ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

Probabilistic static load-balancing of parallel Mining of repeated series

S. Guha priya
M. C. A, M. Phil (computer science)
VELS University
Chennai- 117

Dr.Y.Kalpana
M. C. A, M.phil, phd.,
VELS University
Chennai- 117

ABSTRACT

Repeated series mining is well known and well studied trouble in data mining. The productivity of the algorithm is used in many other regions like chemistry, bioinformatics, and market basket analysis. Unfortunately the repeated series removal is computationally quite expensive. In this project we present a novel parallel algorithm for removal of repeated series based on a static load-balancing. The static load-balancing is done by measuring the computational time using a probabilistic algorithm. For logical size of occurrence, the algorithms achieve speedups up to ≈3/4 .P where P is the number of processors. In the investigational estimation, we show that our technique performs significantly better than the present state-of-the-art techniques. The presented approach is very universal: it can be used for static load-balancing of additional pattern removal algorithms such as itemset/ graph /tree mining algorithms.

Index Terms—Data mining, static load-balancing, parallel algorithms, Repeated series mining, probabilistic algorithms.

1. INTRODUCTION

Repeated pattern removal is an important data mining technique with a wide variety of mined patterns. The mined frequent patterns can be sets of items (item sets), sequences, graphs, trees, etc. Frequent sequence mining was first described in [1]. The GSP algorithm presented in [1] is the first to solve the problem of frequent sequence mining. As the repeated series removal is an extension of itemset mining, the GSP algorithm is an extension of the Apriori algorithm [2]. As a consequence of the slowness and memory consumption of algorithms described in [2, 1], other algorithms were proposed. These two algorithms use the so-called prefix-based equivalence classes (PBECs in short), i.e., represent the pattern as a string and partition

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

the set of all patterns into disjoint sets using prefixes.

There are two kinds of parallel computers: shared memory technology and distributed memory technology. Parallelizing on the shared memory technology is easier than parallelizing on distributed memory technology. Sampling technique that statically load-balance the computation of parallel frequent itemset mining process, are proposed in [3, 4, 5]. In these three papers, the so-called double sampling process and its three variants were proposed.

1	$\langle(1,2,3)(1,2,4)(2,4,5)\rangle$
2	$\langle(1,4)(2,3,4,5)(1)(2,4,5)\rangle$
3	$\langle(3,4,5)(1,2,3)(1)\rangle$
4	$\langle(3)(1)(2,3,5)\rangle$
5	$\langle(2,5)(1,3,5)(1,2,5)(2,4,5)\rangle$

Fig. 1. An example database, which will be used to demonstrate the algorithm.

2. BASIC NOTION

We denote the number of processors by P and each individual processor by pi, 1 <= i <=P. Let us have a set of items B = {e_i} with an arbitrary ordering operation < on the elements of B. We call an n-tuple E = (e1,…, en) an event if and only if: 1) ei 2 B,2) and ei < ej for i < j. Let us have an event E = (e1,…, en), we define the function (.) as a mapping that takes an event E and creates a set fe1,…, eng. Let us have two events E = (e1,…,m en);E0 = (e0 1,…, e0 m), the concatenation operation _ takes two events and produce another event. The concatenation operation _ is defined by E _ E0 = (e1,…, en; e0 1,…, e0 n). Please note that for the concatenation operation _, we consider only such events E,E' so that E . E' is a correct event.

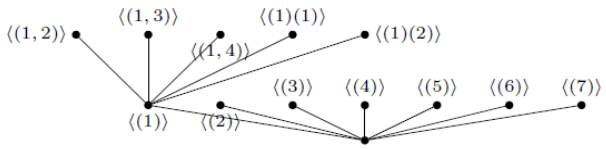

Fig. 2. Let $\mathcal{B} = \{1,2,3,4,5\}$ be a baseset. An incomplete DFS tree of PBECs is shown above. The PBEC $[\langle(1)\rangle]$ is represented by the whole subtree rooted at $\langle(1)\rangle$. Immediate descendant of $[\langle(1)\rangle]$ is $[\langle(1,2)\rangle]$.

