Flink framework and also in Java to compare the results. Flink is one of the most recent and pioneering Big Data processing frameworks. It provides processing models for both streaming and batch data, where the batch processing model is treated as a special case of the streaming one (i.e., finite stream). Flink's software stack includes the DataStream and DataSet APIs for processing infinite and finite data, respectively. These two core APIs are built on top of Flink's core dataflow engine and provide operations on data streams or sets such as mapping, filtering, grouping, etc.

Each Flink program is represented by a data-flow graph (i.e., directed acyclic graph - DAG) that gets executed by Flink's dataflow engine [13]. The data flow graphs are composed of stateful operators and intermediate data stream partitions. The execution of each operator is handled by multiple parallel instances whose number is determined by the parallelism level. Each parallel operator instance is executed in an independent task slot on a machine within a cluster of computers [14].

Parallel frequent itemsets mining algorithms based on Apriori can be categorized into two camps, namely, count distribution and data distribution. In the count distribution camp, each processor of a parallel system calculates the local support counts of all candidate itemsets. Then, all processors compute the total support counts of the candidates by exchanging the local support counts.

The inputs given to the algorithm are files with the business transactions of various sizes in MB. The proposed algorithm works in four phases three of which are MapReduce jobs.

Step 1. Parallel Counting: the first MapReduce job simply scans the whole database and counts the support values of all transactions and finds out frequent items in parallel. As we have discussed earlier MapReduce algorithm first divides data in to smaller fragments and then Reduce operation is carried out on that data. The frequency of itemsets is calculated in this phase to easily sort the data.

Step 2. Sorting frequent itemsets: In second step these frequent itemsets are arranged in descending order of frequency. This is done by applying filters after the Reduce phase.

Step 3. Set Frequency Filter: In this phase the all itemsets filters with frequency and minimum number of transactions with frequent itemsets with proper support count are taken in account.

