Mining Association Rules on Grid Platforms

Raja Tlili and Yahya Slimani

Department of Computer Science, Faculty of Sciences of Tunisia,
Campus Universitaire, 1060 Tunis, Tunisia
raja_tlili@yahoo.fr, yahya.slimani@fst.rnu.tn

Abstract. In this paper we propose a dynamic load balancing strategy to enhance the performance of parallel association rule mining algorithms in the context of a Grid computing environment. This strategy is built upon a distributed model which necessitates small overheads in the communication costs for load updates and for both data and work transfers. It also supports the heterogeneity of the system and it is fault tolerant.

Keywords: Association rules, Performance problem, Distributed algorithms, Grid computing, Dynamic load balancing.

1 Introduction

The fast development of data acquisition and storage technologies led to an exponential growth in worldwide data. In order to decrease the gap between data and useful information, a group of architectures and utilities, some of them are new and others exist since a long time, are grouped under the term data mining. Association rule mining is one of the most important data mining techniques [8, 3]. The most important challenge for this technique is quickly and correctly finding interesting correlation relationships between items in large databases. The algorithms of this technique are computationally and input/output intensive, due to the fact that they have to mine voluminous databases. 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 [9] is recently regarded as one of the most promising platform for data and computation-intensive applications like data mining. In such computing environments, heterogeneity is inevitable due to their distributed nature.

Almost all current parallel association rule mining algorithms assume the homogeneity and use static load balancing strategies. Thus applying them to Grid systems will degrade their performance. The load imbalance that occurs during execution time is caused by the dynamic nature of these algorithms and also by the heterogeneity of such distributed systems. Because of that we have to develop new methodologies to handle this problem, which is the focus of our research.

In this paper, we develop and evaluate a run time load balancing strategy for mining association rule algorithms under a grid computing environment.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 201–210, 2012.

The rest of the paper is organized as follows: Section 2 introduces association rule mining technique and related work. 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 Related Work

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 [8]. 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 $A \cap B = \emptyset$. The rule's support is the joint probability of a transaction containing both A and B at the same time, and is given as $\sigma(A \cup B)$. The confidence of the rule is the conditional probability that a transaction contains B given that it contains A and is given as $\sigma(A \cup B)/\sigma(A)$. 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.

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 [3, 11] and without candidate itemsets generation [14].The Apriori algorithm [3] 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 [11] 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 [14] 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. Sequential 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 [3] 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 [18]. Agrawal et al. proposed a broad taxonomy of parallelization strategies that can be adopted for Apriori in [2].

There also exist many grid data mining projects, like Discovery Net, GridMiner, DMGA [12] 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 [7].

3 Load Balancing: Problem Definition

Work load balancing is the assignment of work to processors in a way that maximizes application performance [6]. 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; and (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 [5] 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.

Static load balancing can be used in applications with constant workloads, as a pre-processor to the computation. Other applications require dynamic load balancers that adjust the decomposition as the computation proceeds [6, 15]. 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. [10], 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 [16]: (1) Repartition methods focus on calculating data distribution in a heterogeneous way, but don't 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 [16]. 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 [17] 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 idle time.

4 The Grid Model

In our study we model a Grid as a collection of T sites with different computational facilities and storage subsystem. Let $G = (S_1, S_2, ..., S_T)$ 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 $cl_{ij'}$). 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. 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

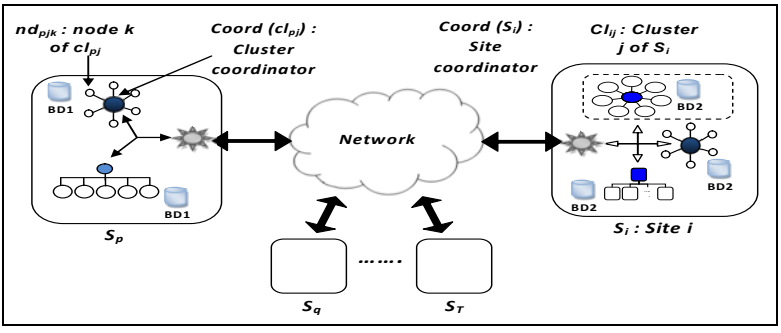

Fig. 1. The system model of a Grid

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). 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 The Dynamic Load Balancing Strategy

Our proposed load balancing strategy depends on three issues: (i) Database architecture (partitioned or not); (ii) Candidates set (duplicated or partitioned); (iii) network communication parameter (bandwidth).

Our strategy could be adopted by algorithms which depend on candidate itmesets 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).

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 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's 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.

Our load balancing strategy acts on three levels: (1) 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; (2) level two depends on the migration of work between clusters within the same site; (3) 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. This choice was made based on the study conducted by H. Renard [13] which states that the ring topology is the most effective for iterative algorithms under distributed environments.

