A Hierarchical Dynamic Load Balancing Strategy for Distributed Data Mining

Raja Tlili and Yahya Slimani

MOSIC, Computer Science Department, Faculty of Sciences of Tunisia
Tunis El Manar University, Tunisia
raja_tlili@yahoo.fr

Abstract

Extracting useful knowledge from data sets measuring in gigabytes and even terabytes is a challenging research area for the data mining community. Sequential approaches suffer from a performance problem due to the fact that they have to mine voluminous databases. Parallelism is introduced as an important solution that could improve the response time and the scalability of these approaches. However, parallelization process is not trivial and still facing many challenges including the workload balancing problem.

In this paper, we propose a hierarchical dynamic load balancing strategy for parallel association rule mining algorithms in the context of a Grid computing environment. The French research grid "Grid'5000" is used as our experimental test-bed. Through a detailed experimental study, we show that our strategy improves the performance and helps the parallel algorithm to scale very well with the number of computational nodes available.

Keywords: *Sequential association rule mining, performance problem, parallelism, workload balancing*

1. Introduction

Data mining is a natural evolution of information technology. This evolution could be witnessed in the following functionalities: data collection and storage, data retrieval and data processing. These functionalities served as a prerequisite for data analysis and understanding. Thus data mining is the knowledge mining from large amounts of data. Data mining involves an integration of techniques from multiple domains such as database technology, high-performance computing, information retrieval, neural networks, statistics, machine learning and image and signal processing [4]. Association rule mining is one of the most important data mining techniques [4, 15]. The effectiveness of this technique is determined by quickly and correctly finding interesting correlation relationships between items in large databases. The discovered knowledge can be useful to decision making, information management and process control. The algorithms of this technique are computationally and input/output intensive. High performance parallel and distributed computing can relieve current association rule mining algorithms from the sequential bottleneck, providing scalability to massive data sets and improving response time.

Grid computing is recently regarded as one of the most promising platform for data and computation-intensive applications like data mining. F. Magoulès et al. define a grid as a hardware and software infrastructure that provides transparent, dependable, pervasive and consistent access to large-scale distributed resources owned and shared by multiple administrative organizations in order to deliver support for a wide range of applications with

the desired qualities of service. These applications can perform high throughput computing, on-demand computing, data intensive computing, or collaborative computing [4]. Contrary to other systems where the focus is to achieve greater performance measured in terms of the number of floating point operations the system can perform per minute, the importance of grids is defined in terms of the amount of work they are able to deliver over a period of time [3]. In such computing environments, heterogeneity is inevitable due to their distributed nature.

Parallel and distributed association rule mining algorithms suffer from a work load imbalance problem during execution time. This work load imbalance is caused by the dynamic nature of these algorithms and also by the heterogeneity of grid systems. Almost all current parallel/distributed algorithms assume the homogeneity and use static load balancing strategies. Thus, applying them to Grid systems will degrade their performance. New methodologies need to be developed to handle this problem, which is the focus of our research.

In this paper, we develop and evaluate a hierarchical and dynamic load balancing strategy for mining association rule algorithms under a grid computing environment. Our major contributions can be summarized as follows: First, we constituted a hierarchical grid model suitable for association rule mining algorithms. Second, we proposed, on the basis of this model, a new dynamic load balancing strategy. Finally, we implemented our strategy and we imbedded it within several parallel association rule mining algorithms. Several tests were conducted using our grid testbed (the French research grid called: Grid5000) where experimental results showed the effectiveness of our approach. To the best of our knowledge, this work could be considered as one of the first attempts to run association rule mining on grids not only correctly but also efficiently.

The rest of the paper is organized as follows: Section 2 introduces association rule mining technique and related work. Section 3 presents parallel/distributed algorithms for mining association rules. Section 4 describes the load balancing problem. Section 5 presents the hierarchical system model of a Grid. Section 6 states the characteristics of the proposed model. In section 7, we propose the dynamic load balancing strategy. Experimental results obtained from implementing this strategy are showed in section 8. Comparisons with existing works are established in section 9. Finally, the paper concludes with section 10.

2. Mining Association Rules

Association Rules Mining (ARM) finds interesting correlation relationships among a large set of data items. 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 [6]. 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.

ARM technique is also widely used in genomics and computational biology. One of its major applications is the analysis of high dimension DNA or protein sequences [6].

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 [17, 21] and without candidate itemsets generation [9].

The Apriori algorithm [17] was the first effective algorithm proposed in the literature. 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.

The DCI algorithm proposed by Orlando and others [21] 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.

The FP-growth algorithm [9] 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.

3. Parallel and Distributed Mining of Association Rules

Sequential 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.

There are three categories of parallel systems:

1. Shared-memory systems, where all processors share together the system memory. Every processor has direct access to the entire dataset. Accessing the same data necessitates synchronization and sequential memory access.
2. Distributed-memory systems. In such systems each processor has its own system memory that cannot be accessed by other processors. Shared data could be transferred by message passing.
3. Hierarchical systems combines shared and distributed systems characteristics. They are constituted of multiprocessor nodes in which memory is shared by intra-node processors but distributed over inter-node processors. These systems use fast networks and can have shared disks.

In order to exploit parallelism in association rule mining algorithms three main paradigms have been followed: data parallelism, task parallelism and hybrid parallelism [11].

1. In data parallelism, the database is partitioned between available processors and each processor executes locally the same task on its local data portion.
2. In task parallelism, different tasks are executed simultaneously and processors, in this case, need to have access to the entire database. If the memory is distributed then the database is replicated. The database is shared in the case where processors have a shared memory.

