

Distributed Frequent Itemsets Mining in Heterogeneous Platforms

Lamine M. Aouad, Nhien-An Le-Khac and Tahar M. Kechadi
School of Computer Science and Informatics
University College Dublin
{lamine.aouad,an.le-khac,tahar.kechadi}@ucd.ie

Abstract. Huge amounts of datasets with different sizes are naturally distributed over the network. In this paper we propose a distributed algorithm for frequent itemsets generation on heterogeneous clusters and grid environments. In addition to the disparity in the performance and the workload capacity in these environments, other constraints are related to the datasets distribution and their nature, and the middleware structure and overheads. The proposed approach uses a dynamic workload management through a block-based partitioning, and takes into account inherent characteristics of the Apriori algorithm related to the candidate sets generation. The proposed technique greatly enhances the performance and achieves high scalability compared to the existing distributed Apriori-based approaches. This approach is evaluated on large scale datasets distributed over a heterogeneous cluster.

Keywords: Frequent itemsets generation, the Apriori algorithm, Distributed computing, Dynamic workload management.

1 Introduction

The data mining field is devoted to the challenge of extracting hidden and useful information from large datasets. Over the last few years, advances in many areas of science, engineering, business, etc. have created huge collections of datasets which are constantly increasing. The ability to process, analyze, and understand these datasets is at the boundaries of several disciplines, including parallel and distributed computing. This is due to the size of the datasets, the heterogeneity and the quality of their content, their inherent distributed nature, etc.

Mining frequent itemsets is an important task in this field. This usually consists of finding large itemsets according to a given support threshold. Since its inception, many efficient frequent itemsets mining algorithms have been proposed in the literature [Agrawal and Srikant 1994], [Park et al. 1995], [Savasere et al. 1995], [Han et al. 2000], etc. For distributed datasets, there have been many efforts devoted to the development of parallel and distributed implementations of frequent itemsets generation such as [Agrawal and Shafer 1996], [Ashrafi et al. 2004], [Cheung et al. 1996], and [Schuster and Wolff 2001]. Most of these implementations assume that the target platform is homogeneous and therefore the datasets are partitioned evenly among the system nodes. Indeed, they usually require multiple synchronizations and communication steps. However, in practice, both the datasets and the processing platforms are more likely to be heterogeneous running multiple and different systems and tools. This leads to unbalanced datasets distributions and workloads.

Since the generation task of a distributed implementation of frequent itemsets mining is very computationally expensive, any unbalanced workload and multiple synchronization constraints will significantly degrade the performance. In this paper, we propose a new efficient distributed approach to deal with the heterogeneity and data distributions. This technique is based on an asynchronous dynamic

workload management and a block-based partitioning to deal with memory constraints in each processing node. Also, the nature of the local generation task, using the Apriori manner, shows that the performance is not related to global pruning strategies since the generated candidates sets are very close in both our approach and classical distribution schemes. This means that the communication and synchronization steps in classical distribution schemes are not efficient for local computations [Purdom et al. 2004].

The paper is organized as follows, the next section surveys some related research work in frequent itemsets generation. In section 3, we define the problem and describe our approach and its features. Section 4 presents the computational platform as well as the management and development tools used to test this technique. In Section 5, we explain the conditions of the experiments, show some performance results, and highlight directions for future work. Finally, we conclude our work in Section 6.

2 Background

Frequent itemsets mining plays an important role in a range of applications including association rules, correlations, causality, episodes, etc. Many research works have addressed this area over the last few years. An interesting algorithm, namely the Apriori [Agrawal and Srikant 1994], has emerged as one of the most popular algorithms. It exploits the observation that all subsets of frequent itemsets must be frequent. It finds iteratively all frequent l -itemsets using the frequent $(l - 1)$ -itemsets discovered previously. Many variants of this algorithm have been developed leading to improvements, such as DHP [Park et al. 1995], DIC [Brin et al. 1997], Partition [Savasere et al. 1995], among others, in a sequential manner, and CD [Agrawal and Shafer 1996], FDM [Cheung et al. 1996], [Cheung et al. 1996], ODAM [Ashrafi et al. 2004], and DDM [Schuster and Wolff 2001], in parallel and distributed manners.

