An Effective Algorithm for Frequent Itemset Mining on Hadoop

Ms. Dhamdhere Jyoti L., Prof. Deshpande Kiran B.

Abstract— **Frequent Itemset Mining is one of the exemplary data mining constraint in most of the data mining applications. It requires unmitigatedly extensive computations and I/O traffic capacity. In this paper we attempt to represent one such distributed algorithm which will run on Hadoop – one of the latest most significant distributed frameworks which support mapreduce paradigm. The proposed approach takes into account quintessential characteristics of the Apriori algorithm related to the frequent itemset generation and through a block-based partitioning uses a dynamic workload management. First, we represent a innovative algorithm for determination of extensive itemsets which uses single pass over the data than classical algorithm. Second, we evaluate the idea of item combination which can contribute to the debased adroitness of the algorithm and enables to abbreviate database size at earlier stage thereby reducing computational cost for later processing. The algorithm substantially enhances the performance and achieves high scalability compared to the existing distributed Apriori based approaches. Proposed algorithm is implemented and tested on large scale datasets distributed over a cluster.**

Index Terms— **Apriori Algorithm, Frequent Itemset Mining, Hadoop, Map Reduce, Distributed Computing.**

I. INTRODUCTION

Data mining is the effective process of discovering patterns which are previously unknown and hidden in large datasets. Current developments and advances in many growing areas of engineering, science, business, etc. are producing tremendous amount of data day by day resulting in heavy requirement of storage. The efficiency to process, analyze, and understand these datasets is at the need of several disciplines, including parallel and distributed computing. This is due to their inherent distributed nature, the quality of their content, the size of the datasets and the heterogeneity etc. One of the most important areas of data mining is association rule mining; it is a task is to find all items or subsets of items which frequently occur and the relationship between them. This is achieved in two main steps: finding frequent itemsets and generating association rules. Frequent Itemset Mining (FIM) tries to discover information from database based on frequent occurrences of an event according to the minimum frequency threshold provided by user.

Manuscript received Aug, 2014.

Ms Dhamdhere Jyoti L., *Department of Computer Engineering, SP's IOK-COE, University of Pune, Pune, India,+919881133338.*

Prof. Deshpande Kiran B., *Department of Computer Engineering, SP's IOK-COE, University of Pune, Pune, India.*

Due to limitations of main memory, FIM becomes inefficient on large databases. This problem can be solved by using Apriori algorithm [1][8][13], where database is scanned multiple times for frequency count of each size of candidate itemsets. Unluckily, single machines are unable to fulfill the memory requirements for handling the complete set of candidate itemsets. Also existing algorithms care to control the output and runtime by increasing the minimum frequency threshold, automatically reducing the number of candidate and frequent itemsets [9].

Parallel programming is getting utmost importance to deal with the massive amounts of data, which is produced and consumed every day. Parallel programming architectures and supporting algorithms, can be grouped into two main categories viz. shared memory and distributed (share nothing). On shared memory systems, all processing units can concurrently access a shared memory area. While, distributed systems are composed of processors that have their own internal memories and communicate with each other by passing messages [9]. It is easier to port algorithms to shared memory parallelism, but they are typically not scalable enough [5][6]. Distributed systems, allow quasi linear scalability for well adapted programs. However, it is not always easy to write or even adapt the programs for distributed systems.

Current algorithms like Apriori are good for the databases that are small in size, but if these algorithms are executed on very large databases in parallel on distributed systems the performance can be improved significantly. Hadoop is an open source distributed framework which is designed based on the Google's Map-reduce programming model [32][33]. Hadoop is capable of analyzing large amount of data. Hadoop is developed by keeping most of the things in mind like-large dataset, write once read many access models, moving computation is cheaper than moving data etc. Apache Hadoop wins terabyte sort benchmark in July 2008. All this capability makes Hadoop suitable for most mining problems. Hadoop has its own file system called Hadoop Distributed File system (HDFS) which is capable of running on commodity hardware with high fault tolerance ability. Data replication is one of the important features of HDFS, which ensures data availability and automatic re-execution on multiple node failure. In this paper we have proposed algorithm which will use the power of Hadoop for mining the frequent Itemset.