3. Hybrid parallelism is the combination of data and task parallelism. This type of parallelism is suitable for hierarchical systems (like grid systems).

Data parallelism is used for voluminous databases where the entire database cannot fit in the main memory. When the search space is large (i.e. huge number of candidate itemsets), task parallelism is preferred. Hybrid parallelism is recommended in case we have both situations: a large-scale database and a huge search space.

Many parallel algorithms for solving the frequent set counting problem have been proposed. Most of them use the Apriori algorithm as fundamental algorithm, because of its success on the sequential setting [11]. The Count Distribution (CD) algorithm is based on data parallelism. In this algorithm, the database is partitioned over multiple processors. Each processor executes locally the Apriori algorithm, and performs a global reduction of local candidate itemsets counts, by the end of each iteration, to obtain the global counts. CD presents a bottleneck when a large processors number is incorporated in execution. The Data Distribution (DD) algorithm is another example of Apriori parallelization. In this algorithm candidate itemsets are partitioned over processors. To calculate the global support, each processor needs to scan the entire database during each iteration. DD algorithm induces a communication overhead due to its need of exchanging fragments between processors. The Intelligent Data Distribution (IDD) algorithm proposes a ring-based, all-to-all broadcast mechanism to reduce the communication cost overhead. The entire database is also communicated [13]. The Fast Parallel Mining (FPM) algorithm outperforms the CD algorithm by the use of local and global pruning techniques. FPM algorithm needs only one round of messages exchange in order to broadcast local supports information to all processors which reduces the communication overhead [11]. The Common Candidate Partitioned Database (CCPD) algorithm was proposed by Zaki for distributed memory machines [12]. With this algorithm, candidate itemsets are generated in parallel. Although CCPD obtains reasonable speedup, the sequential I/O is detrimental to performance. The Hybrid Distribution (HD) algorithm combines CD and IDD algorithms. This algorithm is based on task parallelism. The processors are split into groups and the database is partitioned among these groups. Each group executes the CD algorithm and the nodes of each group (i.e. intra-group) execute the IDD algorithm. The HD algorithm succeeded in reducing the database communication costs. Experiments showed that HD algorithm has the same performance as CD, but can handle much larger databases [12].

Algorithms based on data parallelism require less communication overhead when compared with ones based on task parallelism. The hybrid approach reduces the communication overhead and offers better load balancing. All these approaches have been proposed for parallel machines with shared or distributed memory architecture and having fast interconnection networks. In case of relatively slow networks (like grid systems), the large number of frequent candidate itemsets will produce high communication cost.

Association Rule Mining (ARM) on Grids is a very promising research area. Grid technology provides interesting solutions to data mining. This is due to three main reasons:

1. Grid Platforms provides nontrivial qualities of service, such as security, response time, throughput and availability so that the utility of the combined systems is greater than the sum of its parts.
2. Association rule mining technique is used on very large datasets and grid platforms can help in ameliorating the performances and reducing the execution time.

3. In many cases, association rule mining technique is executed on datasets residing on different geographical locations where a grid can be used to integrate the data sources into one virtual data set.

There exist many grid data mining projects that provide mechanisms for integration and deployment of classical ARM algorithms on grid. WekaG [14] is an application that performs data mining tasks on a grid by implementing a vertical architecture called Data Mining Grid Architecture (DMGA), which is based on the data mining phases: pre-processing, data mining and post-processing. The application implements client/server architecture. The server side is responsible of a set of grid services that implement the different data mining algorithms and data mining phases. A user interface is provided to allow the client to interact with the server.

GridMiner [15] is a Service oriented grid application that integrates all aspects of the data mining process: data cleaning, data integration, data transformation, data mining, pattern evaluation, knowledge presentation and visualization. Currently implemented data mining services available in this tool are: sequential clustering service, sequential sequence service, parallel OLAP and sequential association rule mining in OLAP cubes.

The focus of the previously mentioned grid data mining projects was to run data mining tasks correctly but not efficiently and there were no performance or efficiency issues available in the obtained results.

C. Yang et al. proposed a heuristic data distribution scheme for data mining applications on grid environments [1]. The grid is represented in a master/slave model. This model is represented by a star graph $G = \{P_0, P_1, \ldots, P_n\}$ where P_0 is the master node and the other n nodes, $P_1, \ldots, P_n$, are slave nodes. In addition, there is a virtual communication link L_i connecting the master node and the slave node P_i. They proposed a performance based heuristic to solve the data partition problem for association rule mining applications.

K. Yu et al. proposed a weighted distributed parallel Apriori algorithm [8] in which the transaction identifier of itemsets is stored in a table to compute their occurrence. The algorithm takes the factor of itemsets counts into consideration in order to balance workloads among processors and reduce processor's idle time.

These approaches succeeded in reducing the communication cost overheads between processors but suffer from a scalability problem due to the fact that they are centralized (i.e. only one master and several slaves).

4. Load Balancing and Data Mining

Parallelizing sequential ARM algorithms is not a trivial straightforward process. The parallelization of these algorithms introduces a plethora of new problems including the workload balancing problem.

Work load balancing is the assignment of work to processors in a way that maximizes application performance [7]. The process of load balancing 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 under-loaded ones.

According to different policies used in the previously mentioned phases, Casavant and kuhl [22] classify workload 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.

Static load balancing can be used in applications with constant workloads, as a pre-processor to the computation [7]. Other applications require dynamic load balancers that adjust the decomposition as the computation proceeds [7, 10]. 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.