Some distributed approaches are based on different sequential algorithms, such as the FP-Growth algorithm [Pramudiono and Kitsuregawa 2003], or the D-Sampling algorithm, which basically combines the Sampling algorithm and the DDM approach [Toivonen 1996], [Schuster et al. 2003]. Furthermore, at the best of our knowledge, only one well-adapted algorithm on distributed systems such as the grid has been proposed. This approach, called GridDDM [Chung and Luo 2004], mines maximal frequent itemsets from distributed datasets. It is based on a sequential maximal miner in each node followed by a global mining phase. There exists several papers summarizing the work that has been done in this area. We can cite two overviews by Zaki [Zaki 1999] and [Zaki 2000], which are fairly old but still up to date since only few studies have been carried out in this area in the last few years. A relatively more recent overview is presented in [Park and Kargupta 2002]. Although the authors consider a general survey, many aspects apply to the frequent itemsets mining.

Most of the existing parallel/distributed approaches need multiple synchronization and communication steps. Ranging from the CD (Count Distribution) approach which is a direct distribution of the Apriori algorithm, to the enhanced versions like FDM (Fast Distributed Mining of association rules), ODAM (Optimized Distributed Association Rule Mining Algorithm), or the DDM approach and its variants [Schuster and Wolff 2001], which introduce some advanced techniques for pruning, communications, or optimization in the implementation itself, multiple support counts exchange steps are still required. These steps can generate excessive communications and input/output overheads in widely distributed and heterogeneous platforms.

Unlike these approaches, we propose a well-adapted technique for heterogeneous distributed platforms based on the Apriori algorithm. Indeed, the heterogeneity expressed in term of datasets distributions or the memory size, for instance, may greatly degrade the performance. The implementation has then to take into account out-of-core datasets and computations. Specifically, our approach provides the following contributions:

- Characterizing the performance of the generation task, namely the Apriori algorithm, and its distribution schemes.
- Developing optimization schemes for memory and workload management.
- Finally, experimentally evaluating the effectiveness of our distributed approach on synthetic and real datasets with various execution parameters and distributions ratios.

3 Approach description

In this section, we define the frequent itemsets generation problem, describe our algorithm, and give an analytical discussion that describes the proposed approach.

3.1 Problem definition

Before describing the algorithm, we give a formal description of the problem. The frequent itemsets generation problem can be described as follows. Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of n items, and let $W = \{t_1, t_2, \dots, t_m\}$ be a set of m transactions, where each transaction t_i is a subset of I . An itemset $i \in I$ of size k is called k -itemset, and the support of i is $\sum_{j=1}^m \delta_{ij}$ ($\delta_{ij} = 1$ if $i \in t_j$, 0 otherwise), i.e. the number of transactions in W that have i as a subset. The frequent itemsets mining is to find all $i \in W$ that have a support greater than a minimum support threshold, which is usually defined by the user. We denote by F_k the frequent k -itemsets, and by MF_i the frequent itemsets of size i that are not subsets of any other frequent itemset, i.e. maximal frequent itemsets.

For the distribution of the datasets among the processing nodes, we consider it at two levels of granularity; the dataset partitioning and the memory partitioning. In the first level, the dataset W with D transactions is divided among M nodes; $\{T_1, T_2, \dots, T_M\}$. Note that a transaction is the smallest unit that can be assigned to a node during the partitioning. In other words, we do not allow a transaction to be split between the nodes. Let the size of the partition T_i be D_i . In the second level of partitioning, each partition T_i is divided into b_i blocks $\{\delta_1, \delta_2, \dots, \delta_{b_i}\}$. The size of a block δ_i is defined according to the available memory in the corresponding processing node N_i and information about the dataset such as the number of items, the average transaction width, and also the support threshold. For a given minimum support threshold s_W , an itemset x is *globally frequent* if it is frequent in W ; its support $x.sup$ is greater than $s_W \times D$, and is *locally frequent* in a node N_i if it is frequent in T_i ; its support $x.sup_i$ is greater than $s_W \times D_i$ (or $s_W \times (D_i/b_i)$ per block).

—*Property 1.* A globally frequent itemset must be locally frequent in at least one node.

