Publication pour la présente candidature LAFPT

Titre	S-FPG: A parallel version of FP-Growth algorithm under Apache Spark™
Auteurs	Aissatou Diaby dite Gassama, Fodé Camara, **Samba Ndiaye**
Référence	International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)
Editeur	IEEE
Pages	98 - 101
Année	2017
DOI	10.1109/ICCCBDA.2017.7951891
URL	**https://ieeexplore.ieee.org/document/7951891**
Index	https://www.scopus.com/authid/detail.uri?authorId=6701604512
ISBN	978-1-5090-4498-6
Encadreur	Oui
Extrait d'une thèse	Non

S-FPG: A parallel version of FP-Growth algorithm under Apache Spark™

3 Author(s) Aissatou Diaby dite Gassama ; Fodé Camara ; Samba Ndiaye View All Authors

Abstract:
Frequent Itemsets Mining (FIM) is an essential data mining task, with many real world applications such as market basket analysis, outlier detection, and so one. Many efficient single-node FIM algorithms such as the well-known FP-Growth algorithm have been proposed in the last two decades. However, as large-scale datasets are usually adopted nowadays, these algorithms become inefficient to mine frequent itemsets over big data. Scalable parallel algorithms hold the key to solving the problem in this context. However, the existing parallel versions of FP-Growth algorithm implemented with the disk-based MapReduce model are not efficient enough for iterative computation. In this paper, we propose an implementation of scalable parallel FP-Growth using the in-memory parallel computing framework Apache Spark™. Our experimental results demonstrated that the proposed algorithm can scale well and efficiently process large datasets.

Published in: 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)

Date of Conference: 28-30 April 2017

Date Added to IEEE *Xplore*: 19 June 2017

ISBN Information:
Electronic ISBN: 978-1-5090-4499-3
Print ISBN: 978-1-5090-4498-6
CD-ROM ISBN: 978-1-5090-4497-9
Print on Demand(PoD) ISBN: 978-1-5090-4500-6

INSPEC Accession Number: 16967014

DOI: 10.1109/ICCCBDA.2017.7951891

Publisher: IEEE

Conference Location: Chengdu, China

S-FPG: A Parallel Version of FP-Growth Algorithm under Apache Spark™

Aissatou Diaby dite Gassama[1], Fodé Camara[2], Samba Ndiaye[1]
[1] Department of mathematics, Cheikh Anta Diop University, Dakar, Senegal
[2] Departement of mathematics, Alioune Diop University, Bambey, Senegal
e-mail: aissatou.gassama@ucad.edu.sn, samba.ndiaye@ucad.edu.sn, fode.camara@uadb.edu.sn

***Abstract*—Frequent Itemsets Mining (FIM) is an essential data mining task, with many real world applications such as market basket analysis, outlier detection, and so one. Many efficient single-node FIM algorithms such as the well-known FP-Growth algorithm have been proposed in the last two decades. However, as large-scale datasets are usually adopted nowadays, these algorithms become inefficient to mine frequent itemsets over big data. Scalable parallel algorithms hold the key to solving the problem in this context. However, the existing parallel versions of FP-Growth algorithm implemented with the disk-based MapReduce model are not efficient enough for iterative computation. In this paper, we propose an implementation of scalable parallel FP-Growth using the in-memory parallel computing framework Apache Spark™. Our experimental results demonstrated that the proposed algorithm can scale well and efficiently process large datasets.**

Keywords-frequent itemset mining; fp-growth algorithm; parallel computing; apache spark™

I. INTRODUCTION

Frequency mining problem represents the core of several data mining algorithms such as association rule mining and sequence mining [1]. It also extended to data mining tasks of classification [2] and clustering [3]. The Frequent Itemsets Mining problem (FIM) consists to find relations between attributes in a database. For example, using shopping tickets, we can find associations between items: for example, we can find that bread is frequently purchased with chocolate or wine and chips are often bought together. Many algorithms were proposed to achieve this task. Some well-known algorithms are Apriori and FP-Growth [1, 4].

As datamining area is facing new challenges in the age of big data, the need to improve the efficiency of classical frequent itemsets mining algorithms is a challenge task for researchers. In fact, the applications of traditional single-node algorithms for frequent itemsets mining in the large-scale datasets will easily cause high CPU consumption, high memory cost, high I/O overhead, low computing performance and some other issues [5]. For these reasons, many researchers have proposed several parallel methods to address these issues.

MapReduce parallel programming model provides the first idea for handling big data, and several MapReduce approaches of parallel frequent itemsets mining have been proposed. Among these, we can cite [5, 6, 7, 8].

However MapReduce parallel programming framework causes very high I/O overhead for iterative computations because it is a disk-based model [5]. Then, MapReduce framework is not suited for the frequent itemsets algorithms which need intensive iterated computation.