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 (i.e. equilibrated or overloaded). Where the charge status of a node is determined by the number of candidates waiting for treatment:

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.

5.1 The Dynamic Load Balancing Algorithms

Computing node (`nd`$_{ijk}$)

```
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
```

Cluster coordinator (`coord(`cl_{ij}`)`)

```
Loop :
 -- 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
 -- Constructs frequent itemsets (L_k step)
 -- Constructs candidates itemsets of the following iteration C_(k+1)
    step
 -- Every n steps :
      ◦ Save the local state : ch_ij ,
      ◦ Update if necessary C_(k+1) step
```

Site coordinator (Coord(S_i))
Loop :
-- Updates the global state vector of the site Average(chi))
-- Finds the Max overloaded cluster and the max underloaded cluster
- $Cl_{ijmax} \Rightarrow {}^{max}_{j}\ (ch_{ij}) > Avg(ch_i)$
- $Cl_{ijmin} \Rightarrow {}^{min}_{j}\ (ch_{ij}) < Avg(ch_i)$

-- Finds the Max x_c (with the same prefix) on Cl_{ijmax}
1. $Ch_{ijmin} + x_c.\omega_{ijmin} \leq Avg(ch_i)$
   // To find the best number of candidates to migrate in order to not overload the destination cluster

AND

2. $x_c.\omega_{ijmax} - (x_c.\omega_{ijmin} + len(x_c).\zeta_{ijmaxjmin}) > Seuil_{mc}$
   // $Seuil_{mc}$: le seuil qui va dclencher la migration

-- If x_c exists Then informs the overloaded cl_{ijmax} and the underloaded cl_{ijmin} and updates (ch_i)
-- Asks from the overloaded cluster to send the family of candidates having the same prefix

Where T is the total number of sites; M_i is the total number of clusters of the site S_i ; N_{ij} is the total number of nodes of the cluster cl_{ij} ; $Coord(cl_{ij})$ is the coordinator node of cl_{ij} ; $coord(S_i)$ is the coordinator of S_i ; $\zeta_{ijj'}$ is the transmission speed between clusters cl_{ij} and $cl_{ij'}$; ω_{ijk} is the cycle time of nd_{ijk} ; ch_i is the charge of S_i ; ch_{ij} is the charge of cl_{ij} ; ω_{ij} is the average (ω_{ijk}); $seuil_{mc}$ is the significant time limit to trigger candidate itemsets migration between clusters; $seuil_{mt}$ is the significant time limit to trigger task migration between sites and x_c is the number of candidates to migrate from one cluster to another.

6 Performance Evaluation

In order to evaluate the performance of our workload balancing strategy we parallelized the sequential Apriori which is the fundamental algorithm for frequent set counting algorithms with candidate itemsets generation. It is important to mention that our load balancing could be applied to the entire class of association rule mining algorithms that depends on candidate itemsets generation.

The performance evaluations presented in this section were conducted on Grid'5000 [4], 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. We used heterogeneous clusters in order to generate the maximum workload imbalance. We conducted several experiments, by varying the number of sites, clusters and computational nodes. Due to space limitation, we will present in what follows only the results obtained by using two sites, each site containing two clusters and with 20 computational nodes distributed as follows: 4 nodes/cluster1, 3 nodes/cluster2, 6 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 [1]. Table 1 shows the datasets characteristics.

Table 1. Transactional databases characteristics

Database	Characteristics			
	Items Number	Avg. Trans. Length	Transactions Number	Database Size
DB100T13M	4000	25	1300000	100 Mb
DB300T39M	6000	30	3900000	300 Mb
DB600T78M	8000	35	7800000	600 Mb

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 (DB100T13M) the algorithm performed 11 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. Figure 2 illustrates the speedup obtained as a function of the number of processors used in execution. We can clearly see that for the different 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.

Performance's Comparisons: Table 2 illustrates the differences between: Our Dynamic Load Balancing Approach (DLBA), the Weighted Distributed Parallel Apriori algorithm (WDPA) proposed in [17] and Heuristic Data Distribution Scheme (HDDS) introduced in [16]. 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. Our approach is totally distributed in order to respond to the high level of distribution in grid systems. For input size 1000 transactions and by using 9 processors, the speed up obtained by the WDPA algorithm is

Table 2. Approaches comparison

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

equal to 7.5, the speed up obtained by executing the HDDS algorithm is equal to 5.5, while the speed up of our dynamic load balancing approach (DLBA) is equal to 8.95.