Proof. Let x be an itemset. If its support $x.sup_i$ is smaller than $s_W \times D_i$ for $i = 1, \dots, M$, then its support $x.sup$ is smaller than $s_W \times D$ (since $x.sup = \sum_{i=1}^M x.sup_i$ and $D = \sum_{i=1}^M D_i$), and x cannot be globally frequent. Then, if x is globally frequent, it must be locally frequent in at least one node N_i .

—*Property 2.* All subsets of a globally frequent itemset are globally frequent.

Proof. Let x be an itemset, and let x' be a subset of x . If $x'.sup$ is smaller than $s_W \times D$, then $x.sup$ is also smaller than $s_W \times D$ (since $x.sup \leq x'.sup$), and x cannot be globally frequent. Then, if x is globally frequent, all its subsets must be frequent.

3.2 The algorithm

Basically, the algorithm consists of two phases. The first phase consists of the Apriori generation which can be seen as a set of super-steps. Each super-step consists of a calculation on the local dataset blocks, or a received block, and the potential communication exchanges for the workload and requests management. The second phase consists of a single synchronization barrier which is carried out at the end of the first phase for global results aggregation. We will give a formal description related to these phases in the next section. In the beginning of the first phase, each node N_i broadcast the size of its dataset portion D_i , its number of blocks b_i , and a limited workload description vector WV_i , including system information about the size of the available memory, and the rate of CPU. This information is used to estimate a remote performance time when a given node receives a job request from another processing node.

The local generation of the candidate sets in the first phase is done in the Apriori manner, and only local pruning is considered to generate the locally frequent k -itemsets, i.e. there are no intermediate support counts exchange steps. In addition, all the smaller locally frequent itemsets ($< k$) that are not subsets of any of the locally frequent k -itemset are considered for support counts collection in the first pass of the second phase. In order to generate all existing globally frequent itemsets of sizes 1 to k , few passes can be required. This greatly reduces the communication costs and avoid the multiple synchronization phases usually required in classical distributed algorithms. This is based on the fact that global pruning strategies used in classical distributed implementations do not bring enough useful information compared to the generated overheads due to synchronizations, communications, and input/output operations (I/O) [Aouad et al. 2007]. The detailed approach is given in the Algorithm 1.

As mentioned before, the computation complexity of the Apriori generation depends on several factors such as the number of items in the dataset, the support threshold, etc. Based on this information, the size of blocks is chosen according to the effective main memory available in the corresponding processing node. This can be based on previous executions. Indeed, it is well known that the candidate sets can exponentially grow in the Apriori generation process leading to high demand in memory space. This leads either to a thrashing effect, which can drastically degrade the performance, or one may not be able to deal with the dataset if the implementation is not adapted to out-of-core computations. Note that the block partitioning is performed beforehand. Each node performs then the Apriori generation in its blocks independently of the others. If a given node becomes idle (i.e. it finishes its blocks), it selects a donor processor that has not issued its end notification yet. Among overloaded processors, we select a node with a smaller block size, and the largest remote number of blocks. This policy reduces the overhead of re-partitioning larger blocks into smaller ones that can fit the new destinations main memory. For this purpose, each node N_i constructs a table with the received information and maintains it through the *ThreadUpdate* which collects information about blocks moves between nodes.

Furthermore, due to the platform heterogeneity, a naive application of this approach can lead to inefficient data movement. Indeed, if a given node sends a work request to another node just after it has started its computation, it can take a significant amount of time before the latter treats that request. In our approach, nodes can respond to a request during their counting phase. The information state table and the *ThreadCheckRequest* will allow to process the request quickly and immediately after its reception.

At the end of the first phase, every node asks for remote support counts of its locally frequent item-

Algorithm 1. Distributed frequent itemsets mining using workload management.

