Executing Association Rule Mining Algorithms under a Grid Computing Environment

Raja Tlili
Faculty of Sciences of Tunisia
Department of Computer Science
Campus Universitaire Tunis, El- Manar, 2092 Tunis
00216 71 872 600
Raja.Tlili@fst.rnu.tn

Yahya Slimani
Faculty of Sciences of Tunisia
Department of Computer Science
Campus Universitaire Tunis, El- Manar, 2092 Tunis
00216 71 872 020
Yahya.Slimani@fst.rnu.tn

ABSTRACT
Grids are now regarded as promising platforms for data and computation-intensive applications like data mining. However, the exploration of such large-scale computing resources necessitates the development of new distributed algorithms. The major challenge facing the developers of distributed data mining algorithms is how to adjust the load imbalance that occurs during execution. This load imbalance is due to the dynamic nature of data mining algorithms (i.e. we cannot predict the load before execution) and the heterogeneity of Grid computing systems. In this paper, we propose a dynamic load balancing strategy for distributed association rule mining algorithms under a Grid computing environment. We evaluate the performance of the proposed strategy by the use of Grid'5000. A Grid infrastructure distributed in nine sites around France, for research in large-scale parallel and distributed systems.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *Data mining*.

I.6.4 **[Simulation and Modeling]:** Model Validation and Analysis.

D.1.3 [**Software**]: Concurrent Programming – *Distributed programming*.

F.1.2 [**Theory of Computation**]: Modes of Computation – *Parallelism and concurrency.*

General Terms
Algorithms, Design, Experimentation, Performance.

Keywords
Association rules, Dynamic load balancing, Grid Computing, Parallel association rule mining, Work migration, Apriori algorithm.

PADTAD'11, July 17, 2011, Toronto, ON, Canada

1. INTRODUCTION
Powerful tools are needed to extract knowledge from the growing amount of data being collected and stored. Association Rule mining technique which tends to find interesting correlation relationships between items in voluminous transactional databases has become one of the main data mining techniques [3].

Classic data mining techniques, based on sequential approaches, are often inadequate due to the complexity of the analysis that need to be performed on a huge amount of data. Sequential approaches cannot provide the scalability, in terms of the data dimensionality, size, and runtime performance. Moreover, the increasing trend towards decentralized business organizations, distribution of users, software, and hardware systems magnifies the need for more advanced and flexible approaches and solutions.

Parallel association rule mining is designed for tightly-coupled systems, like shared or distributed memory machines, and clusters based on fast networks. Distributed association rule mining deals with loosely-coupled systems: clusters with average-fast or slow networks and geographically distributed computing nodes. Grid association rule mining supports the distribution of data and allows the use of geographically dispersed computing resources (at a lower cost) in order to achieve performances not ordinarily attainable on a single computational resource.

Although Grid computing systems shares many commonalities with parallel and distributed approaches but there are platform peculiarities and requirements implying extra efforts and new methodologies to deal with the heterogeneity of such systems.

The majority of existing parallel and distributed ARM algorithms were designed for a homogeneous dedicated environment and thus use static load balancing strategies. By running these algorithms under Grid systems their performance degrades due to the load imbalance that appears between resources during execution time. This load imbalance is caused by the dynamic nature of the association rule mining algorithm and also by the heterogeneity of such computing systems. Because of that we need to develop new load balancing schemes that would allow a better exploration of this advanced platform.

In this paper, we develop and evaluate a run time load balancing strategy for mining association rule algorithms under a grid computing environment. The goal of our strategy is to improve the performance of distributed algorithms and ameliorate their response time. The rest of the paper is organized as follows: Section 2 introduces association rule mining technique. Section 3 describes the load balancing problem. Section 4 presents the

system model of a Grid. In section 5, we propose the dynamic load balancing strategy. Experimental results obtained from implementing this strategy are shown in section 6. Finally, the paper concludes with section 7.

2. ASSOCIATION RULE MINING TECHNIQUE

Association rules mining (ARM) finds interesting correlation relationships among a large set of data items. This technique could be applied to a diversity of domains and the knowledge gained can be used in applications ranging from business management, production control, and market analysis, to engineering design and science exploration.

A typical example of this technique is market basket analysis. This process analyses customer buying habits by finding associations between different items that customers place in their "shopping baskets". Such information may be used to plan marketing or advertising strategies, as well as catalog design [3]. Each basket represents a different transaction in the transactional database, associated to this transaction the items bought by a customer. Given a transactional database D, an association rule has the form $A=>B$, where A and B are two itemsets, and $A \cap B = \varnothing$. The rule's support is the joint probability of a transaction containing both A and B at the same time, and is given as $\sigma(AUB)$. The confidence of the rule is the conditional probability that a transaction contains B given that it contains A and is given as $\sigma(AUB)/\sigma(A)$. A rule is frequent if its support is greater than or equal to a pre-determined minimum support and strong if the confidence is more than or equal to a user specified minimum confidence [3].

Association rule mining is a two-step process:

1. The first step consists of finding all frequent itemsets that occur at least as frequently as the fixed minimum support;
2. The second step consists of generating strong implication rules from these frequent itemsets.

The overall performance of mining association rules is determined by the first step which is known as the frequent set counting problem [3].

2.1 Sequential Association Rule Mining Algorithms

Many sequential algorithms for solving the frequent set counting problem have been proposed in the literature. We can define two main methods for determining frequent itemsets supports: with candidate itemsets generation [11, 13] and without candidate itemsets generation [5].