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. [25], 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 [25]:

1. Repartition methods focus on calculating data distribution in a heterogeneous way, but do not pay much attention to the data movement in Grid;
2. Divisible load theory based schemes well model both the computation and communication, but loose validity in case of adaptive application;
3. Prediction based schemes need further investigation in case of long-term applications.

C. Yang et al. proposed a heuristic data distribution scheme for data mining applications on grid environments [1]. They induced load balancing through a heuristic data partition technique that aims to reduce the total execution time of the program. K. Yu et al. proposed a weighted distributed parallel Apriori algorithm [8] in which the transaction identifier of itemsets is stored in a table to compute their occurrence. The algorithm takes the factor of itemset counts into consideration in order to balance workloads among processors and reduce processor's idle time.

5. The Hierarchical Grid Model

Our major contributions can be summarized as follows: First, we constituted a hierarchical grid model suitable for association rule mining algorithms. Second, we proposed, on the basis of this model, a new dynamic load balancing strategy. Finally, we implemented our strategy and we imbedded it within several parallel ARM algorithms.

We first present the constitutive elements of a grid that we will use later to define our model. From a topological point of view, we consider that a grid, as illustrated in Figure1 is

consisted of a set of connected sites, in a star topology, through a global network WAN (Wide Area Network). Each site is composed of a set of clusters each of which is constituted of computing nodes communicating through a local network LAN (Local Area Network) and belonging to the same domain.

The set of resources and the means of communications can be heterogeneous in terms of architectures, the capacity of processing and of storage and networks bandwidths.

To represent a grid composed of the elements described above, we propose to transform it, univocally, in a ring of trees of virtual interconnection as shown in Figure1. This tree is generated according to the aggregation process described below:

1. To each site is associated a tree with two levels:
    - The root level of this tree is a coordinator. Its role is to control and manage the load of the site;
    - The next level corresponds to clusters comprising the site.
2. To each cluster is associated a tree with two levels:
    - The root level of this tree is a coordinator node. Its role is to control and manage the load of the cluster;
    - The next level, representing the leaves, corresponds to the computing nodes of the cluster. Their role is to execute in parallel the association rule mining algorithm.

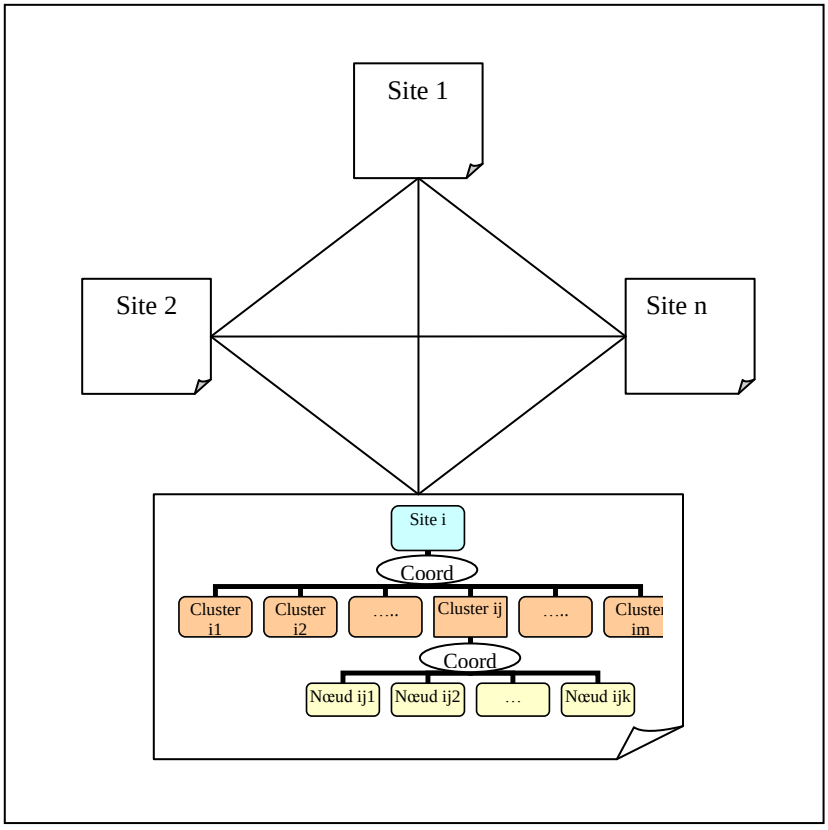

Figure 1. The Hierarchical System Model of a Grid

3. The ring is obtained by the aggregation of the roots of the two-level trees associated to different sites in order to represent the entire grid system:
    - A coordinator for each site. The coordinator may be a cluster with a size (i.e. number of physical nodes) relative to the number of computing nodes included in the execution of the distributed algorithm. The main function of the coordinator is to manage the load of the site. Different coordinators communicate with each other in unidirectional ring topology via a token passing mechanism.
    - Computational clusters with their tree of computational nodes defined previously.

Each site can be instantiated in two configurations denoted respectively $1/N$ and C/N, where N is the number of nodes and C the number of clusters of a site.