Input : $T_i, i = 1, \dots, M$, partitioned into b_i blocks, s_W and k

Output: L , globally large itemsets $\leq k$

```

for  $i = 1$  to  $M$  do
  | Broadcast ( $D_i, b_i, WV_i$ );
end
// first phase: block-based Apriori gen. / workload management
for  $i = 1$  to  $M$  do
  | ThreadCheckRequest ();
  | ThreadUpdate ();
  while nbBlocks > 0 do
    | AprioriGen ( $T_{i,j}, k$ );
  end
  BroadcastNotification ();
  while !allNotifications do
    | SendRequest ();
    if GetResponse ( $m$ ) then
      | ReceiveBlock ( $m$ );
      | AprioriGen ( $T_m, k$ );
    end
  end
end
// second phase: remote support counts computation
for  $i = 1$  to  $M$  do
  while  $LL_i \neq \emptyset$  do
    for  $j = 1$  to  $M$  ( $i \neq j$ ) do
      | ReceiveRemoteSet ( $i, LL_j$ );
      | LocalSupport ( $LL_j$ );
      | SendSupportCounts ( $i, j$ );
    end
    for  $j = 1$  to  $M$  ( $i \neq j$ ) do
      | ReceiveRemoteSupport ( $j$ );
    end
     $L_i +=$  ComputeGloballyLarge ();
     $LL_i =$  SubsetsGloballyFailed ();
  end
end
 $L = \bigcup_{i=1}^M L_i$ 

```

sets. Recall that we consider, for the first collection pass, the locally frequent k -itemsets and locally frequent itemsets with smaller sizes that are not subsets of any bigger set. These sets are denoted by LL_i on N_i . Figure 1 shows a generation example at a given node. The circles indicates the first considered sets for support counts collection from the other nodes ($k = 4$ in this case). This first pass will usually generate almost all frequent itemsets, depending on the considered size k . Otherwise, few collection passes for subsets of globally infrequent itemsets already tested, denoted by $SF_{\{<k\}}$, are performed. Thus, each node receives these sets from the other nodes, computes their local support counts, and sends them back to their corresponding nodes. Each node deduces the globally frequent itemsets, denoted by L_i , according to the global support threshold. Subsets of $L = \bigcup_{i=1}^M L_i$ repre-

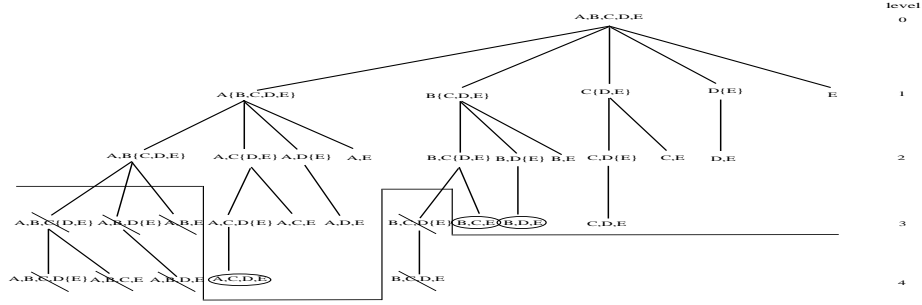


Fig. 1. Local generation example. Circles indicate the considered itemsets for global support counts collection at the first pass. Infrequent sets are crossed out.

sent then all globally frequent itemsets of size 1 to k . Note that during this phase, the supports are not broadcasted, instead each node N_i is responsible for collecting supports of its set LL_i . This eliminates the exchange of unnecessary support counts between processes.

—*Property 3.* Subsets of $L = \bigcup_{i=1}^M L_i$, where L_i is the set of globally frequent itemsets generated from locally frequent itemsets $LL_i = (\bigcup_{j=1}^{k-1} MF_j) \cup F_k \cup SF_{\{<k\}}$, represent the set of all globally frequent itemsets of sizes ranging from 1 to k .

Proof. The set L contains all the globally frequent itemsets of size k , all smaller sizes that are not subsets of them, and all subsets of higher size globally infrequent itemsets. Thus, and according to the property 1, the subsets of these itemsets represent all the globally frequent itemsets of size 1 to k .

This approach presents then a well-adapted scheme for large distributed platforms in addition to a dynamic task formulation that monitors the workload imbalance and moves work through the network as needed. The workload balancing structure is based on the following: a work request occurs at the end of the first super-step (described before) for a given node. At that level, the workload has to be re-balanced asynchronously. Indeed, considering the case in which a computation is particularly unbalanced, this will only be determined after the counting phase has been completed for a given block and nothing can be done to correct it. A synchronous scheme will then be less efficient in reducing the workload imbalance overhead. Note that the platform heterogeneity causes workload imbalances even if the dataset is distributed evenly.