This paper is organized as follows: Section II is for background and literature survey, Section III describes the

Problem Statement and application of Hadoop to solve this problem and implementation details of the proposed system whereas section IV has proposed algorithm and analytical discussion. Section V describes about data set used and finally Section VI concludes this paper.

I. Background

Frequent itemsets is considered to be very important in many data mining tasks that try to discover interesting patterns from databases, such as correlations, association rules, episodes, sequences, clusters, classifiers and many more where association rule mining is the most popular problem [7][8]. The original stimulation of interest for searching association rules came from the need of detail examination of so called supermarket transaction data, i.e. to study customer behavior in terms of the purchased products. Association rules tells how frequently the items are purchased together. For example, an association rule (bread) -> (eggs) (80%) states that four out of five customers that bought bread also bought eggs. Such rules can be matter of importance for decisions about store layout, product pricing, promotions and many others.

With the introduction of algorithm by Agrawal et al. since 1993[1][2], the problem of mining the frequent itemset and association rule are considered to be of utmost important[9][11]. Within the past decade, numerous research papers have been published presenting novel algorithms or improvements on existing algorithms to solve these mining problems more efficiently[3][4][6].

Many variants and improvements of this algorithm have been developed suitable in parallel and distributed systems, such as CD [1][7], FDM [21]. Some distributed approaches are based on different sequential algorithms, such as the FP-Growth algorithm[2][18], the D Sampling algorithm, which is combination of the Sampling algorithm and the DDM approach [31].

A. *The Apriori Algorithm*

AIS algorithm by Agrawal et al. was the first algorithm which generates all frequent itemsets and confident association rules with introduction of this mining problem [1]. Agrawal et al. improved the same algorithm and renamed it as Apriori which makes use of monotonicity property of the support of itemsets and the confidence of association rules [2][7].

Apriori algorithm is a classic algorithm for finding frequent itemsets which is mainly based on level wise search and iteratively discover frequent itemsets with size from 1 to k-itemset.

Basic idea is to minimize the search space by using the Apriori principle:

- An itemset must be frequent if and only if all of its subsets are frequent.
- That is, if $\{AB\}$ is a frequent itemset, then both $\{A\}$ and $\{B\}$ should be frequent.

If there are `n` 1-itemsets that satisfy your minimum support, Apriori and many other algorithms must consider `n*(n-1)/2` 2-itemsets. This of course gets rather expensive. In Apriori, the 2-itemsets often is the largest and most expensive step and 3-itemsets may be worse.

B. *A Frequent-Pattern Tree Approach*

Mining frequent patterns in time-series databases, transaction databases, and various kinds of databases has been studied and analyzed popularly in data mining research. Moreover, candidate set generation is still very expensive, especially when we deal with large number of patterns and / or long patterns [2]. J. Han, J. Pei, and Y. Yin had proposed a novel frequent-pattern tree (FP-tree) structure [18]. This is an extended prefix-tree structure used for storing compressed and crucial information about frequent patterns. It uses an efficient FP-growth mining approach which is FP tree based, focusing on the concept of pattern fragment growth for the complete set of frequent patterns.

The main advantage of FP-growth is that each linked list, starting from an item in the header table representing the cover of that item, is stored in a compressed form [7]. Unfortunately, to accomplish this gain, it needs to maintain a complex data structure and perform a lot of dereferencing and also the FP-tree representation is often much larger.

C. *Sampling*

The sampling algorithm, proposed by Toivonen [30], performs at most two scans through the database by picking a random sample from the database, then finding all relatively frequent patterns in that sample, and then verifying the results with the rest of the database. In the cases where the sampling method does not produce all frequent patterns, the missing patterns can be found by generating all remaining potentially frequent patterns and verifying their supports during a second pass through the database. By decreasing the support threshold, the probability of such a failure can be avoided. However, for a reasonably small probability of failure, the threshold must be drastically decreased, which can cause a combinatorial explosion of the number of candidate patterns.