5.1. The 1/N Model

This instance of the model represents the smallest possible site in a grid, i.e. a site with a single cluster consisting of N computing nodes. This model includes two levels defined as follows:

1. The root level of this tree is the workload manager of the cluster called cluster coordinator. This coordinator has:
    a. A workload Balancing Task, consisting of:
        - Managing the load information of cluster's nodes, which we call the state vector of the cluster;
        - Maintaining the charge status of the cluster;
        - Deciding to start a local load balancing (i.e. intra-site);
        - Informing the computing nodes, to execute the load balancing instructions decided by the cluster coordinator.
    b. A knowledge extraction task, constituted of :
        - Distributing candidate itemsets over different computing nodes. This distribution is dynamic and is intelligent in order to respond to the heterogeneity of grid systems;
        - Receiving the local supports calculated by computing nodes to perform the global reduction operation;
        - Constructing the list of frequent itemsets;
        - Constructing the list of candidate itemsets of the next iteration.
2. The leaves level of the tree, where each leaf corresponds to a computing node. The nodes of this level have:
    a. A workload Balancing Task, consisting of:
        - Updating their charge status (i.e. the status vector of a node);
        - Sending their status vector periodically to the coordinator of the cluster;
        - Executing the load balancing operations decided by the coordinator.

b. A knowledge extraction task, constituted of :

- Receiving candidate itemsets (from the coordinator);
- Calculating their supports;
- Sending local supports to the coordinator.

5.2. The C/N Model

This model represents an extension of the previous model in the sense that we move from one cluster to *C* clusters. We obtain the *C*/*N* defined by a ring of the roots of three-level trees representing the aggregation of *C* models of the type 1/*N* that we can define as follows:

1. The root level of this tree is the workload manager of the site called site coordinator. This coordinator has:
   a. A workload Balancing Task, consisting of:
      - Managing the load information of the site's clusters (i.e. updating global state vector);
      - Maintaining the load status of the site;
      - Deciding to start an inter-sites load balancing;
      - Sending load balancing decisions to different cluster's coordinators to start their execution.
   b. A knowledge extraction task, constituted of :
      - Allocating the appropriate portion of transactions to its clusters;
      - Performing the global reduction operation with the coordinators of other sites to obtain global supports;
      - Constructing the list of frequent itemsets;
      - Constructing the list of candidate itemsets of the next iteration.
2. Every site coordinator is the root of the second level which is constituted of the coordinators of different clusters. These clusters coordinators play the same role as in the model 1/*N*.
3. The third level corresponds to the physical nodes of the grid (i.e. computing nodes) as defined for the leaves in 1/*N* model.

6. Characteristics of the Proposed Model

The model of representation defined previously can be characterized by the following points:

1. The modeling of a grid in a ring topology where each node is the root of a three-level tree is done by a univocal transformation. For every grid corresponds one and only one representing model and this regardless of the topological complexity of the grid. We can consider this ring of trees as a ring of recovery of a grid that will help us in defining the algorithm associated to the workload balancing strategy.

2. The hierarchical structure of the model facilitates the flows of information through different nodes. In terms of information flows, we distinguish four types of flows:
    - Circulating flow: This flow is for the circulation of load information between sites (i.e. global state vector).
    - Ascending flow: Related to the flow of load information at different higher levels (intra-site and intra-cluster).
    - Descending flow: This flow helps to convey the decisions of load balancing taken by the coordinators of different levels of the model.
    - Horizontal flow: It concerns the information necessary for the execution of load balancing operations and also for the execution of the association rule mining algorithm.
3. The proposed model supports the heterogeneity and the dynamicity of the resources. The connection/disconnection of a resource corresponds to addition/deletion operations of a leaf in the relative tree and is regulated by a recovery procedure.
4. The choice of the coordinator of each site and each cluster is governed by a well defined election policy
5. Similarly, the failure/disconnection of one of the coordinators triggers a procedure of fault tolerance for its replacement in a timely within a reasonable period and without prejudice to ongoing treatment.

7. The Hierarchical Workload Balancing Strategy

The proposed modeling of the grid allowed us to define a hierarchical and distributed load balancing strategy. So we distinguish two levels of load balancing: Intra-site and inter-sites.

1. Intra-site load balancing: At this first level, each site coordinator decides to initiate a load balancing operation based on the current load of the site that it manages. This charge is estimated from the different load information (state vectors) sent periodically by the coordinators of the clusters that compose the site. The site coordinator tries as a priority to balance the load by distributing locally between different clusters which are under its control. This approach to locality aims to reduce communication costs, by avoiding inter-sites communications which use the WAN.
2. Inter-sites load balancing: The load balancing, at this second level, is triggered when some sites coordinators fail in their attempts to balance charges locally through their respective sites. The failure of local load balancing may be due to the saturation of the site, or to insufficient charge offer induced by the lightly-loaded cluster with respect to the request formulated by overloaded nodes. In this case, the site coordinator tries to find another site which is capable of accepting the current overload. This search is accomplished by negotiating the transfer of candidate itemsets, transactions or both from the overloaded site to the under-load site.

Our strategy could be adopted by any association rule mining algorithm that depends on candidate itemsets generation. It combines between static and dynamic load balancing by interfering before execution (i.e. static) and during execution (i.e. dynamic).

To respond to the heterogeneity of the computing system we are using (i.e. grid systems), the database is not partitioned into equal partitions in a random manner. Rather than that, the transactional database is partitioned according to 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(S_i)* to allocate to its site the appropriate database portion according to the site processing capacity parameters stored in its information system.

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:

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 under-loaded). 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). If the imbalance still persists, it creates another plan for transactions migration inter-sites (between clusters of the Grid).
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.

7.1. The Workload Balancing Algorithms

The two main advantages of the workload balancing strategy are:

- The priority is given to local workload balancing (i.e. intra-cluster). The objective of that is to privilege local communications (LAN network) in order to reduce the overhead caused by the transfer of work or data.
- The strategy is totally distributed but the decision is taken locally. Actually, we can execute in parallel as much intra-cluster load balancing as much we have clusters in the grid.