For this reason, we propose a new parallel version of FP-Growth algorithm, that we call S-FPG (for Spark FP-Growth), using the Apache Spark™ that is an in-memory-based and iterative computing framework. Our choice of FP-Growth algorithm is motivated by the fact that this algorithm employs a unique search strategy using compact structures resulting in a high performance algorithm that requires only two database passes. In addition, we don't need to iteratively use *k*-frequent itemsets to generate (*k+1*)-frequent itemsets. The experimental results show the efficiency and the scalability of our proposal.

The rest of the paper is organized as follows. Section II presents briefly the basic concepts of FIM. Section III describes the Spark parallel computing framework. Section IV discusses related works. In section V, we give the details of S-FPG algorithm. In Section VI, we evaluate the performance of our algorithm and compare it with existing ones. Section VII concludes the paper and gives some future works.

II. FREQUENT ITEMSET MINING: BACKGROUND

A. Preliminaries

Given a set of items I = {I1, ..., In} and a database *D* as a set of transactions *T*, each transaction is a subset of *I* ($T \subseteq I$) and is identified by an identifier *TID*. An itemset *X* is a subset of items ($X \subseteq I$), and an itemset of length *k* is called a *k*-itemset. The support of an itemset *X* is the percentage of transactions in D that contains *X*. If the support of an itemset is greater than or equal to a given support threshold σ, it is called a frequent itemset. The objective of a FIM algorithm is to find all frequent itemsets, given an input database *D* and a support threshold σ.

The two well-known FIM algorithms are briefly described in the subsections below.

B. Apriori Algorithm

Apriori is the first and the best-known algorithm to mine frequent itemsets [1]. It uses a breadth-first search strategy to count the support of itemsets and uses a candidate generation function which exploits the downward closure property of support. This property states that if an itemset is frequent, then all its subsets are also frequent; and if an itemset is infrequent, then all its supersets must being frequent too.

978-1-5090-4497-9/17/$31.00 ©2017 IEEE

The main drawback of Apriori is the fact that it can be very slow and the bottleneck is the candidate generation step. For example if the database has 10^4 frequent 1-itemsets, they will generate 10^7 2-itemsets candidates even after employing the downward closure. To determine the frequent 2-itemsets, the database needs to be scanned several times. Generally it needs $(n+1)$ scans, where n is the length of the longest pattern.

C. FP-Growth Algorithm

The FP-Growth algorithm [4] uses the frequent pattern tree (FP-Tree) structure. FP-tree is an improved tree structure such that each itemset is stored as a string in the tree along its frequency. In the first pass, the algorithm counts occurrence of items in the dataset, and stores them to a header table. In the second pass, it builds the FP-tree structure by inserting instances. Items in each instance have to be sorted by descending order of their frequency in the dataset, so that the tree can be processed quickly. Items in each instance that frequency is less than minimum threshold are discarded. Then, if many instances share most frequent items, FP-tree provides high compression of initial dataset. Recursive processing of this compressed version of main dataset grows large itemsets directly, instead of generating candidate items and testing them against with the entire dataset. FP-Growth starts from the bottom of the header table (having longest branches), by dividing the compressed dataset (or database) into a set of conditional databases. Each one is associated with one frequent pattern. Finally, the corresponding conditional FP-tree is generated and is mined separately. Using this strategy, the FP-Growth reduces the search costs. Once the recursive process has completed, all large itemsets have been found.

To illustrate the behavior of the FP-Growth algorithm, we consider the following initial dataset Figure 1-(a) and its transformed version within in each transaction is sorted by descending order according to the values of frequent 1-itemset (see Figure 1-(b)). Figure 1-(c) represents the corresponding FP-tree, and Table 1 contains the conditional pattern base, the conditional FP-tree and generated itemsets.

III. Spark Parallel Computing Framework

Apache SparkTM is an open-source cluster computing framework. In contrast to Hadoop's disk-based MapReduce model, Spark's in-memory primitives provide performance up to 10 times faster for certain applications such as FIM [9]. By allowing user programs to load data into a cluster's memory and analyze it iteratively, Apache Spark™ is well suited to data mining algorithms which are often iterative. Apache Spark™ requires a cluster manager and a distributed storage system. For cluster management, Apache Spark™ supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Apache Spark™ can interface with a wide variety of systems, including Hadoop Distributed File System (HDFS), Cassandra and Amazon S3 [9].