D. *Partitioning*

The Partition algorithm, proposed by Savasere et al. uses an approach which is completely different from all previous approaches [20]. Database is stored in main memory using the vertical database layout and the support of an itemset is computed by intersecting the covers of two of its subsets. More specifically, for every frequent item, the algorithm stores its cover. To compute the support of a candidate k-itemset I, which is generated by joining two of its subsets X, Y as in the Apriori algorithm, it intersects the covers of X and Y , resulting in the cover of I.

Of course, storing the covers of all items actually means that the complete database is read into main memory. For large databases, this could be impossible. Therefore, the Partition algorithm uses the following trick. The database is partitioned into several disjoint parts and the algorithm generates for every part all itemsets that are relatively frequent within that part, using the algorithm described in the previous paragraph and shown in Algorithm 4[20]. The parts of the database are chosen in such a way that each part fits into main memory on itself.

The algorithm merges all relatively frequent itemsets of every part together. This results in a superset of all frequent itemsets over the complete database, since an itemset that is frequent in the complete database must be relatively frequent in one of the parts. Then, the actual supports of all itemsets are computed during a second scan through the database. Again, every part is read into main memory using the vertical database layout and the support of every itemset is computed by intersecting the covers of all items occurring in that itemset.

II. APPROACH DESCRIPTION

This section defines the problem definition, proposed algorithm, system and implementation and gives an analytical discussion that describes the proposed approach.

A. *Problem Definition*

The mathematical presentation of the basic concept of support count in apriori algorithm presented in [1]. Let $I = \{i_1, i_2, i_3,\ldots,i_m\}$ be the set of items. Let $T = \{t_1, t_2, t_3,\ldots,t_n\}$ be the set of transactions, where each transaction t is a set of items such that $t \subseteq I$. The item X has support s in the transaction set T is s% of transactions contain X denoted as s = support(X). An association rule can be defined as $A \rightarrow B$, where $\{A, B\} \subseteq I$ and $A \cap B = \phi$. The support of rule $A \rightarrow B$ is support $(A \cup B)$. The problem of mining association rule is to find all the rules that satisfy a user specified minimum support threshold. The itemset X is said to be frequent if its support is greater than or equal to the user defined minimum support threshold and also all of its subsets are also frequent.

Block abstraction policy is very beneficial for a distributed file system. First, for large datasets, it is not required that the blocks from a dataset to be stored on the same disk, so they can take advantage of any of the disks in the cluster. In fact, it would be possible to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster [32][33]. Second, for more simplified subsystem a block can be considered as a unit of abstraction instead of a file. The storage management is also simplified since blocks are of fixed size, thus eliminates metadata concerns. Also, for providing fault tolerance and availability, blocks fit well with replication.

Here we consider the block partitioning for the distribution of the datasets among all processing nodes. The dataset W is divided among M nodes with D transactions as $\{T_1, T_2, \ldots, T_M\}$. Here every block constitutes transactions that are assigned to the nodes. Let's assume size of the partition Ti as Di. Now each partition Ti is divided into bi blocks $\{t_1,t_2,\ldots,t_{bi}\}$. The size of a block t_i is defined as a default value of 64MB or according to the available memory in the processing node N_i and number of items, the average transaction width, and also the support threshold of a dataset. For a given minimum support threshold delta, an itemset x is *globally frequent* if it is frequent in W; its support x.support is greater than delta × D, and is *locally frequent* in a node Ni if it is frequent in Ti; its support x.support is greater than delta × Di.

B. *Proposed System*

As Hadoop requires data in the form of key-value pairs as input and output, it need to first arrange the transactional data into a suitable format where key is the transaction ID or offset of every line and values will be the comma separated list of items in that transaction. A simple two phase and loosely coupled architecture can be used to implement large range of data mining problem. Also for each phase in Map-Reduce, the data should be in the form of key and value pairs so we need to decide on the intermediate key value structure. These intermediate key value pairs are passed to the reduce phase at the end of the map phase to extract the frequency count of itemsets.

Primarily, the algorithm consists of two phases. The first phase consists of the combination generation which can be seen as main-steps. Each main-step consists of a calculation on the local dataset blocks, or a received block in the formation of ${}^{n}C_k$ combinations, and the potential communication exchanges for the workload and requests management. The second phase consists of a synchronization which is carried out at the end of the first phase for final results aggregation. In the beginning of the first phase, each node Ni is assigned a block of its dataset portion Di, its number of blocks bi, and a workload vector Vi. This information is used to determine a remote performance time when another processing node transfers a job request to some other node.