Apriori [11] was the first effective algorithm proposed. This algorithm uses a generate-and-test approach which depends on generating candidate itemsets and testing if they are frequent. It uses an iterative approach known as a level-wise search, where k-itemsets are used to explore $(k+1)$-itemsets. During the initial pass over the database the support of all 1-itemsets is counted. Frequent 1-itemsets are used to generate all possible candidate 2-itemsets. Then the database is scanned again to obtain the number of occurrences of these candidates, and the frequent 2-itemsets are selected for the next iteration.

DCI algorithm proposed by Orlando and others [13] is also based on candidate itemsets generation. It adopts a hybrid approach to compute itemsets supports, by exploiting a counting-based method (with a horizontal database layout) during its first iterations and an intersection-based technique (with a vertical database layout) when the pruned dataset can fit into the main memory.

FP-growth algorithm [5] allows frequent itemsets discovery without candidate itemsets generation. First it builds from the transactional database a compact data structure called the FP-tree then extracts frequent itemsets directly from the FP-tree.

2.2 Parallel Association Rule Mining Algorithms

Association rule mining algorithms suffer from a high computational complexity which derives from the size of its search space and the high demands of data access. Parallelism is expected to relieve these algorithms from the sequential bottleneck, providing the ability to scale the massive datasets, and improving the response time. However, parallelizing these algorithms is not trivial and is facing many challenges including the workload balancing problem. Many parallel algorithms for solving the frequent set counting problem have been proposed. Most of them use Apriori algorithm [11] as fundamental algorithm, because of its success on the sequential setting. The reader could refer to the survey of Zaki on association rules mining algorithms and relative parallelization schemas [7]. Agrawal et al. proposed a broad taxonomy of parallelization strategies that can be adopted for Apriori in [10].

There also exist many grid data mining projects, like Discovery Net, GridMiner, DMGA [9] which provide mechanisms for integration and deployment of classical algorithms on grid. Also the DisDaMin project that deals with data mining issues (as association rules, clustering, etc.) using distributed computing [15].

3. THE NEED OF LOAD BALANCING: PROBLEM DESCRIPTION

Work load balancing is the assignment of work to processors in a way that maximizes application performance [4]. A typical distributed system will have a number of processors working independently with each other. Each processor possesses an initial load, which represents an amount of work to be performed, and each may have a different processing capacity (i.e. different architecture, operating system, CPU speed, memory size and available disk space). To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all processors in a way that minimizes both processor idle time and inter-processor communication.

Work-load balancing process can be generalized into four basic steps:

1. Monitoring processor load and state;
2. Exchanging workload and state information between processors;
3. Decision making;
4. Data migration. The decision phase is triggered when the load imbalance is detected to calculate optimal data redistribution.

In the fourth and last phase, data migrates from overloaded processors to underloaded ones. According to different policies

used in the previously mentioned phases, Casavant and kuhl [14] classify work-load balancing schemes into three major classes: ***(1)*** Static versus dynamic load balancing; ***(2)*** Centralized versus distributed load balancing ; ***(3)*** Application-level versus system-level load balancing.

3.1 Load Balancing in Parallel Association Rule Mining Algorithms

Static load balancing can be used in applications with constant workloads, as a pre-processor to the computation [4]. Other applications require dynamic load balancers that adjust the decomposition as the computation proceeds [4, 6]. This is due to their nature which is characterized by workloads that are unpredictable and change during execution. Data mining is one of these applications.

Parallel association rule mining algorithms have a dynamic nature because of their dependency on the degree of correlation between itemsets in the transactional database which cannot be predictable before execution. Basically, current algorithms assume the homogeneity and stability of the whole system, and new methodologies are needed to handle the previously mentioned issues.

3.2 Load Balancing in Grid Computing

Although intensive works have been done in load balancing, the different nature of a Grid computing environment from the traditional distributed system, prevent existing static load balancing schemes from benefiting large-scale applications. An excellent survey from Y. Li et al. [17], displays the existing solutions and the new efforts in dynamic load balancing that aim to address the new challenges in Grid.

The work done so far to cope with one or more challenges brought by Grid: heterogeneity, resource sharing, high latency and dynamic system state, can be identified by three categories as mentioned in [17]:

1. Repartition methods focus on calculating a new optimized workload distribution based on the current load distribution and system state. Though many variant repartition algorithms try to make balance between the load balance and the migration cost, they share the drawback that stems from the prior knowledge of the workload and system state, and cannot adapt to the system changes. Meanwhile, most algorithms are based on the homogenous processors that are often not the cases in Grid.
2. Divisible load theory based schemes arbitrarily partition the workload of the application and distribute it to multiple processors or machines in a linear way, giving a tractable methodology in load distribution modeling both computation and communication. This method takes full consideration of the communication latency in Grid and makes an optimized load distribution decision to achieve the goal of application-level load balancing. However, it suffers from the inability to dynamic changes of the processor utilization rate and fluctuations of network channel, which limited this approach to static load balancing area.
3. Prediction based schemes require accurate estimate of the future computation time and communication cost to establish the performance evaluation model. The experiments set shows that these schemes works well for short-time task but need further investigation in case of long-term applications.

