A Review of Scalable Approaches for Frequent Itemset Mining

Daniele Apiletti, Paolo Garza, and Fabio Pulvirenti(✉)

Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy
{daniele.apiletti,paolo.garza,fabio.pulvirenti}@polito.it

Abstract. Frequent Itemset Mining is a popular data mining task with the aim of discovering frequently co-occurring items and, hence, correlations, hidden in data. Many attempts to apply this family of techniques to Big Data have been presented. Unfortunately, few implementations proved to efficiently scale to huge collections of information. This review presents a comparison of a carefully selected subset of the most efficient and scalable approaches. Focusing on Hadoop and Spark platforms, we consider not only the analysis dimensions typical of the data mining domain, but also criteria to be valued in the Big Data environment.

Keywords: Frequent Itemset Mining · MapReduce · Spark · Data mining

1 Introduction

The increasing capabilities of recent applications to produce huge amounts of information has drastically changed the importance of Data Mining. In both academic and industrial domains, the interest towards data mining techniques, which focus on extracting effective and usable knowledge from large collections of data, has risen. In this paper we focus on Frequent Itemset Mining, which is a data mining technique that discovers frequently co-occurring items. Existing itemset mining algorithms revealed to be very efficient on medium-scale datasets but very resource intensive in Big Data contexts. In general, applying data mining techniques to Big Data has often entailed to cope with computational costs that are likely to become bottlenecks when memory-based algorithms are used. For this reason, parallel and distributed approaches based on the MapReduce paradigm [1] have been proposed. Designed to cope with Big Data, the main idea of the MapReduce paradigm consists in splitting the processing of the data into independent tasks, each one working on a chunk of data. Hadoop is the most scalable open-source MapReduce platform. However, in recent years Apache Spark has grown to become a valid alternative platform, that can run on top of Yarn, the Hadoop resource manager. In this survey, we compare a carefully selected subset of Frequent Itemset Mining algorithms that exploit Hadoop and Spark platforms. The paper presents an overview of the challenges and the algorithms, then compares advantages and drawbacks of each approach.

T. Morzy et al. (Eds): ADBIS 2015, CCIS 539, pp. 243–247, 2015.
DOI: 10.1007/978-3-319-23201-0_27

2 Frequent Itemset Mining

The Frequent Itemset Mining [2] process extracts patterns of items from a transactional dataset $\mathcal{D}$, where items correspond to boolean attributes. The support of an itemset I in $\mathcal{D}$ is defined as the ratio between the number of transactions in $\mathcal{D}$ that contains I and the total number of transactions in $\mathcal{D}$. An itemset I is considered frequent if its support is greater than a minimum support threshold.

3 Hadoop and Spark

This survey compares the most scalable itemset mining algorithms exploiting the Hadoop [3] and Spark [4] platforms. Both stems from a distributed programming model introduced by Google, the MapReduce paradigm [1]. MapReduce applications are divided into two major phases that are known as Map and Reduce, divided by a shuffle phase in which data are sorted and aggregated. The Spark framework, with its cached Resilient Distributed Datasets (RDD), usually outperforms Hadoop MapReduce, in particular when iterative processing is required, as in Frequent Itemset Mining. Furthermore, both platforms offer algorithm libraries such as Mahout [5] for Hadoop and MLLib [6] for Spark.

4 Algorithms

This paper compares five selected Hadoop and Spark implementations based on the most scalable Frequent Itemset Mining algorithms.

[7] is a MapReduce implementations of FP-Growth [8] and it has represented for years the only concrete and effective distributed FIM algorithm based on Hadoop. FP-Growth is based on an FP-tree transposition of the transaction dataset and a recursive divide-and-conquer approach. The parallel version initially builds a set of independent FP-trees that are distributed to the cluster nodes. Then, by applying one instance of the (traditional) FP-growth algorithm on each FP-tree, the complete set of frequent itemsets is generated. Since the generated FP-trees are independent, the mining phase can be performed in parallel, i.e., one independent task for each FP-tree is executed to mine a part of the frequent itemsets.

BigFIM and Dist-Eclat [9], instead, are based on the Apriori and Eclat algorithms, respectively. Both of them consists of two phases. The first has the target to find the k-sized prefixes on which, in the second phase, the algorithms build independent subtrees. While the second phase is the same for both the algorithms, in the first phase they extracts the prefixes exploiting two different strategies. BigFIM exploits the Apriori algorithm [10], which uses a bottom up approach: itemsets are extended one item at a time and their frequency is tested against the dataset. Dist-Eclat, instead, uses an Eclat approach to generate the first phase prefixes: Eclat algorithm [11] is based on equivalence classes (groups of itemsets sharing a common prefix), which are smartly merged to obtain all

the candidates. In the second phase, both the algorithms proceed with an Eclat-like independent subtree mining. Dist-Eclat is very fast but, with some prefixes configuration, it assumes that the whole initial dataset (transposed in vertical format) can be stored in nodes main memory. BigFIM proved to be slower than Dist-Eclat but able to process larger datasets, even when Dist-Eclat runs out of memory.

YAFIM in [12] represents, instead, an Apriori distributed implementation developed in Apache Spark. The framework solves the challenges related to the iterative nature of the Apriori algorithm, exploiting Spark RDDs to speed up counting operations.

Finally, Spark PFP [6] represents a pure transposition of FP-Growth to Apache Spark; it is included in MLLib, the Spark Machine Learning library. The algorithm implementation in Spark is very close to the Hadoop sibling, i.e, it first builds independent FP-trees and then invokes the mining step on each tree (one independent task for each FP-tree).