IV. Related Works

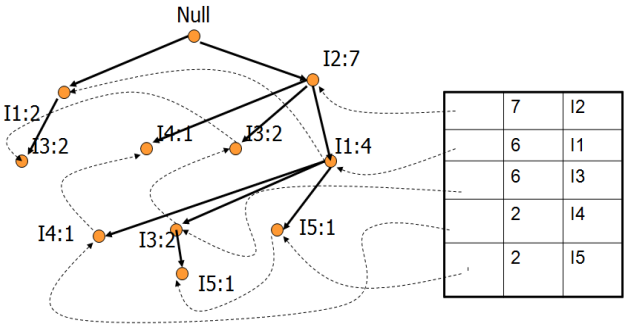
To address the FIM problem over big data issue, several algorithms were proposed [6, 7, 8]. The main drawback of these works is the fact that they are based on MapReduce model.

ID	Items
T100	I1, I2, I5
T200	I2, I4
T300	I1,I3
T400	I1,I2,I4
T500	I2,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3

(a)

ID	Items
T100	I2, I1,I5
T200	I2, I4
T300	I1,I3
T400	I2,I1,I4
T500	I2,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3

(b)



(c)

Figure 1. A sample transaction set, its sorted version and the corresponding FP-tree structure

TABLE I. Conditional Pattern Base and Conditional FP-Tree Generated with FP-Tree Structure Given in Figure 1.

Item	Conditional pattern base	Conditional FP-tree	Genered itemsets
I5	<I2,I1>:1,<I2,I1,I3>:1	I2:2 I1:2	{I2,I5}:2 {I1,I5}:2 {I2,I1,I5}:2
I4	<I2,I1>:1,<I2>:1	I2:2	{I2, I4}:2
I3	<I2,I1>:2,<I2>:2,<I1>:2	I2:4 I1:2	{I2,I3}:4 {I1,I3}:4 {I2,I1,I3}:2
I1	<I2>:4	I2:4	{I2,I1}:4

To the best of our knowledge, the pioneering work of parallel frequent itemsets mining with Apache Spark™ is described in [5].

V. THE S-FPG ALGORITHM

Algorithme 1- SFPG
Input: f, the input file and minimal support d
Output: Complete set of frequent itemsets
Begin
1 : T=flatmap(line=>f.getTransaction())
2 : Foreach transaction line in T
3 : Foreach item I in transaction line
4 : mapToPair(I=>(I,1))
5 : EndForeach
6 : EndForeach
7: F1 = reduceByKey(_+_).filter(item which frequency is greater than d)
8 : Generate the header table from F1
9 : Build the FPTree as follows :
10 : Create root of a FP-tree with label "null"
11 : Forall transaction line ∈ T do
12 : Sort frequent items in transaction line according to F1. Let sorted list be [head|L] where head is the head of the list and L the rest.
13 : addItems([head|L], tree)
14 : end Forall
15 : FP-Growth(tree)
End

Algorithm 2 addItems([head|L], tree)
Begin
1: if root has a child N such that item[N]=item[head] then
2 : count[N] ← count[N] + 1
3 : else
4 : Create new node N with count = 1, parent linked to root and node-link linked to nodes with the same item via next
5 : end if
6 : if head<>null then
7 : addItems (L,N)
8 : end if
End

Algorithm 3 FP-Growth(Tree)
Begin
1 : if Tree contains a single path P then
2 : Generate all combinations of the subpaths β of P. Each β represents a frequent itemset with support is minimum support of nodes init.
3 : else
4 : For all ai in header table of Tree do
5 : Construct ai's conditional pattern base and then ai's conditional FP-Tree Treeai
6 : if Treeai is not null then
7 : FP-Growth(Treeai)
8 : end if
9 : end for
10 : end if
End

VI. PERFORMANCE EVALUATION

The experiments were performed on a cluster consisting of 4 nodes, where each node has 8 cores Intel® Xeon® E5 processors running at 2.60 GHz, with 56 GB memory and a 382 GB disk. The computing nodes are all running at the Windows Server 2012 and Java 1.7.0_55.

We used six benchmark real-world datasets chosen from the FIM repository [10]: retail is market basket data from an anonymous Belgian retail store. Chess is a dataset listing chess end game positions for king vs. king and rook. Dataset bms-pos contains several years of point-of-sale data from a large electronic retailer ; whereas datasets bms-webview1 and bms-webview2 contain several months of clickstream data from an e-commerce website. Some statistics of those datasets are presented in Table II.