4. GRID MODEL

In our study we model a Grid as a collection of T sites with different computational facilities and storage subsystem. Let $G=(S_1, S_{2,\ldots,} S_T)$ denotes a set of sites, where each site S_i is defined as a vector with three parameters $S_i = (M_i , Coord(S_i) , L_i)$, where M_i is the total number of clusters in S_i, $Coord(S_i)$ is the workload manager, named the coordinator of S_i, which is responsible of detecting the workload imbalance and the transfer of the appropriate amount of work from an overloaded cluster to another lightly loaded cluster within the same site (intra-site) or if it is necessary to another remote site (inter-sites). This transfer takes into account the transmission speed between clusters which is denoted $\zeta_{ijj'}$ (if the transmission is from cluster cl_{ij} to cluster $cl_{ij'}$). If work migration should be performed between sites, then the destination site will be fixed through communication between the coordinators of different sites. This communication is done in a unidirectional ring topology via a token passing mechanism. And L_i is the computational load of S_i.

Each cluster is characterized by a vector of four parameters $cl_{ij}=(N_{ij} , Coord(cl_{ij}) , L_{ij}, \omega_{ij})$, where N_{ij} is the total number of nodes in cl_{ij} , $Coord(cl_{ij})$ is the coordinator node of cl_{ij} which ensures a dynamic smart distribution of candidates to its own nodes, L_{ij} is the computational load of cluster cl_{ij} and ω_{ij} is its processing time which is the mean of processing times of cluster's nodes. In fact each node nd_{ijk} has its processing time denoted ω_{ijk} . Thus

$$\omega_{ij} = \text{Average}(\omega_{ijk}) = (\sum_{k} \omega_{ijk}) / N_{ij} \qquad (1)$$

Figure 1 shows the Grid system model. To avoid keeping global state information in a large-scale system (where this information would be very huge), the proposed load balancing model is distributed in both intra-site and inter-sites. Each site in the Grid has a workload manager, called the coordinator, which accommodates submitted transactional database partitions and the list of candidates of the previous iteration of the association rules mining algorithm. Each coordinator aims at tracking the global workload status by periodically exchanging a "state vector" with other coordinators in the system. Depending on the workload state of each node, the frequency of candidate itemsets may be calculated in its local node or will be transferred to another lightly loaded node within the same site. If the coordinator cannot fix the workload imbalance locally, it selects part of transactions to be sent to a remote site through the network. The destination of migrated work is chosen according to the following hierarchy:

First The coordinator of the cluster $Coord(cl_{ij})$ selects the available node within the same cluster; If the workload imbalance still persists then $Coord(cl_{ij})$ searches for an available node in another cluster but within the same site; Finally, in extreme cases, work will be send to a remote site. The coordinator of the site $Coord(S_i)$ will look for the nearest site available to receive this workload (i.e. least communication cost).

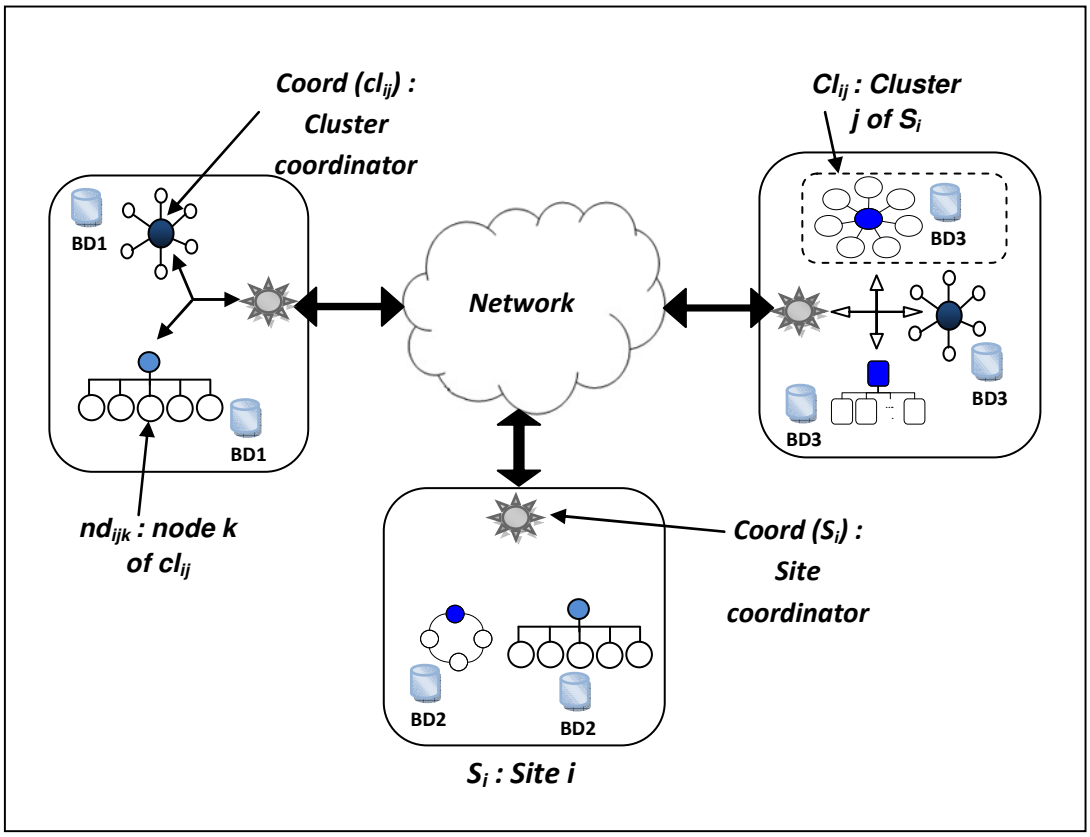

Figure 1. The system model of a Grid.