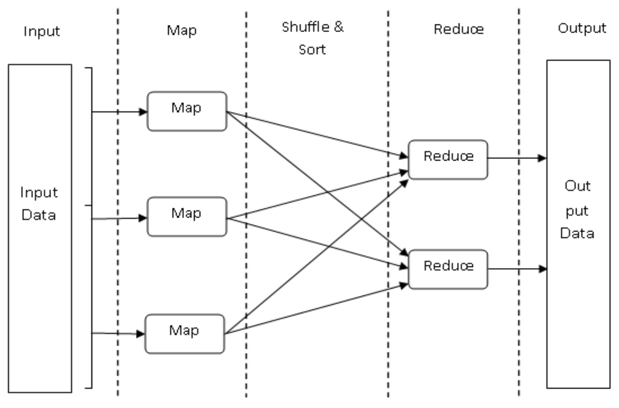


Fig. 1: Map-Reduce Dataflow

C. *System Layout*

As per previous discussion, number of items in the dataset and the support threshold affects the computation complexity of the Apriori generation. It is clear that the combinations of candidate sets can exponentially grow in the Apriori generation process resulting in high demand in memory space. This gives rise either to a thrashing effect, which can remarkably degrade the performance, or unable to deal with the dataset if the implementation is not adapted to out of core computations. Each node performs then the ${}^{n}C_k$ combination generation in its blocks independently of the others. If a given node finishes its blocks processing, it selects a processor that has not issued its end notification yet. This policy achieves workload balance and fast execution time.

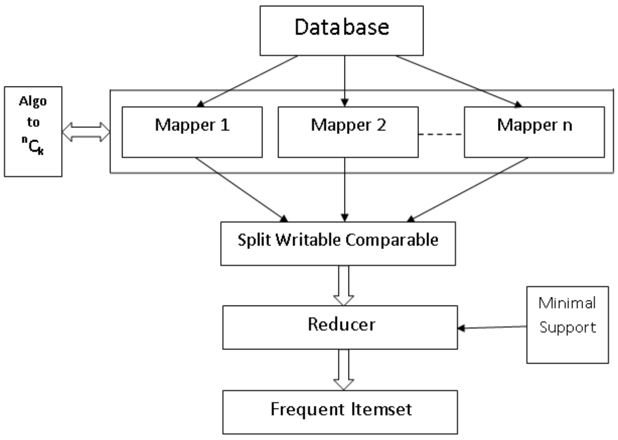


Fig 2: Proposed system layout

The proposed algorithm is based on Hadoop Map-Reduce programming. The database is organized in such a format that each line contains information for a particular transaction. In the proposed system is using TextInputFormat as Input format so each mapper will get each transaction as input. On receiving items in transaction the map phase will create nCk combination of all items in that transaction to reduce the database scan. For each such combination generated in map phase will emit the key as item combination and value as TransactionId.

The output of all map phase will be given to the shuffling and sorting phase. To compare the item pair combination, we have written a customized comparator. Each reducer will get key as item pair and value as list of all transactions in which that item pair occur. For each such key value pair the reducer will calculate the sum of all transaction in which that item pair occur and compare it with minimal support and emit the output item pair as key and value as null. In this way we get frequent item pairs with less database scan but it will increase the number of in-memory computation to generate combination. Following algorithm illustrates the mapper and reducer used for Apriori Algorithm.

III. PROPOSED ALGORITHM

The proposed algorithm is a basic distributed implementation of the Apriori algorithm and it is easy to adapt to map reduce. Algorithm 1 and 2 shows the pseudo codes of the appropriate mapper and reducer. Figure 3 gives the overview of the communication of the algorithm.