3.3 Analytical discussion

The main idea of this approach is that the distribution of the Apriori algorithm does not need global pruning strategies since the global candidate sets obtained using a global pruning strategy is usually very close to those obtained by local pruning only. Figure 2 shows plots of generated candidate sets on two processing nodes using various support thresholds for the census dataset (described in the next section). It is shown that the difference between the candidate sets using only local pruning (denoted by GFM), and with a global pruning strategy (FDM), is very small. We also introduced a local block partitioning taking into account the available memory, in addition to a local periodic checking for requests during the counting phases. This means that any process can respond to remote task requests during its counting phase. This block-based workload balancing approach eliminates the waiting time incurred at the end of the first phase and in receiving work. It also allows the additional communication

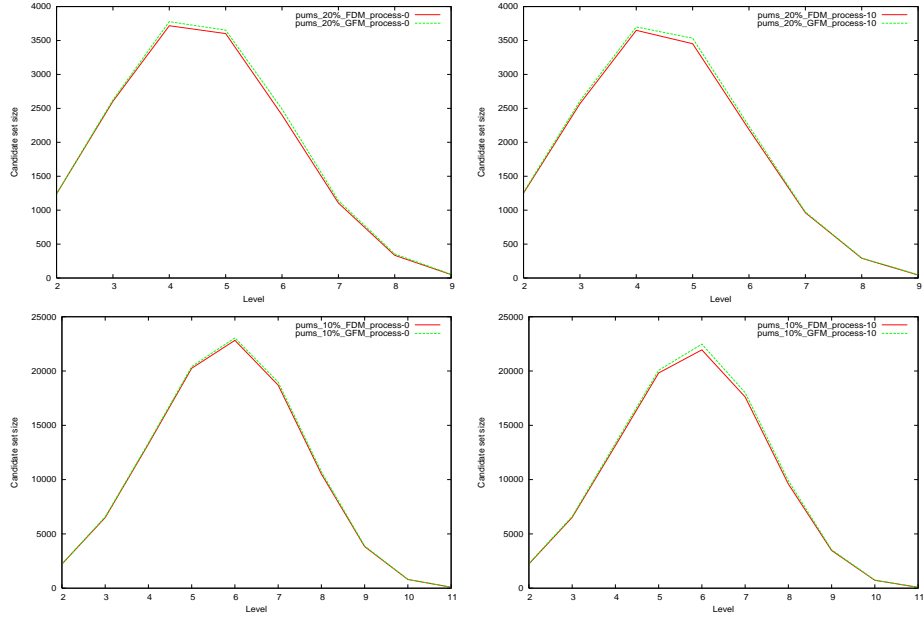


Fig. 2. Candidate sets using a global pruning strategy (FDM) and only local pruning (GFM) on the census dataset.

overheads to be overlapped by computations.

Considering the basic implementation without workload management, the super-step introduced earlier can be parameterized using only the Apriori generation cost p_i by:

$$C_{basic} = b_i p_i$$

which actually corresponds to the cost of the first phase. The parameter p_i can be written as the sum of the generation at each level as:

$$p_i = \sum_{l=1}^k p_{i,l}$$

According to our experiments, $p_{i,l}$ are quite similar in this implementation compared to the FDM approach since the candidate sets are very close in both cases (Fig. 2) [Aouad et al. 2007]. Using the FDM approach, the generation cost for the first phase is:

$$C_{FDM} = b_i \sum_{l=1}^k (p_{i,l} + (M-1) c_{i,l})$$

where $c_{i,l}$ is the locally frequent itemsets at the level l , and $(M-1) c_{i,l}$ the communication cost at the same level. This approach introduces one step of communications per iteration. Note that synchronization is implicit in each communication step.

Consider now the version using the workload management, and the additional parameter bc_i corresponding to the communication cost of a local block at a given site N_i . The maximum cost of the first

phase of our approach is given by:

$$CP_{max} = (\arg \max_{i \in N_i} bw_i p_i) + bc_i + p_j$$

where bw_i is the number of blocks at the site N_i which is still at its first super-step, and p_j corresponds to the remote computation cost on the site N_j of the potentially received last block from N_i .