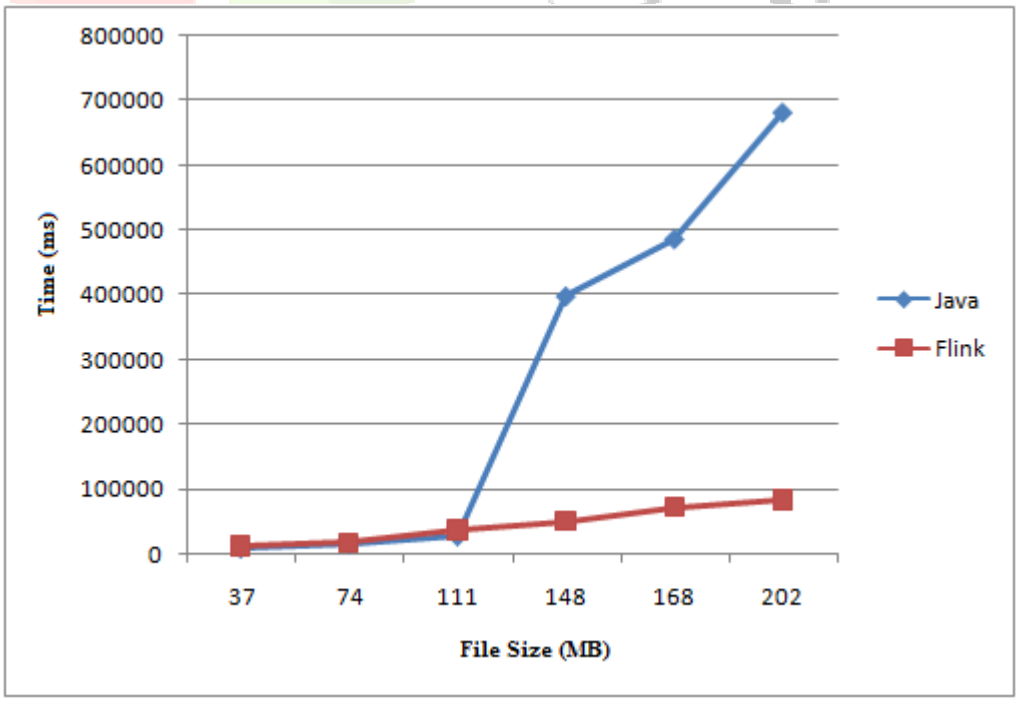

Fig. 1. Performance measurement of Flink and Java

IV. Result Analysis

In fig. 1 we can analyze the performance of Java and Flink for processing data itemset to count frequent itemset. The observation shows that the performance of the Java is limited to 111MB of data when it exceeds this limit it will take too much time to respond or it will generate an out of heap space exception. Instead the best results can be found with implementation of algorithm using Flink. As from the above graph Flink responds in 100 seconds for 202 MB of file to calculate frequent itemset. The Flink algorithm works in parallel fashion to calculate the frequent itemset. There are five main algorithms to calculate and find frequent data itemset as: Input Map function, Item Set Calculate Frequency, Item Set Frequency Filter Function, Item Set Reduce Function, Transaction Group Reduce Function. As per the size of file increases the performance of Java increase up to certain limits and then it starts decreasing even sometimes stops responding by generating exception of heap space but in case Flink it gives a gradual increase performance as size increases.

V. Conclusion

From the above discussion we can conclude that scalability and load balancing challenges in the existing parallel mining algorithm for frequent itemsets we can use FiDoop implementation using flink. We have designed and implemented FiDoop to efficiently handle large databases using Java and flink. And from the observations it is clear that flink will solve the issues found in Java in better way.

References

[1] Yaling Xun, Jifu Zhang, and Xiao Qin, "FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce," IEEE transactions on Systems, Man, And Cybernetics: Systems, vol. 46, no. 3, pp. 313-325, March 2016.

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

[3] Y. Xun, J. Zhang, X. Qin and X. Zhao, "FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 101-114, Jan. 1 2017.

[4] I. Pramudiono and M. Kitsuregawa, "Parallel fp-growth on pc cluster," in Advances in Knowledge Discovery and Data Mining. Springer, 2003, pp. 467–473.

[5] Y. Xun, J. Zhang, and X. Qin, "Fidoop: Parallel mining of frequent itemsets using mapreduce," IEEE Transactions on Systems, Man, and Cybernetics: Systems, doi: 10.1109/TSMC.2015.2437327, 2015.

[6] M.-Y. Lin, P.-Y. Lee, and S.-C. Hsueh, "Apriori-based frequent itemset mining algorithms on mapreduce," in Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ser. ICUIMC '12. New York, NY, USA: ACM, 2012, pp. 76:1–76:8.

[7] X. Lin, "Mr-apriori: Association rules algorithm based on mapreduce," in Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on. IEEE, 2014, pp. 141–144.

[8] Available at [Online]: https://www.techopedia.com/definition/30306/association-rule-mining

[9] Y.-J. Tsay, T.-J. Hsu, and J.-R. Yu, "FIUT: A new method for mining frequent itemsets," Inf. Sci., vol. 179, no. 11, pp. 1724–1737, 2009.

[10] A.Kaviyarasu , J.Gopalakrishnan, M.Nandhakumar, S.Senthilnathan, "A Map-Reduce Framework for parallel mining on Hadoop Cluster", SAJET, vol. 3, no. 9, pp. 39-45, 2017.

[11] S. Sonar, S. Kawad, K. Murudkar, S. Waghere, "Frequent ITEMSET Mining Map Reduce", IFERP, vol. 4, pp. 2394-2320, Nov. 2017.

[12] Available at [Online]: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Feature-wise-comparison-between-Apache-Hadoop-vs-Spark-vs-Flink

[13] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36, 4 (2015).

[14] The Apache Software Foundation. 2014. Apache Flink. https://flink.apache. org/. (2014).