For example: E1 = (1, 2), E2 = (4, 5) then (E1) = f1, 2g and (E2) = f4, 5g. An example of concatenated events is E1 _ E2 = (1, 2, 4, 5). The concatenation operation _ may produce invalid events: let us have two events E1 = (1, 2) and E2 = (2, 5) then the concatenation operation would produce (1, 2, 2, 5). However, (1, 2, 2, 5) is not a valid event and is not allowed, see definition of event.

3 BRIEF DESCRIPTION OF THE SEQUENTIAL PREFIXSPAN ALGORITHM

The Prefixspan algorithm uses the notion of the PBEC. As shown in Section 2 the PBECs forms a tree. The tree allows the

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

enumeration of the set of all frequent sequences F. To enumerate F the algorithm explore the tree using the pseudo-projected database and PBECs in the DFS manner, see Section 2. The volumes of the pseudo projected database in terms of the number of transactions give us the support. Hence, for a sequence S the algorithm preserves the pseudo-database as the data structure for accumulating information about the embeddings of S in the database transactions. The algorithm uses the monotonicity principle and computes the support of the possible extensions of sequence S. In this section details are given for the following operations: 1) the collection of frequent extensions; 2) construction of the initial pseudo-projected database; 3) projection using a sequence extension; 4) projection using an event extension.

Algorithm 1 Prefixspan Sequential

PREFIXSPAN-SEQUENTIAL(Database $\mathcal{D}$, Integer min_supp)
1: $\Sigma \leftarrow$ all frequent extensions from $\mathcal{D}$
2: **for** for all $e \in \Sigma$ **do**
3: $Q \leftarrow \langle(e)\rangle$
4: create projection $\mathcal{D}|_Q$
5: PREFIXSPAN-MAINLOOP$(\mathcal{D}, Q, \mathcal{D}|_Q, min_supp)$
6: **end for**

Algorithm 2 Prefixspan sequential – mainloop

PREFIXSPAN-MAINLOOP(Database $\mathcal{D}$, Sequence Q, Database $\mathcal{D}|_Q$, Integer min_supp)
1: Collect all extensions and its support Σ of Q using $\mathcal{D}$ and $\mathcal{D}|_Q$.
2: **for** each $X \in \Sigma$ **do**
3: $Q' \leftarrow Q$ extended with X. $\{X$ *can be either* $e_($ *or* $e, e \in \mathcal{B}\}$
4: **if** $\sigma(Q', \mathcal{D}) < min\_supp$ **then continue**
5: Create $\mathcal{D}|_{Q'}$ using $\mathcal{D}$, $\mathcal{D}|_Q$, and X.
6: PREFIXSPAN-MAINLOOP$(\mathcal{D}, Q', \mathcal{D}|_{Q'}, min_supp)$
7: **end for**

4 OVERVIEW OF EXISTING ALGORITHMS

4.1 overview of sequential algorithms

There are many BFS and DFS sequential algorithms for removal of repeated series. The initial sequential algorithm for removal of repeated series is based on the Apriori algorithm. The Apriori algorithm is a BFS algorithm initially created for removal of repeated item sets [2]. An development of this algorithm, created by the same authors, is the GSP algorithm [1]. Both algorithms use BFS and make multiple passes over the database combined with the monotonicity principle. These two algorithms suffer from similar problems as the Apriori algorithm [3] for frequent itemset removal, e.g., they are slow and needs much more memory, compared to DFS algorithms.

4.2 overview of parallel algorithms

As the frequent sequence mining is computationally quite expensive, there was an effort to come up with parallel

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

algorithms. There are two kinds of parallel computers architectures: 1) shared memory or 2) distributed memory. Parallelization on shared memory computers is quite easy as the hardware supports parallelization. An example of shared memory sequence mining algorithm is in [7]. The problem of mining of frequent sequences on clusters of workstations is hard, because estimating the execution time is an not an easy problem. This section discusses two main groups of the parallel algorithms: 1) algorithms that use the dynamic load-balancing of PBECs; 2) algorithms that use the static load-balancing of PBECs.