Table 1. Algorithm analysis

Name	Framework	Underlying algorithm	Data distribution	Search Strategy	Communication cost handling	Load balance handling
PFP	Hadoop	FP-Growth	dense	Depth First	Yes	No
Spark PFP	Spark	FP-Growth	dense	Depth First	Yes	No
Dist-Eclat	Hadoop	Eclat	dense	Depth First	Yes (best effort with load balancing)	Yes
BigFIM	Hadoop Hadoop	Apriori and Eclat	dense and sparse	Breadth First and Depth First	Yes (best effort with load balancing)	Yes
YAFIM	Spark	Apriori	sparse	Breadth First	Yes	No

5 Analysis Criteria

In this section we introduce the criteria adopted to evaluate the algorithms. The first set of features are related to the algorithm implementation (e.g., the adopted framework and the underlying algorithm). While Hadoop is an established platform, Spark popularity is growing fast. Hence, Spark implementations are considered more promising and future proof.

Secondly, we consider communication costs and load balancing features. These are two of the most important features in distributed processing but they are often undervalued. Communication cost is a crucial part of the behavior of a parallelized algorithm. It does often overwhelm computation costs and it can become a bottleneck for the overall performance. Load balancing, as well, influences performance and limits the parallelization. Developing an algorithm with a heavy-tailed main reducer that keeps working for a long after all the other nodes have stopped their computation is a common issue.

Finally, we have evaluated the datasets used in the experimental sections of the surveyed papers. Table 1 reports the classification of the five algorithms, based on the criteria described in this section.

6 Frequent Itemset Mining Algorithms Evaluation

From an analytical point of view, BigFIM and Dist-Eclat are the algorithms devoting the most attention to communication costs and load balancing (the other algorithms do not address load balancing at all). For instance, the motivations behind the choice of the length of the prefixes generated during the first step of both algorithms are very interesting. In fact, that choice significantly affects both communication cost and load balancing. The former would benefit of shorter prefixes while the latter would improve with a deeper level of the mining phase before the redistribution of the seeds. Hence, depending on the data distribution and the characteristics of the Hadoop cluster, BigFIM and Dist-Eclat can be tuned to optimize communication cost or load balancing, obviously impacting on the overall execution time.

Parallel FP-Growth (PFP) is based on the generation of independent FP-trees that allow achieving work independence among the nodes. However, the independent FP-trees can have different characteristics (e.g., some are more dense than others) and this factor impacts significantly on the execution time of the mining tasks that are executed independently on each FP-tree. When the FP-trees are significantly different, the tasks are unbalanced, and hence the whole mining process is unbalanced. This problem could be potentially solved by splitting complex trees in sub-trees: however, defining a metric to split a tree is not easy.

YAFIM exploits the Spark architecture and APIs to handle communication costs. Its assumption that all transactions must fit into the RDD may limit its potential. The Spark PFP implementation is integrated in the MLLib collection. It is characterized by dynamic and smooth handling of the different stages of the algorithm, without a strict division in phases. Its main advantage over the Hadoop sibling is the low I/O cost, potentially leading to a single read of the dataset, by loading the transactions in an RDD and processing the data in main memory, whereas the Hadoop-based implementation of PFP performs much more I/O operations.

All surveyed papers show interesting results on very large datasets. Only YAFIM presents results on relatively small datasets, and focuses mainly on the comparison against the Hadoop Apriori implementation. Finally, the number of input parameters, that is another important characteristics of data mining algorithms, is limited for almost all the considered implementations. BigFIM and Dist-Eclat, with their customizable length of first-phase prefixes, could require some experiments to find the proper set of parameters (depending on the dataset distribution and the cluster configuration). However, this parameter allows BigFIM and Dist-Eclat to handle both communication costs and load balancing. Among Hadoop algorithms, relying on experimental evaluations presented in the papers, BigFIM can be considered as the current baseline in this survey. For future perspective and developments, we consider Spark implementations more promising than Hadoop ones, even if the formers currently appear less mature. Spark algorithms have just started to appear in literature and we expect to find more complete implementations very soon.

7 Conclusions

We presented a critical review of scalable Frequent Itemset Mining implementations based on Hadoop and Spark platforms, extending the analysis to critical dimensions such as load balancing and communication costs. Our future plan is to enrich this analysis with real benchmarking to understand the real scalability of each approach.

Acknowledgments. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 619633 (Integrated Project ONTIC).

References

1. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004, p. 10 (2004)
2. Pang-Ning, T., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley (2006)
3. Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project **11**, 21 (2007)
4. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI 2012, p. 2 (2012)
5. The Apache Mahout machine learning library (2013). http://mahout.apache.org/
6. The Apache Spark scalable machine learning library (2013). https://spark.apache.org/mllib/
7. Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E.Y.: PFP: parallel fp-growth for query recommendation. In: RecSys 2008, pp. 107–114 (2008)
8. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: SIGMOD 2000, pp. 1–12 (2000)
9. Moens, S., Aksehirli, E., Goethals, B.: Frequent itemset mining for big data. In: SML: BigData 2013 Workshop on Scalable Machine Learning. IEEE (2013)
10. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 1994, pp. 487–499 (1994)
11. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: KDD 1997, pp. 283–286. AAAI Press (1997)
12. Qiu, H., Gu, R., Yuan, C., Huang, Y.: YAFIM: a parallel frequent itemset mining algorithm with spark. In: IPDPSW 2014, pp. 1664–1671, May 2014