Following is the dynamic load balancing algorithms:

Module node :

Loop :

- 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.
- Every n steps :
  - Updates NSV (Node State Vector)
  - Sends NSV to the Cluster

Module cluster coordinator

- *Init Execution:* Receive Partition of the DB.
- *Normal Execution:*

  Loop :
  - Distributes candidate itemsets between nodes according to their capacities. Candidates are distributed by their (*k-1*) common prefix.
  - Performs the global reduction of supports to obtain global frequencies.
  - Constructs frequent itemsets (L_k step).
  - Constructs candidates itemsets of the following iteration (C_{k+1} step).
  - Every n steps :
    - Saves the local state;
    - Updates if necessary C_{k+1} step.
- *Monitoring :*
  - ***Every n steps :***
    - Receives NSV (Node State Vector)
    - Checks if any overload in nodes
    - Updates CSV (Cluster State Vector)
    - Sends CSV
    - Checks if an Overload is detected in some nodes
- *Load Balancing :*
  - ***If an Overload is detected:***
    - Checks (CSV)
    - Searches_Candidate, in nodes, to balance the load
    - **If** Find **then** Start_load balance (intra-cluster)
    - **Else** asks Site to Start_load Balance (intra-Site)

Module site coordinator

Loop :

- Updates the global state vector of the site: average(ch_i)).
- Finds the Max overloaded cluster and the max under-loaded cluster:
  - $Cl_{ijmax} \rightarrow \max_j (ch_{ij}) > \text{average}(ch_i)$
  - $Clijmin \rightarrow \min_j (ch_{ij}) < \text{average}(ch_i)$
- Finds the Max x_c (with the same prefix) on cl_{ijmax} (intra-Site):
  1. $Ch_{ijmin} + x_c \cdot \omega_{ijmin} \leq \text{average}(ch_i)$

     //To find the best number of candidates to migrate in order to not overload the destination cluster

     AND
  2. $x_c \cdot \omega_{ijmax} - (x_c \cdot \omega_{ijmin} + \text{long}(x_c).\zeta_{ijmaxjmin}) > Seuil_{mc}$
- **If** x_c exists **Then** informs the overloaded c_{lijmax} and the underloaded c_{lijmin} and updates (ch_i).
  - **Asks** from the overloaded cluster to send the family of candidates having the same prefix.
- **Else** asks other sites to balance the load (inter-Sites).

Where:

$\zeta_{ijj'}$: Transmission speed between clusters cl_{ij} and $cl_{ij'}$;
ω_{ijk} : Cycle time of nd_{ijk};
ch_i : Charge of S_i;
ch_{ij} : Charge of cl_{ij};
ω_{ij} : Average(ω_{ijk});
$seuil_{mc}$: Significant time limit to trigger candidate itemsets migration between clusters;
$seuil_{mt}$: Significant time limit to trigger task migration between sites;
x_c : Number of candidates to migrate from one cluster to another.

All load balancing algorithms are executed in parallel with the ARM algorithm without inducing an overhead in execution time. Computing nodes continue working even during work or data migration.

8. Performance Evaluation

A preliminary, partially distributed, version of our load balancing approach [18] was tested, via simulations, by using the GridSim toolkit developed by Buyya and Murshed [26]. A modified version of our approach [19] was tested under a real grid: Grid'5000. We will report in this section the tests of the last and final version of our hierarchical, dynamic and totally distributed load balancing strategy performed under Grid'5000.

8.1. Parallelization Approach

Our goal is to limit the number of communications and synchronizations, and to 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 ARM algorithm.

In order to evaluate the performance of our workload balancing strategy we parallelized several ARM algorithms. We will report in what follows the results obtained from parallelizing the sequential Apriori [17] which is the fundamental algorithm for ARM algorithms with candidate itemsets generation.

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). Where we aimed to study parallelism inside the program code. 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 [13].

We can resume the important basic concepts of our parallelization method in what follows:

- *Site:* The transactional database is partitioned between sites according to their capacity of treatment. Candidate itemsets are duplicated between sites (in order to reduce the communication cost between sites).
- *Cluster:* Every database portion is shared between nodes of the same site if they have the same storage subsystem, otherwise it will be duplicated. Candidate itemsets are partitioned between site's clusters according to their capacity of treatment.
- *Node :* It receives a group of candidate itemsets from the coordinator of the cluster. It calculates their supports and 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) common prefix. This coordinator performs the global reduction of supports to obtain global frequencies. It is also the responsible of workload balancing operations of its cluster
- *Site's coordinator:* Searches for the maximum loaded cluster (or site) and the minimum loaded cluster (or site). After that, it decides the migration of the necessary amount of work (candidates or transactions or both) from the maximum to the minimum loaded clusters or sites.

8.2. Experimental Platform

The experimental testbed for our tests is Grid'5000 [2], a dedicated reconfigurable and controllable experimental platform. The infrastructure of Grid'5000 is geographically distributed on different sites hosting the instrument, initially 9 sites in France (10 since 2011). It gathers roughly 5000 CPU cores featuring four architectures (Itanium, Xeon, G5 and Opteron) distributed into more than 13 clusters.

Grid5000's nodes are accessible through the OAR batch scheduler, from a central user interface shared by all the users of the cluster. The home directories of the users are mounted with NFS on each of the infrastructure's clusters. Data can thus be directly accessed inside a cluster. Data transfers between clusters have to be handled by the users. The storage capacity inside each cluster is a couple of hundreds of gigabytes [2].