4.3 Problems of existing parallel algorithms

The problem with all these algorithms is that they do not load-balance the computation, see Section 9 for comparison with selective sampling. Parallelization of the sequential algorithms is difficult for two main reasons:

(1) Computational complexity: it is well known that estimating the number of frequent itemsets [8, 9] is #P-hard problem. Consider the problem of mining of frequent sequence with only one event. But such problem is similar to the mining of frequent itemsets. This means that estimating the number of frequent sequences is at least #P-hard, see Section 5. From the computational complexity follows that the selective sampling is modifying the task in an unpredictable way.

(2) Size of the PBECs: In order to parallelize the sequential Prefixspan algorithm, we split the set of all frequent sequences using PBECs. The same approach is used in [10]. However, in [10] they use prefixes of size 1. The algorithm [10] does not show how to split the PBECs using longer prefixes.

5 OVERVIEW OF OUR PROPOSED METHOD

Proposed is a novel parallel method that statically load-balance the computation. That is: the set of all frequent sequences is first split into PBECs, the relative execution time of each PBEC is estimated and finally the PBECs are assigned to processors. The method estimates the processing time of one PBEC by the sequential Prefixspan algorithm using sampling. In this section, we explain the intuition behind the process. It is important to be aware that the running time of the sequential algorithm scales with: 1) the database size; 2) the number of frequent sequences; 3) the number of embeddings of a frequent sequence in database transactions.

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

1) The **whole database** D is used to run a sequential algorithm on the database and sample the output of the algorithm, i.e., the set of all frequent sequences F. Such approach does not make sense: the sequential algorithm is executed on the whole database D. Therefore, it runs for at least the same amount of time as the sequential algorithm we use for comparison of the speedup of our parallel algorithm. 2) A **database sample** Ď⊆D is used to run a sequential algorithm using the relative support, producing F '. F' is used as an approximation to F, however, F' can be quite huge. Therefore, the sample F's ⊆F' is used for partitioning and scheduling. Such an algorithm reduces the execution time of the sequential algorithm by reducing the database size: |Ď|<<|D|. For a PBEC [S], the value |[S] ∩F'|/|F'| estimates the relative processing time of a PBEC.|[S]∩F'|/|F'| is estimated by |[S]∩F'$_s$|/|F's|. We call this approach the double sampling process.

6 ESTIMATION OF THE SUPPORT AND THE RELATIVE SIZE OF A PBEC USING SAMPLING

In this Section, we show theoretical justification of the double sampling process. As described in the Section 5, the algorithm creates a sample database Ď⊆D. Ď is then used for computation of the set of all frequent sequences F' using the relative support min_supp*. Afterward, F' is sampled, producing F'$_s$⊆F'. Unfortunately, good bounds on the double sampling process are not available. However, probabilistic guarantees exists on the estimated values, such that: (1) guarantees on the estimate of the support of a sequence, i.e., probabilistic estimate whether a sequence is frequent or not; (2) relative size of a PBEC [S] with S given independently of F's. Combining (1) and (2) should give a good estimate of the relative processing time of the algorithm in a single PBEC.

(1) Estimation of the support: The support of a sequence can be estimated using the Chernoff bounds and indicator variables. The Chernoff bounds and gives us estimate whether a sequence is a subsequence of a transaction in a database or not. We denote by $err_{supp}(S, Ď) = |ϭ(S,D) - ϭ (S, Ď)$ the error of the estimate of the support of sequence S in the database D. Now, we can formulate the Chernoff bounds used for estimation of the support, originally formulated for frequent itemset mining.

(2) Estimation of the relative size of a PBEC: Let us have a sequence S and a PBEC [S]. We want to compute F'$_s$⊆F'such

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

that F' is obtained from Ď using relative support min_supp*. The objective is to estimate the relative size of |[S]∩F'|/|F'|by |[S]∩F'$_s$|/|F'$_s$|. We hope that |[S]∩F|/|F| is approximated by |[S]∩F'|/|F'|, which is approximated by |[S]∩F'$_s$|/|F'$_s$|. Please note that S must be given independently of F'$_s$. The relative size of a PBEC can be used as the estimate of the relative processing time of the PBEC by a sequential algorithm. This estimate ignores some details of the sequential algorithm.