- Input:-
  D-Dataset containing different transactions with itemsets
  S- Minimal Support

- **Algorithm 1:- Mapper of the proposed algorithm**

```
Map<Transaction_id, Itemset>
{
//Split itemset based on space
String items [] =itemset.split (" ")

for (int k: items.length)
 {

//Generate ⁿCₖ combinations to reduce database
//scan
item_combinations=generateCombinations(n,k)
}

for(item_combination:item_combinations)
{
//For each combination emits combination as key
//and  txn_id as value
  emit(item_combination,txn_id)
}
}
```

In Map function of the proposed algorithm every row of the given dataset is treated as single transaction and is assigned a unique transaction ID as txnid. The itemsets in a transaction are split based on space. If any transaction has more than 9 items then that particular transaction is further split into sub transaction with subtxnid to speed up the process. Next step is to form all possible combinations for every transaction. Formation of ^nC_k combinations reduces the database scans. Every combination is checked for symmetric property to avoid duplicate keys. The output of mapper is in the form of key value pair and is provide as input o the reducer.

- **Algorithm 2:- Reducer of the proposed algorithm**

```
Reduce<item_combination,Iterable<txnids>
{
int count=0;
//Count the occurrence of each txnid in which
//item_combination occurs
For( txnid:txnids)
{
  count++;
}
If(count>=minimum_support)
{
//if count>=minimum_support  emit
//item_combination as key, and Value will be null
        Emit(item_combination,NullWritable);
}
}
```

Input to the reducer is key value pair where key is itemset {2,3} and value is in the form of <1,1,1> i.e. number of occurrences of that itemset. The reducer simply adds the values from key value pair and check for whether it satisfies the minimum support. If yes then that particular itemset is considered frequent and written in output file.

A. *Analytical Discussion*

Considering the basic Apriori implementation without workload management:

$$Cost = bi\ pi$$

This is actual cost of the first phase. Thus pi can be calculated as summation of generation of candidate set of Aprori:

$$Pi = \sum_{j=1}^{n} Pi,j$$

According to the experimentation, pi,j will be similar in this implementation as the candidate sets are very close in both cases. So the generation cost for the first phase is:

$$Cost_{FIM} = bi \sum_{j=1}^{n}(Pi.j + (m-1)Ci,j)$$

where ci,j is the frequent itemsets, and (m-1) ci,j is the communication cost. In this approach one step of communications is directly proportional to the iteration. Here synchronization is implicit in each communication step. Now consider the case using the workload management, and the additional parameter bci which is nothing but the communication cost of a local block at a given site Ni. The maximum cost of the first phase of our approach is given by:

$$Cost_{max} = (\arg\max_{i \in Ni} bw_i\, p_i) + bc_i + p_j$$

where bwi is the number of blocks at the site Ni, and pj is remote computation cost on the site Nj.

Lemma.

$$CPmax \in [C_{FIM} - \sum_{j=1}^{n} \text{argmaxi} \in \text{Ni}\ (M-1)\, \text{ci, j, CFIM})]$$

Proof – when all the blocks are executed locally at a given site in the worst case. This means that CPmax is smaller or equal to argmaxi ∈ Ni bipi[27]. Consider the global candidates set GCi,j obtained using global pruning strategies, and the set LCi,j, the local candidates set obtained only by local pruning. Considering wide range of conditions and datasets, the set GCi,j is very close to the set LCi,j, for i = 1, ..,M and j = 1, .., k. This means that the sum Pi = $\sum_{j=1}^{n} Pi,j$ is quite similar in both cases. Accordingly considering the same conditions, C_{FIM} – bipi = C_{FIM} – bi Pi = $\sum_{j=1}^{n} Pi,j$ is bounded by bi $\sum_{j=1}^{n}(Pi.j + (m-1)Ci,j)$. Therefore, in the case of a single communication step, the difference CPmax – CFIM can reach $\sum_{j=1}^{n}$ argmaxi ∈ Ni (M − 1) ci, j).

It is clear from the test that communication steps costs and synchronization are very important . Further, in the case of data distribution for large datasets, dynamic workload management works very efficiently. The additional communications in forming combination are overlapped by local computations. In this approach, the adopted policy for dynamic job requests and data movement, tends to the additional communication overheads of forming combinations on blocks and are overlapped with the computations.