8.3. Experiments

The datasets used in tests are synthetically generated. Table 1 describes the datasets characteristics.

Table 1. Transactional Databases Characteristics

Database	Number Items	Avg. Transaction Length	Number Transactions	Database size
DB300T39M	6000	30	3900000	300 Mb
DB500T63M	7200	35	6300000	500 Mb
DB900T132M	9500	47	13200000	900 Mb

Table2 specifies the number of sites, clusters and nodes used to execute the ARM algorithm. In order to generate the maximum workload imbalance, we used heterogeneous

sites and clusters. Programs were realized using C++ and MPI on a linux heterogeneous environment. Pipeline mechanisms for the distinct phases were implemented.

Table 2. Hardware Distribution

	Number of sites	Number of clusters	Maximal Number of Nodes
Experiment 1	2	8	32
Experiment 2	3	6	32

For both experiments 1 and 2 the support value is fixed and the number of nodes incorporated in execution is varied from 1 to 32. Figure 2 plots the results of Experiment 1. It displays the execution time obtained from running the parallel version of Apriori algorithm without paying attention to the workload imbalance that occurs during execution and the time obtained when our hierarchical workload balancing strategy is embedded in the parallel implementation.

Figure 3 shows the results of Experiment 2. It displays the execution time obtained when using three sites. Each site contains 2 clusters with different physical characteristics and having different number of computational nodes. With 32 processors, our workload balancing strategy has reduced the execution time of about 70%.

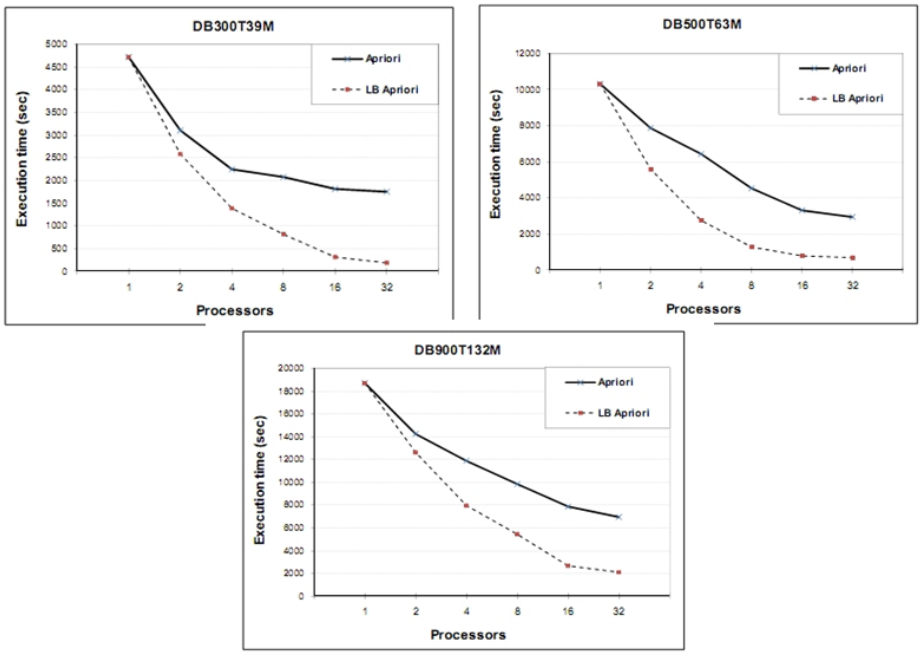

Figure 2.: Execution Time of Experiment 1 (Apriori with and without load balancing).

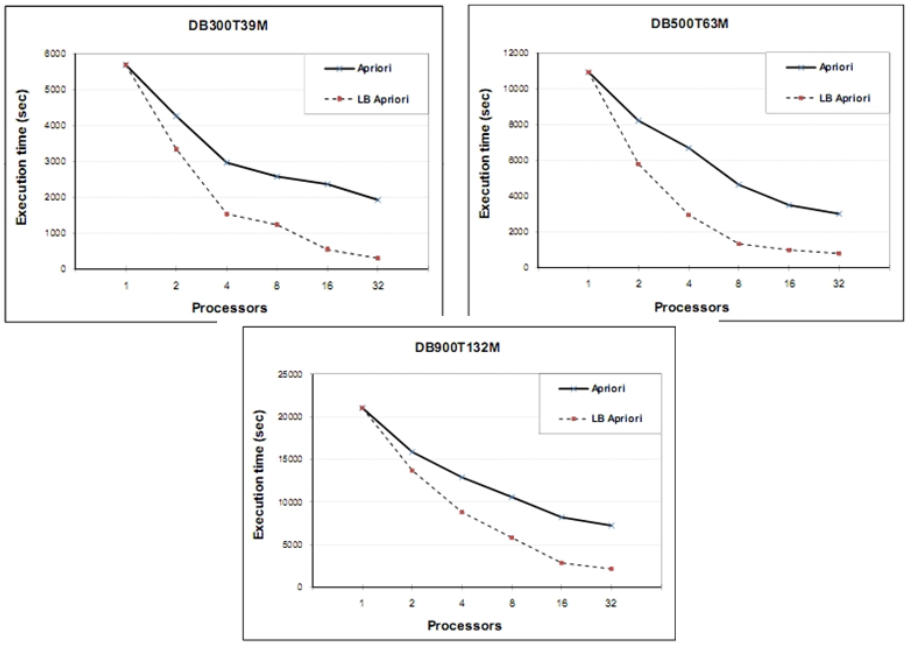

Figure 3. Execution Time of Experiment 2 (LB Apriori is the load balanced Apriori).