7 WEIGHTING PREFIX TREE

Until now, we have considered the relative processing time of a PBEC, as the relative size of samples in that PBEC. The relative size of a PBEC estimates the relative processing time of that PBEC. The reason is that the processing time increase when the numbers of frequent sequences in a PBEC increase.

(1) Estimating the weight of collection of frequent extensions: From the discussion in Section 3 follows that to collect frequent extensions of S, we have to process items of transactions (k,Q) € D, S ≤ Q. We need to estimate the computational complexity (processing time) of the collection of frequent extensions of the Prefixspan algorithm that has the whole database D as its input. The estimate is computed as the average number of items processed per transaction by the same sequential algorithm with the database sample D' as its input.

(2) Computing the weight of the projection operation: in the prior case, we expected the computational complexity execution time) of the operation of the prefixspan algorithm with the whole database D as its input. We do so by computing the average number of steps of the operation in the database sample Ď. Let us have a pseudo-projected transaction (k,{$l_{1,...,}l_n$})€Ď|s and its corresponding transaction (k,Q})€ Ď.

8 THE PARALLEL PREFIXSPAN ALGORITHM

This section contains the main contribution of the paper. All the ideas presented in the previous sections are integrated here, showing how to execute the Prefixspan in parallel. The parallel Prefixspan algorithm has four phases. In the Phase 1, the method produces the weighting tree T containing the estimates of the relative processing time of the PBECs, see Section 8.1. In the Phase 2, the method partitions the set F into PBECs, using the tree T, and schedule PBECs on processor. In the Phase 3, the method

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

distributes the database in such a way that each processor can process independently its assigned PBECs.

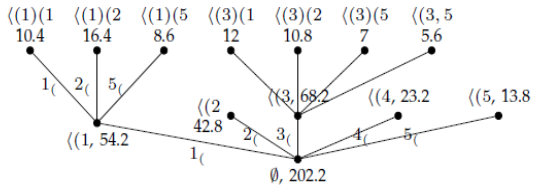

Fig. 3. A part of the weighting tree for the database from Figure 1 for support 4. We omit the subtree rooted at ⟨(2. Let us consider a split for $P = 2$ processors. We need one level of the tree and assign the PBECs with prefix ⟨(1, ⟨(4 to p_1 and ⟨(2, ⟨(3 to p_2. For $P = 8$ we have to further split the PBECs. If we do not have more information about the subtree rooted at ⟨(2, i.e., we cannot split [⟨(2)⟩], and the scheduling results in disbalance, i.e., bad speedup. The weighting tree is constructed using the database from Figure 1.

Algorithm 3 PREFIXSPAN-PARALLEL(Database $\mathcal{D}$, Integer *min_supp*)

1: Phase 1: Execute an arbitrary frequent sequence miner on database sample $\tilde{\mathcal{D}} \subseteq \mathcal{D}$, create $\mathcal{T}$.
2: Phase 2: Partition $\mathcal{F}$ into PBECs and schedule them on the processors.
3: Phase 3: Exchange database among processors.
4: Phase 4: Execute an arbitrary sequence miner in the assigned PBECs.

The motivation behind Algorithm 3 is that the algorithm time increase when: 1) dataset size increase; 2) the support decreases, or in another words when the size of F increase. See Section 5 for details.

9. EXPERIMENTAL EVALUATIONS

In this section, we experimentally evaluate the proposed method. The whole algorithm was implemented in C++ (compiled with gcc 4.4) using MPI, resulting in _ 30'000 lines of code. The implementation was executed on the CESNET metacentrum on the zegox cluster. Each zegox's node contains two Intel E5-2620 equipped with 1_-Infiniband. Nodes were exclusively allocated for these measurements and used a maximum of 5 cores per node (to avoid influences from other jobs).