Our model is fault-tolerant. In fact, it takes into consideration the probability of failure of a coordinator node. If the coordinator node does not give response within a fixed period of time, an election policy is invoked to choose another coordinator node.

5. PROPOSED DYNAMIC LOAD BALANCING STRATEGY

Dynamic load balancing is necessary for the efficient use of highly distributed systems (like Grids) and when solving problems with unpredictable load estimates (like association rule mining). That is why we chose to develop a dynamic work load balancing strategy.

Our proposed load balancing strategy depends on three issues:

1. Database architecture (partitioned or not);
2. Candidates set (duplicated or partitioned);
3. Network *communication parameter (bandwidth).*

The strategy could be adopted by algorithms which depend on candidate itemsets generation to solve the frequent set counting problem. It combines between static and dynamic load balancing and this by *interfering before execution (i.e. static) and during execution (i.e. dynamic).*

Before execution: To respond to the heterogeneity of the computing system we are using (Grid) the database is not just partitioned into equal partitions in a random manner. Rather than that, the transactional database is partitioned depending on the characteristics of different sites, where the size of each partition is determined according to the site processing capacity (i.e., different architecture, operating system, CPU speed, etc.). It is the responsibility of the coordinator of the site Coord(Si) to allocate to its site the appropriate database portion according to the site processing capacity parameters stored in its information system.

During execution: Our load balancing strategy acts on three levels:

4. Level one is the migration of work between nodes of the same cluster. If the skew in workload still persists the coordinator of the cluster *Coord(cl_{ij})* moves to the next level;
5. Level two depends on the migration of work between clusters within the same site;
6. And finally if work migration of the previous two levels is not sufficient then the coordinator of the overloaded cluster *Coord(cl_{ij})* asks from the coordinator of the site *Coord(S_i)* to move to the third level which searches for the possibility of migrating work between sites. Communication between the coordinators of different sites is done in a unidirectional ring topology via a token passing mechanism.

The following workload balancing process is invoked when needed. It is the responsibility of distributed coordinators to detect that need dynamically according to the charge status of their relative nodes:

```
Begin
    For All (cl_{i,j} in S_i) do
        Update (Coord_i , GL_i(L_{i,j}))      // Update the global vector of workload of each site
    End For

    AL_i ← (Σ_{j=1}^{M_i} L_{i,j}) / M_i      // Calculate the average workload of the site S_i

    If AL_i > LMax_i Then                     // The site is overloaded
        Sort GL_i(L_{i,j})                    // The sort is done in a decreasing order to detect the more overloaded node.
        Search in V_i
        If Exists (cl_{p,q} in S_p) Where (EET_{i,j} > Coefinter *( CCS_{i,p} + EET_{p,q})) Then
            Transactions_Migration inter_Site(S_i , S_p)      // Workload balancing inter-sites
        End If
    Else
        If (L_{i,j} > Limit_{i,j}) Then    //An overload is detected in a node N_{i,j}
            If Exists (cl_{i,k} in S_i ) Such that ((L_{i,k} < Limit_{i,k}) AND (EET_{i,j} > (Coefintra *( CCN_{i,j,k} + EET_{i,k})))) Then
                Candidates_Migration intra_Site (cl_{i,j} , cl_{i,k})      // Load balancing intra-site
            Else                          // Load balancing inter-sites
                                          // the migration inter-site is more expensive (in time) than the migration intra-site.
                Search in V_i
                If Exists (cl_{p,q} in S_p) Where (EET_{i,j} > Coefinter *( CCS_{i,p} + EET_{p,q})) Then
                    Transactions_Migration inter_Site(S_i , S_p)
                End If
            End If
        End If
    End IF
End
```

Figure 2. The run time work load balancing strategy.

1. From the intra-site level, coordinators of each cluster update their global workload vector by acquiring workload information from their local nodes. From the Grid level, coordinators of different sites periodically calculate their average workload in order to detect their workload state (overloaded or underloaded). If an imbalance is detected, coordinators proceed to the following steps.
2. The coordinator of the overloaded cluster makes a plan for candidates migration intra-site (between nodes of the same site) using equation *(2)*. If the imbalance still persist, it creates another plan for transactions migration inter-sites (between clusters of the Grid) using equation *(3)*.

$$EET_{i,j} > Coefintra * (CCN_{i,j,k} + EET_{i,k}) \quad (2)$$

$$EET_{i,j} > Coefinter * (CCS_{i,p} + EET_{p,q}) \quad (3)$$

Where $EET_{i,j}$ is the estimated required processing time for node $N_{i,j}$ of the site S_i, $EET_{i,k}$ is the estimated required time to process the same operations in another node $N_{i,k}$ of the same site Si in the case of equation (2), $EET_{p,q}$ is the is the estimated required processing time in another node $N_{p,q}$ of a remote site S_p in the case of equation (2), $CCN_{i,j,k}$ is the communication cost between nodes $N_{i,j}$ and $N_{i,k}$ of the site S_i, $CCS_{i,p}$ is the communication cost between sites S_i and S_p, *Coefintra* is the coefficient of decision of the intra-site migration, and *Coefinter* is the coefficient of decision of the inter-site migration.