—*Lemma.* $CP_{max} \in [C_{FDM} - \sum_{l=1}^{k-1} \arg \max_{i \in N_i} (M-1) c_{i,l}, C_{FDM}]$.

Proof. In the worst case, all the local blocks at a given site are executed locally. This means that CP_{max} is smaller or equal to $\arg \max_{i \in N_i} b_i p_i$. Now, consider the global candidates set $GC_{i,l}$ obtained using global pruning strategies in the FDM approach, and the set $LC_{i,l}$, the local candidates set obtained only by local prunings in our implementation. Under a wide range of conditions and datasets, the set $GC_{i,l}$ is very close to the set $LC_{i,l}$, for $i = 1, \dots, M$ and $l = 1, \dots, k$. This means that the sum $\sum_{l=1}^k p_{i,l}$ is quite similar in both cases. According to this and considering the same execution conditions, $C_{FDM} - b_i p_i = C_{FDM} - b_i \sum_{l=1}^k p_{i,l}$ is bounded by $\sum_{l=1}^{k-1} \arg \max_{i \in N_i} (M-1) c_{i,l}$. Therefore, in the case of a single communication step, the difference $CP_{max} - C_{FDM}$ can reach $\sum_{l=1}^{k-1} \arg \max_{i \in N_i} (M-1) c_{i,l}$.

4 Experimentation tool and platform

This section presents the experimental tool, its workflow manager, and the experimentations setup. Our implementation uses pre-defined scenarios for data distribution since the middleware does not handle data/task locality, i.e. there is no direct communication between ongoing processes and no explicit data/task management.

4.1 Condor/DAGMan

The Condor system is a distributed batch system providing a job management mechanism, resource monitoring and management, scheduling and priority schemes [Thain et al. 2004]. The Condor system provides a ClassAds mechanism for matching resource requests and offers checkpointing and migration mechanisms. It also provides job management capabilities for the grid through Condor-G (using the Globus Toolkit) and Condor-C, which allows jobs migration between machines job queues.

DAGMan is a directed acyclic graph representation manager, which allows the user to express dependencies between Condor jobs. It allows the user to list the jobs to be done with constraints on the order through several description files for the DAG and the jobs within the task graph. It also provides fault-tolerant capabilities allowing to resume a workflow where it was left off, in the case of a crash or other failures. However, the scripting language required by DAGMan is not flexible since every job in the DAG has to have its own condor submit description file.

4.2 Experimentation platform

The platform used to perform our experiments is a cluster of heterogeneous workstations connected by a Fast Ethernet network. It consists of 4 Pentium III nodes (1GHz and 512MB), 1 Pentium III node (733MHz and 384MB), 1 Pentium III node (500MHz and 192MB), and 2 bi-processors Pentium IV nodes (3.40GHz and 1GB memory).

5 Tests

For these tests, a synthetic dataset and a census dataset (the PUMS dataset available from the UC Irvine KDD Archive) are used. The datasets size is 0.5×10^6 transactions, with average transaction size of 10 for the synthetic dataset, and 30 for the census dataset. Different random distributions were

Table I. Average execution times for the synthetic dataset.

Support	ratio	FDM	GFM-W	Factor
0.2%	1:1	385	219	43%
0.1%	1:1	1276	757	40%
0.2%	1:5	293	141	51%
0.1%	1:5	1048	617	41%
0.2%	1:10	355	164	53%
0.1%	1:10	2395	846	64%

generated with various ratios; up to 1:10¹. We define the memory imbalance factor as the ratio between the smallest memory size and the biggest memory size available in the platform. In our experiments, the memory imbalance factor is 2.5 (since the bi-processors share the 1gig memory). This leads to different block sizes, ranging from 6000 to 20000 transactions for both datasets. We consider the generation of the globally frequent itemsets up to $k = 4$ for the synthetic dataset, and $k = 10$ for the census dataset.