IV. DATASET

The experiment is done using two real data sets which are publicly available and have different characteristics. One of which is generated by IBM synthetic data generator [1][7] and other is the basket data set, contains transactions from a retail store. Table I shows the number of items and the number of transactions in each data set, and the minimum, maximum and average length of the transactions. Additionally, Table II shows for each data set the lowest minimal support threshold that was used in experiments, the number of frequent items and itemsets, and the size of the longest frequent itemset that was found.

A. *Data Set Characteristics*

Table I. Number of Items & Transactions in each Data Set

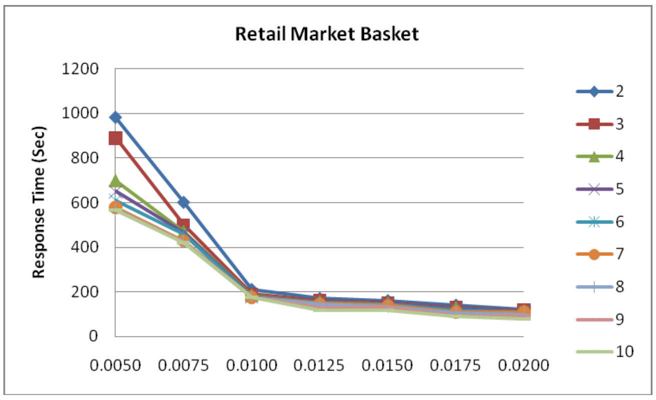
Data Set	Σ	$\lvert F1 \rvert$	$\lvert F \rvert$	Max { k $\lvert$ $\lvert F_k \rvert > 0$ }
T20I7D500K	700	804	550126	18
Retail Market Basket	7	8051	285758	11

Table II. Lowest Min Threshold for each Data Set

Data Set	#Items	#Transactions	Mn$\lvert T \rvert$	Max$\lvert T \rvert$	Avg$\lvert T \rvert$
T20I7D500 K	942	90000	4	77	39
Retail Market Basket	16470	88163	1	51	13

The input which is expected to be given to the proposed model is as follows

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
38 39 47 48 38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62 3 39 48 63 64 65 66 67 68 32 69
48 70 71 72 39 73 74 75 76 77 78 79
36 38 39 41 48 79 80 81
82 83 84 41 85 86 87 88
39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
36 38 39 48 89
39 41 102 103 104 105 106 107 108



Improved Algorithm performance with Minimum Support

V. Conclusions

In this paper, presented a new map-reduce based algorithm addressing problem of mining frequent itemsets using dynamic workload management through a block-based partitioning. The block-based approach deals with memory constraints since the basic task of generating combinations may need very large memory space depending on several parameters including the support threshold. Our approach also exploits fundamental property of the itemsets generation task that shows that the arbiter communication steps, in exemplary implementations such as the FIM approach, are performance constraining. Literally, global pruning strategies bring off enough valuable information in comparison to the generated synchronization and I/O overheads. The features of proposed algorithm are it uses a properly tuned estimation to measure the correct itemset during the pass. It incorporates management of buffer and ensures completeness. The redundancy is eliminated by handling duplicates carefully and the workload management. It shows that the proposed algorithm achieves very pleasing performance and high scalability compared to a classical Apriori-based implementation.

Acknowledgment

Authors sincerely thank the all anonymous researchers for constructive suggestions. Authors would also thank the college authorities for providing the required infrastructure and support. First author is very thankful to Prof. Kiran B. Deshpande for his guidance.

References

[1] Ferenc Kovacs and Janos Illes Frequent Itemset Mining on Hadoop, ICCC 2013 IEEE 9th International conference on Computational Cybrnetics, Tihany, Hungary, July 8-0, 2013.

[2] J. Han, J. Pei, and Y. yin. Mining Frequent Pattern Without Candidate Generation. A frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 2003

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In J.B. Bocca, M. Jarke, and C. Zaniolo, editors, Proceedings 20th International Conference on Very Large Data Bases, pages 487–499. Morgan Kaufmann, 1994.

[4] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, volume 22(2) of SIGMOD Record, pages 207–216. ACM Press, 1993

[5] R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad. A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing, 61(3):350–371, March 2001.