The previously mentioned two equations serves in ensuring that before performing any candidate itemsets migration between nodes within the same site, or transactions migration between different sites, the coordinator must guaranty that transactions or candidates migration will improve the performance of the Grid. The processing time at a local node must dominate (by a prefixed threshold) the processing time at a remote node added to it the time spent in

communication and transactions (or candidates) movements. Otherwise, it will be better to process transactions (or candidates) locally. The definition of this coefficient of domination (or threshold) depends of the environment of execution and the size of data to be processed. The coefficient of the intra-site migration is smaller than the coefficient for the inter-sites migration, because the communication cost intra-site is much less than the communication cost inter-sites.

The information used to characterize the load of a node, a cluster or a site are its current load represented by the number of instructions that need to be executed and its speed which is quantified in terms of number of executed instructions per unit of time. In our case all processing time estimates are calculated through the number of candidate itemsets generated at each iteration that the algorithm has to calculate their frequencies. The estimated required processing time $EET_{i,j}$ for a node $N_{i,j}$ is calculated by multiplying the cardinal of candidate itemsets generated at the beginning of a specific iteration by the node's processing time denoted ω_{ij}. And this is done dynamically at the beginning of each iteration of the association rule mining algorithm.

3. The concerned coordinator (the coordinator of the overloaded cluster or the coordinator of the overloaded site) sends migration plan to all processing nodes and instructs them to reallocate the work load.

For each site S_i, the coordinator will execute the algorithm displayed in figure 2. Where *Coord* (S_i) is the coordinator node of the site S_i, M_i is the number of computational clusters, GL_i is the global vector of workloads of all nodes in the site S_i, AL_i is the average workload of the site S_i, $LMax_i$ is the threshold of the maximum workload of the site S_i, L_{ij} is the local workload of the cluster cl_{ij} of the site S_i, $Limit_{i,j}$ is the threshold of the maximum workload of the cluster $cl_{i,j}$ in the site S_i , $CCN_{i,j,k}$ is the communication cost between clusterd $cl_{i,j}$ and $cl_{i,k}$ of the site S_i, $EET_{i,j}$ is the estimated required time for cluster $cl_{i,j}$ of the site S_i to complete the processing of remaining transactions data, $CCS_{i,p}$ is the communication cost between sites S_i and S_p, V_i is the state vector of all the other coordinators in the Grid. The state of the coordinator of each site is stored in the vector with these information: *Id-site*, $CCS_{i,p}$ and L_i. This vector is sorted by $CCS_{i,p}$ and L_i in order to construct a logical ring of communication between sites.

The state of imbalance of a cluster or a site is ascertained on the basis of the current load index, where this index is the ratio between the load and the speed of treatment. Practically this is done by dynamically defining (i.e. during each iteration of the association rule mining algorithm) the load balancing threasholds ($Limit_{i,j}$ for the cluster and $LMax_i$ for the site). Where $Limit_{i,j}$ is defined as the average initial load of $cl_{i,j}$ added to it the standard deviation of the workloads of the nodes of $cl_{i,j}$. This help in measuring the scope of load changes between a cluster and its nodes. $LMax_i$ takes into consideration the average initial load of S_i, added to it the standard deviation of the workloads of the clusters of S_i.

6. PERFORMANCE EVALUATION

We evaluated the performance of the load balancing strategy proposed in section 5 by the use of Grid'5000 the experimental platform dedicated to grid research [1]. We implemented a parallel version of the sequential Apriori [11] and we added to it our work load balancing strategy. We conducted many experiments with different data sets and by varying the support threshold and the number of nodes used for execution.

6.1 Case Study: The Apriori Algorithm

The specific characteristics of the problem of frequent set counting associated with those of the computing environment (Grid) must be taken into account. While association rule mining method is based on global criteria (support, frequencies), we are only disposed by local (partial) data views due to the fact of distribution.

The treatment must be done on the entire database, comparing each partition of the base with all the others must be possible in order to be able to obtain global information.

Our goal is to limit the number of communications and synchronizations, and to be benefit as much as possible from the available computing power. This could be done by exploiting all possible ways of parallelism and if necessary by using a pipeline approach between dependent tasks in order to be able to parallelize the various stages of the frequent set counting algorithm.

In order to evaluate the performance of our workload balancing strategy we parallelized the sequential Apriori algorithm which is the fundamental algorithm for frequent set counting algorithms with candidates' generation. To reduce the number of accesses to the transactional database we used the depth-first Apriori proposed by W. Kosters et al. [16]. This version of Apriori needs only three passes over the transactional database, while classic Apriori needs k-passes (where k is the length of the maximal itemset). Data parallelism is not sufficient to improve the performance of association rule mining algorithms. Subsets of extremely large data sets may also be very large. So, in order to extract the maximum of parallelism, we applied a hybrid parallelisation technique (i.e. the combination of data and task parallelism). This could be done through searching inside the algorithm procedures for independent segments and analyzing the loops to detect tasks (or instructions) that could be executed simultaneously.

A hybrid approach between candidate duplication and candidate partitioning is used. The candidate itemsets are duplicated all over the sites of the Grid, but they are partitioned between the nodes of each site. The reason for partitioning the candidate itemsets is that when the minimum support threshold is low they overflow the memory space and incur a lot of disk I/O. So, the candidate itemsets are partitioned into equivalence classes based on their common *(k-2)* length prefixes. A detailed explanation of candidate itemsets clustering could be found in [8]. We can resume the important basic concepts of our parallelization method in what follows:

Site:

- The transactional database is partitioned between sites according to the capacity of treatment of each site.
- Candidate itemsets are duplicated between sites (in order to reduce the communication cost between sites).