TABLE II. CHARACTERISTICS OF BENCHMARK DATASETS

Name	**|T|**	**|I|**	**$\|t\|_{avg}$**
retail	88,162	16,470	10.31
bms-pos	515,597	1,657	6.50
bms-webview1	59,602	497	2.50
bms-webview2	77,512	330,286	5.00
chess	3,196	75	37.00

A. *Speed up Evaluation*

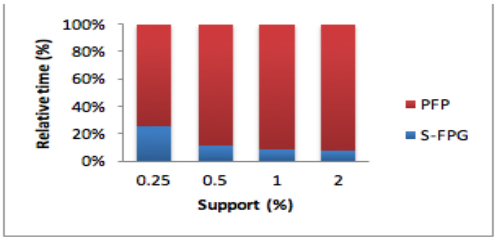
We evaluated the speedup metric by evaluate how much faster S-FPG algorithm is than PFP implementation provided by the Mahout library [11], a corresponding MapReduce algorithm, by decreasing the minimal support while keeping the size of datasets constant.
We measure the speedup metric with the following formula:

$$S = \frac{T_{PFP}}{T_{S-FPG}}$$

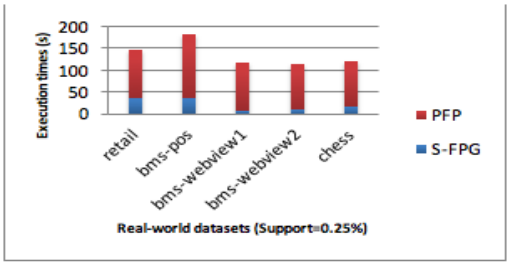
where S is the resultant speedup, $TPFP$ is the PFP execution time and $TS{-}FPG$ is the S-FPG execution time.

The running times of S-FPG and PFP algorithms are plotted with support threshold 0.25% for real-world databases in Figure 2 (b).

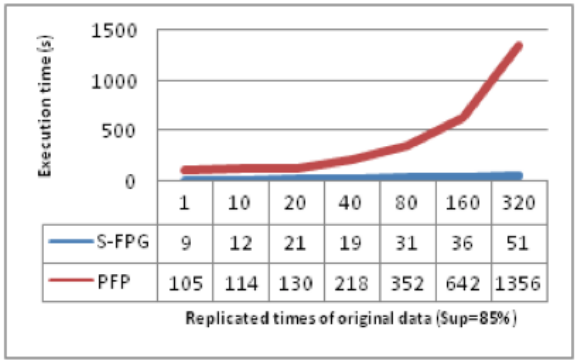
Figure 2 (b) shows that S-FPG outperforms PFP about 15 times in average for bms-webview1, 10 times for bms-webview2, 6 times for chess, 4 times for bms-pos and 3 times for retail dataset. Figure 2 (a) also shows the difference of execution times using 100% stacked columns. We can see that the speedup factor between S-FPG and PFP algorithms decreases when the support threshold is growing.



(a)



(b)



(c)

Figure 2. Experimental results.

B. *Size up Evaluation*

To evaluate the size up factor, we keep the support threshold to 85% by replicating chess dataset 10, 20, 40, 80, 160 and 320 times in order to get larger datasets. Figure 2 (c) shows the data scalability performance. We can see that the execution time cost of our algorithm grows slowly when the size of the dataset increases. In contrast, the execution time of PFP algorithm grows exponentially.

VII. CONCLUSION

In this paper, we presented a new parallel version of FP-Growth algorithm to mine a frequent itemsets from big data. We demonstrated that the proposed algorithm can scale well and efficiently process large datasets. Our proposal confirms the suitability of Apache Spark™ for FIM problem over large databases. In the future, we plan to improve S-FPG algorithm by using a data structure more compact than FP-tree to reduce the memory consumption.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proceedings of ACM SIGMOD, Washington DC*, 1993.

[2] B. Liu, W . Hsu, and Y . Ma. Integrating Classification and Association Rule Mining. *Proceedings ofACM SIGKDD, New York, NY, 1998*.

[3] K. Wang, X. Chu, and B. Liu. Clustering Transactions Using Large Items. *Proceedings of ACMCIKM*, USA, 1999.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In W. Chen, J. Naughton, and P. A. Bernstein, editors, *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12. ACM Press, May 2000.

[5] Hongjian Qiu, Rong Gu, Chunfeng Yuan, Yihua Huang, Yihua Huang. YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark. *Proc. of the 2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops(ParLearning*), pages 1664 - 1671, Phoenix, AZ, USA, May. 19-25, 2014.

[6] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. PFP: Parallel FP-Growth for Query Recommendation. *In Proceedings of the 2008 ACM conference on Recommender systems*.

[7] Li N., Zeng L., He Q. & Shi Z. Parallel Implementation of Apriori Algorithm Based on MapReduce. *Proc. of the 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD '12)*. Kyoto, IEEE: 236 – 241.

[8] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. *In 2013 IEEE International Conference on Big Data*, IEEE, 2013, pages 111–118.

[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley*, July 2011.

[10] Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data, 2004.

[11] Apache Software Foundation, ―Apache Mahout, June 2010, URL http://mahout.apache.org/.