For comparison purposes, we implemented a classical distributed Apriori-based algorithm, namely the FDM approach [Cheung et al. 1996], [Cheung et al. 1996]. In this approach, the frequent itemsets generation of size k consists of k synchronization steps for support counts requests, $k + 1$ parallel activities for the Apriori generation process, and k parallel activities for remote support counts computation. In our implementation, the FDM technique runs on all available local blocks, in a sequential manner, before each support counts exchange step.

We run both techniques on both datasets under the same conditions. Also, our tests scenarios use real measurements for task performance prediction from previous executions. The average processing times for different datasets distributions and various support thresholds are given in Tables I and II. We can see that our technique outperforms the FDM approach and reduces the computing times by a factor up to 86%. The factor is more important in the case of the census dataset since we generate larger frequent itemsets sizes. This leads to additional communication and I/O overheads for the FDM technique. Our implementation eliminates all intermediate synchronization steps (mainly reduced from k to 1). Also, the remote support computations are very computationally expensive and generate important I/O overheads in the FDM approach. Consequently, this greatly degrades its performance, specially in the case of large systems and larger frequent itemsets sizes.

The tests show that the gain is important since synchronization and communication steps costs are far more important than the increase in the overall computation cost using only local pruning in our approach. Furthermore, it shows that the dynamic workload management is efficient in the case of data distribution imbalance and platform heterogeneity. The additional communications are overlapped by local computations. Indeed, in our approach, the adopted dynamic policy for job requests checking and data movement, that lead to the additional communication overheads, are overlapped with the computations, i.e. counting steps on blocks.

¹Ratio 1:x means the difference factor between local sizes ranges from 1 to x.

5.1 Discussion

The heterogeneity and its related issues make it difficult to design efficient distributed algorithms. The presented approach tackled some of these constraints. It is based on the following objectives: (1) synchronization and communication overheads are reduced to the minimum, (2) a block-based approach is introduced to deal with local memory constraints, and (3) a dynamic workload management, overlapped by computations, is introduced to deal with the platform heterogeneity and imbalanced workloads. In addition to the dataset properties, the gain factor is related to the dataset distribution and therefore the number of local blocks, as well as the largest generated size of frequent itemsets.

The proposed approach is more scalable since in some executions the data structure related to the large amount of remote support counts exchanges, for the FDM approach, do not fit in main memory, and may need explicit I/O management for the local intermediate itemsets support counting. The performance of the FDM approach for these imbalanced workloads on highly heterogeneous nodes are badly affected since it is given by the slowest and/or the most overloaded node.

Table II. Average execution times for the census dataset.

Support	ratio	FDM	GFM-W	Factor
15%	1:1	3289	1158	65%
20%	1:1	1443	252	82%
15%	1:5	3588	1351	62%
20%	1:5	2104	294	86%
15%	1:10	2824	1196	58%
20%	1:10	1536	260	83%

Another important point for distributed implementations, especially in clusters and grid environments, is the execution overhead issue. Indeed, latencies related to the management and programming tools, such as for the job preparation and submission, or task scheduling, are of prime importance and can be the first cause of efficiency loss. However, we did not consider them in our results. This means that our evaluation takes into account the processing time of each task, at each stage, including only their communication phases. Future evaluations will consider details about the cost of the whole hierarchy of the overheads for both versions. Other implementations using different middleware and workflow management engines will also be considered. Currently, ongoing tests using the Globus Toolkit [Foster 2006] and the Commodity Grid kit [von Laszewski et al. 2001], in a nation-wide grid platform [Cappello and all 2005], show that the execution overheads are much smaller than in the Condor/DAGMan system, reaching roughly 50% less overhead, excluding the task scheduling cost which was explicitly managed in those tests.

6 Conclusion

We presented a distributed frequent itemsets generation approach using dynamic workload management through a block-based partitioning. The block-based approach deals with memory constraints since the basic task may need very large memory space depending on several parameters including the support threshold, and information about the dataset. Our approach also exploits an inherent property of the itemsets generation task that shows that the intermediate communication steps, in classical implementations such as the FDM approach, are performance constraining. Indeed, global pruning

strategies do not bring enough useful information in comparison to the generated synchronization and I/O overheads. Furthermore, the workload management strategy tackles the imbalanced workloads of the platform heterogeneity and/or uneven dataset distribution. Experiments have been conducted on a heterogeneous testbed and show that the proposed algorithm achieves very good performance and high scalability compared to a classical Apriori-based implementation.