Cluster:

- Every database partition is shared between nodes of the same site if they have the same storage subsystem, otherwise it will be duplicated.
- Candidates are partitioned between site's clusters (according to the capacity of treatment of the cluster)

Node :

- Receives a group of candidates from the coordinator of the cluster.
- Calculates their supports.
- Sends local supports to cluster's coordinator which performs the global supports reduction.

Cluster's coordinator:

- Distributes candidate itemsets between nodes according to their capacities. Candidates are distributed by their (k-1) commun prefix.
- Performs the global reduction of supports to obtain global frequencies.
- Responsible for workload balancing operation of his cluster

Site's coordinator:

- Search for the maximum loaded cluster (or site) and the minimum loaded cluster (or site).
- Migration of the necessary amount of work (candidates or transactions or both) from the maximum to the minimum loaded clusters or sites.

6.2 Experimental Platform

The performance evaluations presented in this section were conducted on Grid'5000 [1], a dedicated reconfigurable and controllable experimental platform featuring 13 clusters, each with 58 to 342 PCs, interconnected through Renater (the French Educational and Research wide area Network). It gathers roughly 5000 CPU cores featuring four architectures (Itanium, Xeon, G5 and Opteron) distributed into 13 clusters over 9 cities in France (Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis, and Toulouse).

We used heterogeneous clusters in order to generate the maximum workload imbalance. Nodes are interconnected within each cluster by a Gigabit Ethernet switch. All the nodes were booted under Linux on Grid'5000. Nodes were reserved by the reservation system which ensures that no other user could log on them during the experiments.

We achieved several experiments, by varying the number of sites, clusters and computational nodes. We will present in what follows only the results obtained by using two sites, each site containing two clusters and with 16 computational nodes distributed as follows: 3 nodes/cluster1, 2 nodes/cluster2, 4 nodes/cluster3 and 7 nodes/cluster4. We allocated clusters with different sizes to show the effectiveness of our approach in dealing with the heterogeneity of the system.

The datasets used in tests are synthetic, and are generated using the IBM-generator [12]. Table 1 shows the datasets characteristics.

Table 1. Transactional databases characteristics

Database	# Items	Avg. Length	# Transactions	Database size
DB70T1M	4000	20	1000000	70 Mb
DB100T13M	4000	25	1300000	100 Mb

Figure 3 depicts the execution time obtained from running the parallel version of Apriori without the work load balancing strategy and the time obtained when the strategy is embedded in the parallel implementation. The database is initially partitioned over different sites, where the size of different portions depends on the site's capacity (CPU speed, memory size, available disk space …). We can clearly see that the parallel execution time with

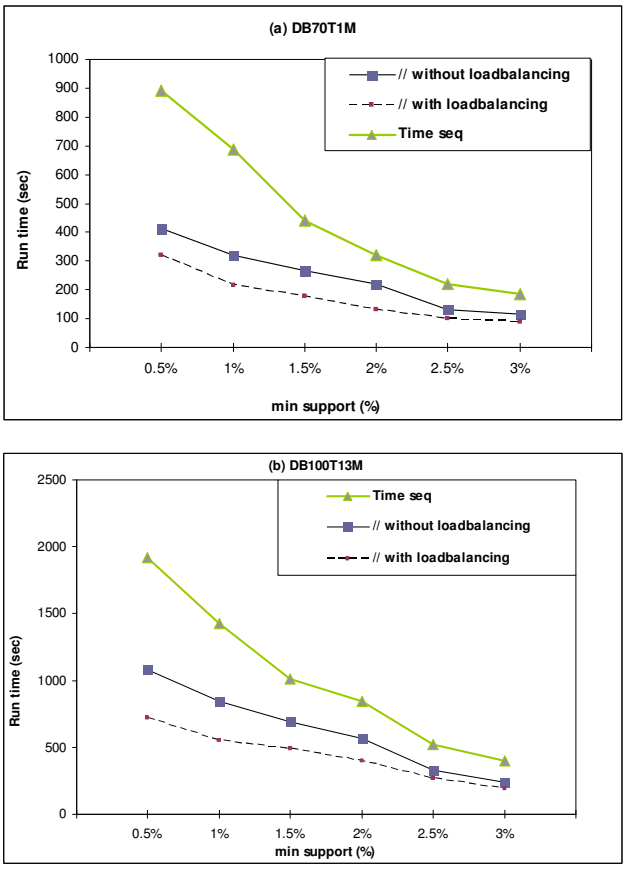

Figure 3. Run time with and without load balancing for different support values.

work load balancing outperforms the time needed for the parallel execution without taking care to the load imbalance that may occur during the execution of the association rule mining algorithm.

Our work load balancing strategy has reduced the execution time of the second data set (DB100T13M) about 40% for very small supports and 20% for larger supports. This shows that as the size of the dataset is large and with small support values (which increases the size of generated candidate itemsets), the amount of work increases. Therefore the probability of work load imbalance between processors is bigger.

Figure 4 shows the time needed for workload balancing (work migration and communication). It is clear that computation time dominates the time needed for communication and work migration, which means that the overhead caused by the proposed workload balancing strategy could be negligible.