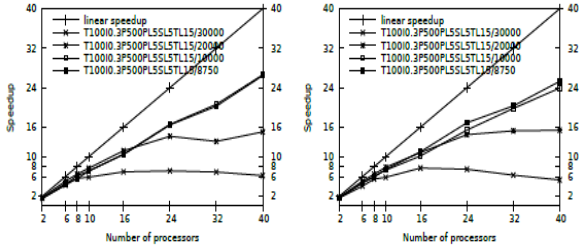

Fig.4. 1) k-depth weighting tree; 2) sample weighting tree;

One event was made from ids of the resources fetched in a window of 10 seconds by one IP address. From the transactions, items were removed if presented in every transaction. In Figure 4 are shown the speedups of our method. All of the proposed methods have speedups up to 20– 32 on 40 processors for lower values of support. These three methods exhibits similar performance on the datasets generated using the IBM generator. The speedups are lower, for higher values of support. For example, the T1000I0.3P500PL5SL5TL15 dataset has quite good speedups for supports 10'000 an 8'750 and bad speedups for supports 30'000 and 20'000.

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

10. CONCLUSIONS

We have proposed an algorithm for mining of frequent sequences using static load-balancing. The method creates a sample of frequent sequences and use this sample for estimating the relative processing time of the algorithm in the PBECs. The estimate of the relative processing time is in fact performed by estimating the computational complexity of processing various PBECs. The relative processing time is then used for partitioning and scheduling of the PBECs. The problem is that the estimated size of a PBEC is dependent on the construction of the PBEC (which should not happen). This dependency could be probably removed by using, for example, the bootstrap method. That is: getting the whole e Fs and making bootstrap samples of e Fs that are used for partitioning and estimation of the size of PBECs. Currently, those does not seems to be necessary, as the speedups are quite satisfactory.

11. FUTURE WORK

In future we have to implement the parallel algorithm to reduce the complexity of computational time and implement the result of frequent sequence mining using static load balancing. Additionally we have to reduce the slowness and memory consumption of a process.

REFERENCES

[1] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. Advances in Database TechnologyEDBT'96, pages 1–17, 1996.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In Proceedings of 20th International Conference on Very Large Data Bases, pages 487–499. Morgan Kaufmann, 1994.

[3] R. Kessl. Static load balancing of parallel mining of frequent itemsets using reservoir sampling. In LNCS 6871, Machine Learning and Data Mining in Pattern Recognition - 7th International Conference, pages 553–567. Springer, 2011.

[4] R. Kessl and P. Tvrd´ık. Probabilistic load balancing method for parallel mining of all frequent itemsets. In PDCS '06: Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems, pages 578–586, Anaheim, CA, USA, 2006. ACTA Press.

[5] R. Kessl and P. Tvrd´ık. Toward more parallel frequent itemset mining algorithms. In PDCS '07: Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, pages 97–103, Anaheim, CA, USA, 2007. ACTA Press.

[6] R. Agrawal and R. Srikant. Mining sequential patterns. In Data Engineering, 1995. Proceedings of the Eleventh

ISSN (ONLINE): 2454-9762
ISSN (PRINT): 2454-9762
Available online at www.ijarmate.com
International Journal of Advanced Research in Management, Architecture, Technology and Engineering (IJARMATE)
Vol. 2, Issue 9, September 2016.

International Conference on, pages 3–14. IEEE, 1995.

[7] M. J. Zaki. Parallel sequence mining on shared-memory machines. Journal of Parallel and Distributed Computing, 61(3):401– 426, 2001.

[8] D. Gunopulos, R. Khardon, and R. S. Sharma. Discovering all most specific sentences. ACM Transactions on Database Systems, 28:140–174, 2003.

[9] D. Gunopulos, H. Mannila, and S. Saluja. Discovering all most specific sentences by randomized algorithms. In ICDT, pages 215–229, 1997.

[10] S. Cong, J. Han, J. Hoeflinger, and D. Padua. A samplingbased framework for parallel data mining. In Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 255–265. ACM, 2005.