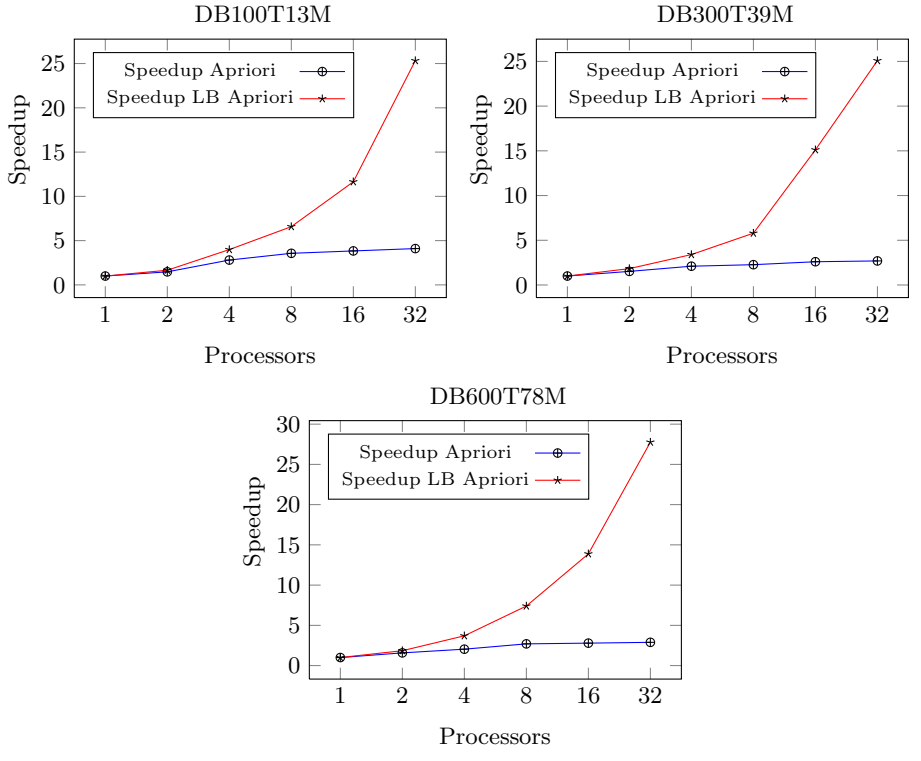

Fig. 2. Comparing the speedup of parallel Apriori with and without load balancing

7 Conclusion

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. Experimentations 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 effect of the database type (dense and sparse) on our strategy. We also aim to adopt our strategy to association rule mining algorithms without candidate itemsets generation.

References

[1] Generator of databases site, http://www.almaden.ibm.com/cs/quest
[2] Agrawal, R., Shafer, J.C.: Parallel mining of association rules (December 1996)
[3] Agrawal, R., Srikant, R.: Fast algorithms for mining associations rules in large databases (September 1994)
[4] Cappello, F., Caron, E., Dayde, M., Desprez, F., Jegou, Y., Primet, P.V.B., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Quetier, B., Richard, O.: Grid'5000: a large scale and highly reconfigurable grid experimental testbed (November 2005)
[5] Casavant, T.L., Kuhl, J.G.: Taxonomy of scheduling in general purpose distributed computing systems (February 1988)
[6] Devine, K., Boman, E., Heaphy, R., Hendrickson, B.: New challenges in dynamic load balancing (2005)
[7] Fiolet, V., Toursel, B.: Distributed data mining. in scalable computing: Practice and expériences (scpe) (March 2005)
[8] Han, J., Kamber, M.: Data mining: concepts and techniques (2000)
[9] Foster, I., Kesselman, C.: The Grid2: Blue print for a New Computing Infrastructure (2003)
[10] Li, Y., Lan, Z.: A survey of load balancing in grid computing (2004)
[11] Orlando, S., Palmerini, P., Perego, R.: A scalable multi-strategy algorithm for counting frequent sets (2002)
[12] Perez, M., Sanchez, A., Robles, V., Herrero, P., Pena, J.: Design and implementation of a data mining grid-aware architecture (2007)
[13] Renard, H.: Equilibrage de Charge et Redistribution de donnes sur Plates-formes htrognes (December 2005)
[14] Wang, K., Tang, L., Han, J., Liu, J.: Top down fp-growth for association rule mining in pakdd 2002 (May 2002)
[15] Willebeek-LeMair, M.H., Reeves, A.P.: Strategies for dynamic load balancing on highly parallel computers (September 1993)
[16] Yang, C.T., Shih, W.C., Tseng, S.S.: A heuristic data distribution scheme for data mining applications on grid environments (June 2008)
[17] Yu, K.M., Zhou, J.L.: A weighted load-balancing parallel apriori algorithm for association rule mining (2008)
[18] Zaki, M.: Parallel and distributed association mining: A survey (December 1999)