We also tried to use small data sets (like mushroom and chess with Kb sizes) and the results from the experiments show us one important issue of data mining on grids. When the data set used is not big enough, the number of parallel nodes used should be decreased or there will not be any improvement on execution time. The overhead of transferring data and results is too big when compared with computing time needed for smaller sets.

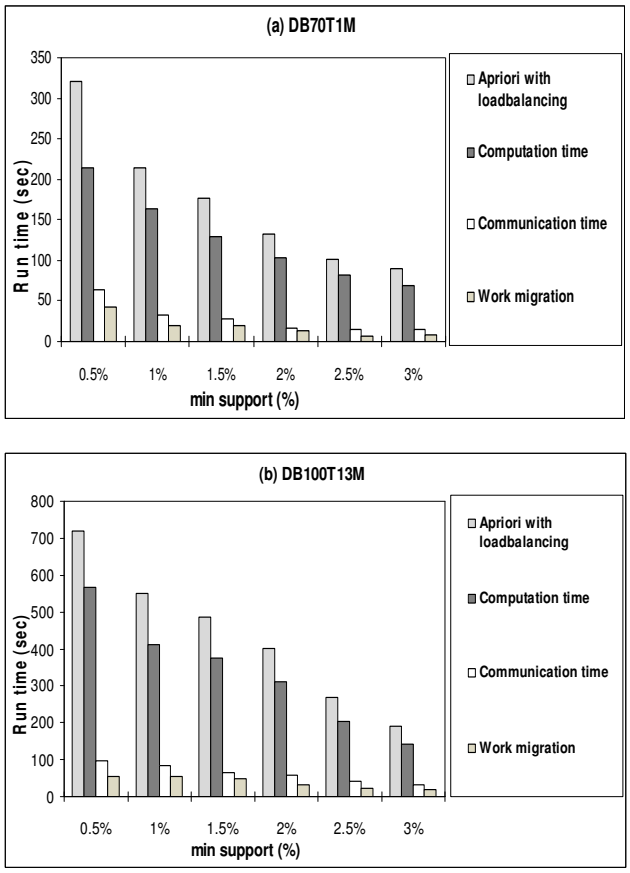

Figure 4. Run time, communication time and workload balancing time for dataSet1.

Figure 5 illustrates the speedup obtained as a function of different support values. We can clearly see that for both datasets we achieved better speed up with the load balancing approach.

The drop in speedup for relatively higher support values is due to the fact that when the support threshold increases the number of candidate itemsets generated decreases (i.e. less computation to be performed). In this case it would be better to decrease the number of nodes incorporated in execution so that the communication cost will not be higher than the computation cost. In fact, there is not a fixed optimal number of processors that could be used for execution. The number of processors used should be proportional to the size of data sets to be mined. The easiest way to determine that optimal number is via experiments.

The first iteration of association rule mining algorithm is a phase of initiation for workload balancing (i.e. creating state vectors and processing time estimates, etc). For the first dataset (DB70T1M) the algorithm performed 10 iterations in order to generate all possible frequent itemsets. Candidate itemsets migration (intra-site) is initiated two times during the second iteration, and once during the third and fourth iterations.

For the second data set (DB100T13M) candidate itemsets migration (intra-site) is launched two times during the second iteration, and transaction migration (inter-sites) is established during the third iteration. Another candidate itemsets migration was needed during the fifth and seventh iteration. The algorithm for the second data set iterates 14 times in order to generate all possible frequent itemsets.

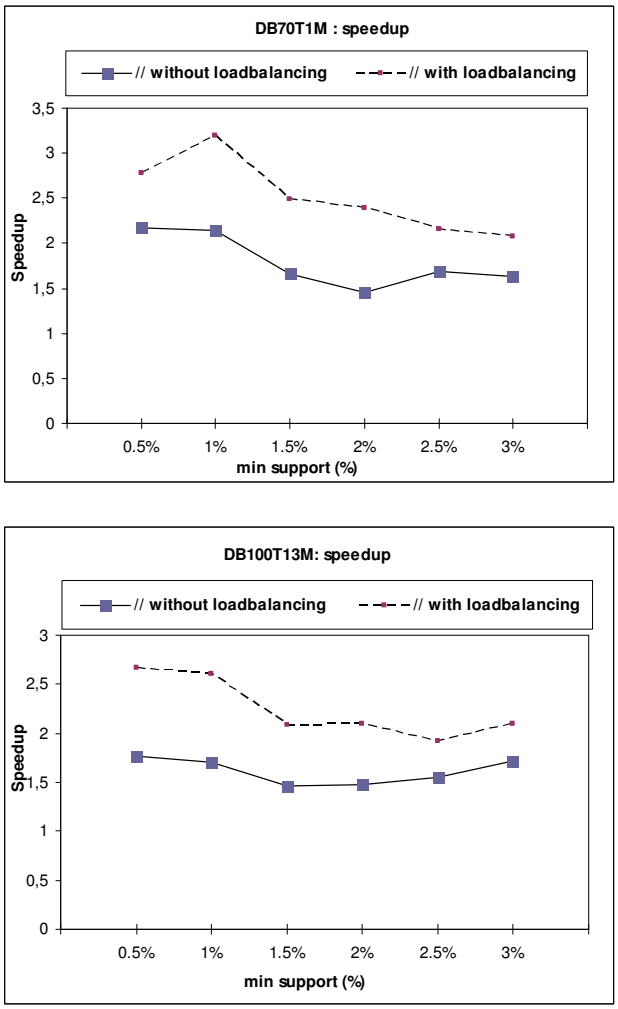

Figure 5. Comparing the speedup of parallel Apriori with and without load balancing