References

- AGRAWAL, R. AND SHAFER, J. C. 1996. Parallel mining of association rules. *IEEE Trans. On Knowledge And Data Engineering* 8, 962–969.
- AGRAWAL, R. AND SRIKANT, R. 1994. Fast Algorithms for Mining Association Rules. In J. B. BOCCA, M. JARKE, AND C. ZANIOLO (Eds.), *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pp. 487–499. Morgan Kaufmann.
- AOUAD, L. M., LE-KHAC, N.-A., AND KECHADI, T. M. 2007. Performance study of distributed apriori-like frequent itemsets mining. Technical report, University College Dublin.
- ASHRAFI, M. Z., TANIAR, D., AND SMITH, K. A. 2004. ODAM: An Optimized Distributed Association Rule Mining Algorithm. *IEEE Distributed Systems Online* 5, 3.
- BRIN, S., MOTWANI, R., ULLMAN, J. D., AND TSUR, S. 1997. Dynamic itemset counting and implication rules for market basket data. In J. PECKHAM (Ed.), *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pp. 255–264. ACM Press.
- CAPPELLO, F. AND ALL 2005. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *GRID 2005, 6th IEEE/ACM international workshop on Grid Computing. SuperComputing*.
- 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: International Conference on Parallel and Distributed Information Systems*. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD.
- CHEUNG, D. W., NG, V. T., FU, A. W., AND FU, Y. 1996. Efficient mining of association rules in distributed databases. *IEEE Trans. Knowledge and Data Eng.* 8, 6, 911 – 922.
- CHUNG, S. M. AND LUO, C. 2004. Distributed mining of maximal frequent itemsets from databases on a cluster of workstations. In *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, pp. 499–507. IEEE Computer Society.
- FOSTER, I. 2006. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology* 21, 4.
- HAN, J., PEI, J., AND YIN, Y. 2000. Mining frequent patterns without candidate generation. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp. 1–12. ACM Press.
- PARK, B. AND KARGUPTA, H. 2002. Distributed data mining: Algorithms, systems, and applications. In *Data Mining Handbook*.
- PARK, J. S., CHEN, M.-S., AND YU, P. S. 1995. An effective hash-based algorithm for mining association rules. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp. 175–186. ACM Press.
- PRAMUDIONO, I. AND KITSUREGAWA, M. 2003. Parallel FP-Growth on PC Cluster. In *Proceedings of the Seventh Pacific-Asia Conference of Knowledge Discovery and Data Mining (PAKDD03)*, Seoul, Korea, pp. 467–473.

- PURDOM, P. W., GUCHT, D. V., AND GROTH, D. P. 2004. Average-case performance of the apriori algorithm. *SIAM Journal on Computing* 33, 5, 1223–1260.
- SAVASERE, A., OMIECINSKI, E., AND NAVATHE, S. B. 1995. An efficient algorithm for mining association rules in large databases. In *The VLDB Journal*, pp. 432–444.
- SCHUSTER, A. AND WOLFF, R. 2001. Communication efficient distributed mining of association rules. In *ACM SIGMOD, Boston*.
- SCHUSTER, A., WOLFF, R., AND TROCK, D. 2003. A High-Performance Distributed Algorithm for Mining Association Rules. In *Third IEEE International Conference on Data Mining*, Florida, USA.
- THAIN, D., TANNENBAUM, T., AND LIVNY, M. 2004. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*.
- TOIVONEN, H. 1996. Sampling large databases for association rules. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, San Francisco, CA, USA, pp. 134–145. Morgan Kaufmann Publishers Inc.
- VON LASZEWSKI, G., FOSTER, I., GAWOR, J., AND LANE, P. 2001. A Java commodity grid kit. *Concurrency and Computation: Practice and Experience* 13, 8-9, 645–662.
- ZAKI, M. 1999. Parallel and Distributed Association Mining: A Survey. *IEEE Concurrency*.
- ZAKI, M. 2000. Parallel and distributed data mining: An introduction. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, London, UK, pp. 1–23. Springer-Verlag.