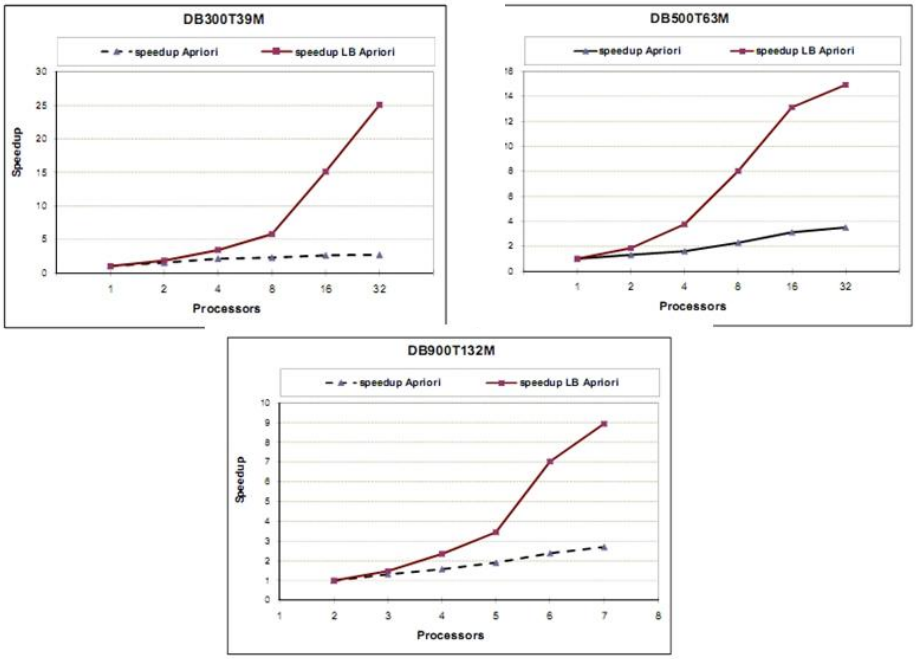

Figure 4. Speed up of Experiment1 of Apriori with and without load balancing.

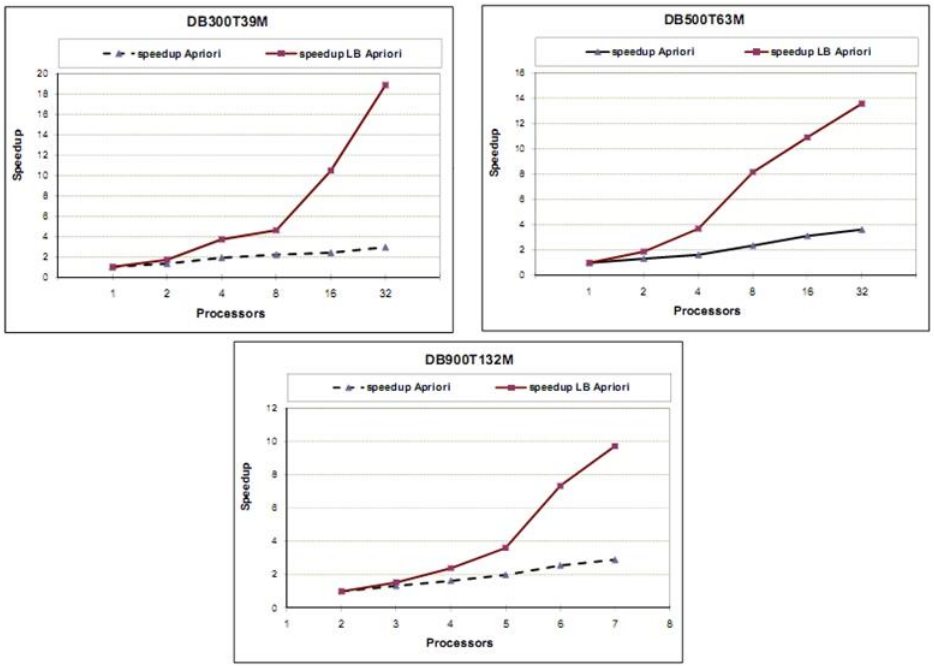

Figure 5. Speed up of Experiment 2.

Figures 4 and 5 illustrate the speedup obtained, from experiments 1 and 2 respectively, as a function of the number of processors used in execution. We can clearly see that for the different datasets we achieved better speedup with the workload balancing approach.

9. Discussion

After parallelizing the Apriori algorithm, several tests were conducted where we varied the support threshold and the number of nodes incorporated in execution. This was done with large size datasets. We noticed that the execution time increases instead of decreasing when we increase the number of computing nodes. We took detailed measurements of the execution time within each computing node. We remarked a big skew in the computing time between different nodes before reaching a barrier of synchronization in order to consolidate results (i.e. to calculate the global support from different local supports). This is due to two reasons. The first reason is the skew in heterogeneous system (i.e. computing nodes have different characteristics). The second reason is that the workload depends on the itemset. Some itemsets have higher support value, which means that they need more computing time. So after partitioning the database among different processors, the amount of work needed for different portions is not equal. This increases processors idle time behind each barrier of synchronization and thus the total execution time increases. The problem is that the real support value is obtained only after the execution. So an appropriate partitioning that is based on the amount of work that should be done on each dataset portion is not possible before execution. In order to benefit from increasing the number of computing nodes, dynamic load balancers are needed to adjust the decomposition as the computation proceeds. In our dynamic load balancing approach, we exploit information from the previous iteration of the ARM algorithm. Our decision to establish a load balancing process at iteration *k* is based on the information on the support value of itemsets at iteration *k-1*.