[6] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, Advances in Knowledge Discovery and Data Mining, pages 307–328. MIT Press, 1996

[7] Bart Goethals. Survey on Frequent Pattern Mining, HIIT Basic Research Unit Department of Computer Science University of Helsinki P.O. box 26, FIN-00014 Helsinki Finlan.

[8] Lin, Ming-Yen and Lee, Pei-Yu and Hsueh, Sue-Chen Apriori-based frequent itemset mining algorithm on Mapreduce, ICUIMC'12 Proceeding of the 6th International Conference on Ubiquitous Information Management and Communication, 2012.

[9] Sandy Moens, Emin Aksehirli and Bart Goethals, Frequent Itemset Mining for Big Data, Universiteit Antwerpen, Belgium.

[10] R. Agrawal and R. Srikant. Quest Synthetic Data Generator. IBM Almaden Research Center, San Jose, California. 38

[11] R.J. Bayardo, Jr. Efficiently mining long patterns from databases. In L.M. Haas and A. Tiwary, editors, Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, volume 27(2) of SIGMOD Record, pages 85–93. ACM Press, 1998.

[12] C.L. Blake and C.J. Merz. UCI Repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[13] C. Borgelt and R. Kruse. Induction of association rules: Apriori implementation. In W. H¨ardle and B. R¨onz, editors, Proceedings of the 15th Conference on Computational Statistics, pages 395–400, 2002.

[14] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. Data Mining and Knowledge Discovery, 2003. To appear.

[15] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, volume 26(2) of SIGMOD Record, pages 255–264. ACM Press, 1997.

[16] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, Proceedings of the 39 6th European Conference on Principles of Data Mining and Knowledge Discovery, volume 2431 of Lecture Notes in Computer Science, pages 74–85. Springer, 2002.

[17] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In Chen et al., pages 1–12.

[18] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 2003.

[19] J. Hipp, U. G¨untzer, and G. Nakhaeizadeh. Mining association rules: Deriving a superior algorithm by analyzing today's approaches. In D.A. Zighed, H.J. Komorowski, and J.M. Zytkow, editors, Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, volume 1910 of Lecture Notes in Computer Science, pages 159–168. Springer, 2000.

[20] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDDCup 2000 organizers' report: Peeling the onion. SIGKDD Explorations, 2(2):86–98, 2000. http://www.ecn.purdue.edu/KDDCUP.

[21] CHEUNG, D. W., HAN, J., NG, V. T., FU, A. W., AND FU, Y. 1996. A fast distributed algorithm for mining association rules. In PDIS: nternational Conference on Parallel and Distributed Information Systems. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD.

[22] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery, 1(3):241–258, November 1997.

[23] H. Mannila, H. Toivonen, and A.I. Verkamo. Efficient algorithms for discovering association rules. In U.M. Fayyad and R. Uthurusamy, editors, Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, pages 181–192. AAAI Press, 1994.

[24] S. Orlando, P. Palmerini, and R. Perego. Enhancing the apriori algorithm for frequent set counting. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery, volume 2114 of Lecture Notes in Computer Science, pages 71–82. Springer, 2001.

[25] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and resource-aware mining of frequent sets. In V. Kumar, S. Tsumoto, P.S.

Yu, and N.Zhong, editors, Proceedings of the 2002 IEEE International Conference on Data Mining. IEEE Computer Society, 2002. To appear.

[26] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In Dayal et al., pages 432–444.

[27] Lamine M. Aouad, Nhien-An Le-Khac and Tahar M. Kechadi, Distributed Frequent Itemsets Mining in Heterogeneous Platforms, Journal of ngineering Computing and Architecture, Volume 1, Issue 2, 2007

[28] P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In D. Hand, D. Keim, and R.T. Ng, editors, Proceedings of the Eight ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, pages 32–41. ACM Press, 2002.

[29] M.J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In R. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, Proceedings of the Second SIAM International Conference on Data Mining, 2002. 42

[30] M.J. Zaki. Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12(3):372–390, May/June 2000.

[31] H. Toivonen. Sampling large databases for association rules. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, Proceedings 22nd International Conference on Very Large Data Bases, pages 134–145. Morgan Kaufmann, 1996.

[32] http://hadoop.apache.org

[33] http://www.apache.org