7. CONCLUSION

Data mining could be applied to a diversity of domains, where there exist huge amounts of data that need to be analyzed in order to provide useful knowledge. The knowledge gained can be used in applications ranging from business management, production control, and market analysis, to engineering design and science exploration. As an example of applications that provides ample

opportunities and challenges for data mining, the analysis of software problems of heterogeneous distributed systems. In such systems logs collected from different nodes should be merged in order to create a view of what happened in the system. After that an analysis is needed to detect the cause of different problems. In this case we can apply both descriptive and predictive data mining tasks [3]. Clustering (descriptive data mining technique) can be used in grouping the collected logs based on some specific criteria like the type of operating system used. Then an association rule mining algorithm can be applied on the obtained classes to detect the kind of behavior related to each operating system.

Regardless of the domain of application of data mining, the amount of data to be analyzed is very huge. Serial algorithms lose their efficiency and are not able to provide knowledge quickly. Parallel and distributed systems could be the natural solution to this problem. But implementing such parallel and distributed algorithms is not trivial and is faced by many challenges including load balancing.

Data mining algorithms have a dynamic nature during execution time which causes load-imbalance between the different processing nodes. Such algorithms require dynamic load balancers that adjust the decomposition as the computation proceeds. Numerous static load balancing strategies have been developed where dynamic load balancing still an open and challenging research area. In this article we developed a dynamic load balancing strategy for association rule mining algorithms (with candidate itemsets generation) under a Grid computing environment. Our load balancing strategy acts on three levels. In the first level load is adjusted via the migration of work between nodes of the same cluster. If the skew in workload still persists candidate itemsets migration is established between clusters within the same site. And finally if it is needed the coordinator moves to the third level and searches for the possibility of migrating work between sites.

The load balancing strategy derives several necessary coefficients and time estimates at run time. This is done during each iteration of the association rule mining algorithm in order to respond to the dynamicity of the system. Experiments showed that our strategy succeeded in achieving better use of the Grid architecture assuming load balancing and this for large sized datasets. In the future, we plan to study the impact of the database type (dense and sparse) on our strategy.

8. REFERENCES

[1] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Vicat-Blanc Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, B. Quetier, O. Richard, Grid'5000: a large scale and highly reconfigurable grid experimental testbed, in: SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD, Seattle, Washington, USA, November 2005, IEEE/ACM, pp. 99–106, 2005.

[2] I. Foster and C. Kesselman, The Grid2: Blue print for a New Computing Infrastructure. Morgan Kaufmann, 2003.

[3] J. Han and M.. Kamber. Data Mining : concepts and techniques. Maurgan Kaufman Publishers, 2000.

[4] K. Devine, E. Boman, R. Heaphy and B. Hendrickson, "New Challenges in Dynamic Load Balancing". Appl. Num. Maths, Vol.52, issues 2-3, 133-152, 2005.

[5] K. Wang, L. Tang, J. Han and J. Liu, "Top Down FP-Growth for Association Rule Mining". In Proc. Of the 6th Pacific-Assia Conf. on Advances in Knowledge Discovery and Data Mining, Taipei, pp. 334-370, 2002.

[6] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers". IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 9, pages 979-993, September, 1993.

[7] M. J. Zaki, "Parallel and Distributed Association Mining: a Survey". IEEE Concurrency, 7(4): pp14-25, 1999.

[8] M. J. Zaki, S. Parthasarathy, M. Ogihara and W. Li. "New Algorithms for Fast Discovery of Association Rules". University of Rochester, Technical Report 651, July 1997.

[9] M.S. Perez, A. Sanchez, V. Robles, P. Herrero, J. Pena, Design and Implementation of a data mining grid-aware architecture, Future Generation Computing Systems 23 (1), pp 42–47, 2007.

[10] R. Agrawal and J. C. Shafer. "Parallel Mining of Association Rules". IEEE Transactions on Knowledge and Data Engineering , 8:962-969, 1996.

[11] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In Proc. of the Int'l Conf of VLDB'94, pp 478-499, 1994.

[12] Generator of Databases Site : http://www.almaden.ibm.com/cs/quest.

[13] S. Orlando, P. Palmerini and R. Perego, "A Scalable Multi-Strategy Algorithm for Counting Frequent Sets". In Proc. Of the 4th International Conference on Knowledge Discovery and Data Mining (KDD), New York, USA, 2002.

[14] T. L. Casavant and J. G. Kuhl, "Taxonomy of Scheduling in General Purpose Distributed Computing Systems". IEEE Transactions on Software Engineering, 14(2): 141, February, 1988.

[15] V. Fiolet, B. Toursel, Distributed data mining, Scalable Computing: Practice and Experiences 6 (1), pp 99–109, 2005.

[16] W. Kosters and W. Pijls. Apriori, A Depth First Implementation. In Proceedings of the FIMI Workshop of Frequent Itemset Mining Implementation, Melbourne, Florida, USA, 2003.

[17] Y. Li and Z. Lan, "A Survey of Load Balancing in Grid Computing". Computational and information Science, First International Symposium, CIS 2004, Shanghai, China, 2004.