The first iteration of the ARM algorithm is a phase of initiation for workload balancing. Different state vectors are created and time estimates are calculated for next iterations. For experiment 1 and for the first datasets (DB300T39M), the ARM algorithm performed 26 iterations in order to generate all possible frequent itemsets. Candidate itemsets migration (intra-site) is initiated two times during the second iteration, two times during the fourth iteration and once during the seventh, tenth and the sixteenth iterations. Transactions migration (intersites) is initiated once during iteration 12.

There is not a fixed optimal number of processors that should be used for execution. The number of processors used must be proportional to the size of data sets to be mined. The easiest way to determine that optimal number is via experiments.

We can notice that the execution time increases when 3 sites are incorporated in execution in experiment 2 instead of 2 sites in experiment 1. This is due to the fact that communication cost between clusters of different sites is high when compared with intra-site communication. Our strategy improves the performance and helps the parallel algorithm to scale very well with the number of computational nodes available.

We tested the scale-up behaviour of the workload balanced Apriori when the size of the dataset or of the computational nodes increased. We used the configuration of power of 2 computational nodes and expanded the system from 1 node to 2, 4, 8, 16 and 32 nodes. The results presented in Figure 2 and 3 shows that the execution time of the workload balanced Apriori decreases remarkably as the number of nodes increases. Thus, the workload balanced Apriori scale much better than the classical parallel Apriori.

Approaches Comparison

Table 3 illustrates the differences between: our Hierarchical Dynamic Load Balancing Approach (HDLBA), the Weighted Distributed Parallel Apriori algorithm (WDPA) proposed in [8] and Heuristic Data Distribution Scheme (HDDS) introduced in [1].

Table 3. Approaches Comparison

Approach Name	Approach used to Balance Load	Characteristics	Speed Up (using 9 processors)
WDPA	One Master/P Slaves	Centralized	7.5
HDDS	One Master/P Slaves	Centralized	5.5
HDLBA	Hierarchy of coordinators	Distributed	8.95

Both WDPA and HDDS are based on a centralized (master/slave) load balancing approach where there is one master responsible of data distribution and n computing slaves. This would cause a scalability problem as the number of computing nodes increases. Our approach is totally distributed in order to respond to the high level of distribution in grid systems.

10. Conclusion

Due to their dynamic nature, ARM algorithms require dynamic load balancing approaches capable of adjusting loads as the computation proceeds. In this paper we developed a dynamic load balancing strategy for ARM algorithms under a Grid computing environment. Experimentations showed that our strategy improves the performance and helps the parallel algorithm to scale very well with the number of computational nodes available. In the future, we aim to adopt our strategy to ARM algorithms without candidate itemsets generation.

Acknowledgements

- We wish to thank Professor Margaret H. Dunham and Dr. Michael Hahsler for providing us with large datasets for tests.
- Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER, CPER Nord-Pas-de-Calais/FEDER Campus Intelligence Ambiante, several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] C. T. Yang, W. C. Shih, and Shian-Shyong Tseng, "A Heuristic Data Distribution Scheme for Data Mining Applications on Grid Environments", IEEE World Congress on Computational Intelligence (FUZZ-IEEE 2008), pp.2398-2404, **(2008)** June.

[2] 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, **(2005)**, pp. 99–106.

[3] Frédéric Magoulès, Jie Pan, Kiat-An Tan and Abhinit Kumar, Introduction to Grid Computing, CRC Press Taylor & Francis Group, **(2009)**.

[4] Frédéric Magoulès, Thi-Mai-Huong Nguyen and Lei Yu, Grid Resource Management Toward Virtual and Services Compliant Grid Computing, CRC Press Taylor & Francis Group, **(2009)**.

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

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

[7] 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)**.

[8] Kun-Ming Yu, Jia-Ling Zhou. A Weighted Load-balancing Parallel Apriori Algorithm for Association Rule Mining, In Proceedings of the IEEE International Conference on Granular Computing (GrC'08), **(2008)**.

[9] 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)**.

[10] 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)**.

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

[12] Mohammed J. Zaki, Ching-Tien Ho, Large-scale parallel data mining, Vol (1957), Springer, **(2000)**.

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

[14] 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)**.

[15] P. Brezany, J. Hofer, A Min Tjoa, A. W¨ohrer. GridMiner: An Infrastructure for Data Mining on Computational Grids. In Proceedings of the APAC Conference and Exhibition on Advanced Computing, Grid Applications and eResearch, Queensland Australia, **(2003)** October.

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

[17] 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)**.

[18] Raja Tlili and Yahya Slimani, Dynamic load balancing for large-scale distributed association rule mining, In Proceedings of the 9th International Symposium on Performance Science (Algiers, Algeria), **(2009)** May.

[19] Raja Tlili and Yahya Slimani, computing environment, In Proceedings of the 9th International Workshop on Parallel and Distributed Systems : Testing, Analysis, and Debugging (PADTAD'11) held in conjunction with 11th International Symposium on Software Testing and Analysis (ISSTA'11) Toronto, Canada, **(2011)** July.

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

[21] 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)**.

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

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

[24] 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)**.

[25] 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)**.

[26] R. Buyya and M. Murshed. GridSim: a Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. Concurrency Computation: Practice and Experiments, **(2002)**.