A Dynamic Load Balancing Strategy for Parallel Association Rule Mining in Grid Computing

TLILI Raja

Department of Computer Science

Faculty of Sciences of Tunisia
Campus universitaire, 1060 Tunis

Tunisie

Raja_tlili@yahoo.fr

SLIMANI Yahya

Department of Computer Science

Faculty of Sciences of Tunisia
Campus universitaire, 1060 Tunis

Tunisie

yahya.slimani@fst.rnu.tn

RÉSUMÉ. La parallélisation des algorithmes d'extraction des règles associatives est une approche qui vise à améliorer leurs performances, permettant le passage à l'échelle et la réduction des temps de calcul. Cependant, la parallélisation de ces algorithmes n'est pas triviale et fait face à plusieurs défis, notamment celui de l'équilibrage de charge.

L'objectif de cet article et de proposer une stratégie dynamique d'équilibrage de charge pour l'exécution d'algorithmes d'extraction des règles associatives (basés sur l'algorithme Apriori) dans des grilles de calcul. Le but de cette stratégie est de palier au problème de l'hétérogénéité de ces architectures et à l'aspect dynamique de ces algorithmes.

ABSTRACT. Parallelisation approach is a good technique to enhance the performance of sequential data mining algorithms. However, parallelizing these algorithms is not a trivial task and is facing many challenges including the workload balancing problem.

The goal of this paper is to propose a dynamic load balancing strategy for parallel association rule mining algorithms in the context of a Grid computing environment. The main objective of this strategy is to reduce the exponential complexity of sequential data mining algorithms based on Apriori algorithm.

MOTS-CLÉS : Règles associatives, Grilles de calcul, Equilibrage dynamique de charge, Algorithme Apriori.

KEYWORDS: Association rules, Grid computing, Dynamic load balancing, Apriori algorithm.

1. Introduction

The demand for high performance computing continues to increase everyday. The computational need of a wide range of scientific and commercial applications cannot be met even by the fastest computers available [6]. Grid computing is recently regarded as one of the most promising platform for data and computation-intensive applications like data mining. A Grid can be envisioned as a collection of geographically dispersed computing and storage resources interconnected with high speed networks and effectively utilized in order to achieve performances not ordinarily attainable on a single computational resource [1]. In such computing environments heterogeneity is inevitable due to their distributed nature.

The Association rules mining technique which trends to find interesting correlation relationships between items in a large database of sales transactions has become one of the most important data mining techniques and has attracted a great deal of attention in the information industry in recent years. Although algorithms of this technique have a simple statement, they are computationally and input/output intensive. So, implementing these algorithms under a Grid environment seems to be interesting. Almost all current parallel algorithms assume the homogeneity and thus use static load balancing strategies. These strategies are essentially based on initially partitioning the transactional database and no further interfering is done during execution time to correct the load imbalance caused by the dynamic nature of the algorithms of this technique and also by the heterogeneity of such distributed systems. Thus applying them to Grid systems will degrade their performance due to the load imbalance between resources that appear during execution time. Because of that we have to develop methodologies to handle this problem, which is the focus of our research.

In this paper, we propose a run time load balancing strategy for association rules mining algorithms under a Grid computing environment. The rest of the paper is organized as follows. Section 2 introduces parallel association rule mining. Section 3 describes the load balancing problem. Section 4 presents the system model of a Grid, and proposes a dynamic load balancing strategy. Experimental results obtained from implementing this strategy are shown in section 5. Finally, the paper concludes with section 6.

2. Parallel association rule mining

2.1. Association rule

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 [2]. Each basket represents a different transaction in the transactional database, associated to this transaction the items bought by a customer. An example of an association rule over such a database could be that “80% of the customers that bought bread and milk also bought eggs”.

Given a transactional database ***D***, an association rule has the form ***X=>Y,*** where ***X*** and ***Y*** are two itemsets, and $X \cap Y = \varnothing$. The rule's support is the joint probability of a transaction containing both ***X*** and ***Y*** at the same time, and is given as $\sigma(X \cup Y)$. The confidence of the rule is the conditional probability that a transaction contains ***Y*** given that it contains ***X*** and is given as $\sigma(X \cup Y)/\sigma(X)$. 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. Association rules 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 [2].

2.2. Parallel association rules mining algorithms

Association rules 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 [6] 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 [4]. Agrawal et al. proposed a broad taxonomy of parallelization strategies that can be adopted for Apriori in [6].

3. Load balancing: problem description

3.1. General framework for load balancing

Load balancing is the assignment of work to processors [3]. 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 based on their capacities. If all communication links are of infinite bandwidth, the load distribution would suffer from no delay, but this does not represent real distributed environments. Therefore, load balancing is also a decision making process of whether to allow loads migration or not. In addition to that, the load on each processor as well as on the network can vary from time to time based on the workload brought about by users. With all these factors taken into account, load balancing can be generalized into four basic steps: ***(1)*** Monitoring processor load and state; ***(2)*** Exchanging workload and state information between processors; ***(3)*** Calculating the new workload distribution; and ***(4)*** Actual data movement [3].

In applications with constant workloads, *static load balancing* can be used as a pre-processor to the computation. Other applications, such as data mining, have workloads that are unpredictable and change during the computation. Hence, this kind of applications requires *dynamic load balancers* that adjust the decomposition as the computation proceeds.

Parallel association rule mining algorithms have a dynamic nature because of their dependency on the degree of correlation between itemsets in the transactional database. 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

Grid computing is recently regarded as one of the most promising platforms for data-intensive applications like data mining. In such systems heterogeneity is inevitable. Although intensive works have been done in load balancing, the different nature of a Grid computing environment from the traditional distributed system, prevents existing load balancing schemes from benefiting large-scale applications. An excellent survey from Y. Li and others [8], displays the existing solutions and the new efforts in 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 [8]: ***(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.

4. Proposed system model

4.1 Load balancing model

In our study we model a data Grid as a collection of *n* sites with different computational facilities and storage subsystem. Let $G = (S_1, S_{2,\ldots,} S_n)$ denotes a set of sites, where each site is defined as a vector with four parameters $S_i = (NN_i, CN_i, Mem_i, Band_i)$, where NN_i is the number of computational nodes, CN_i is the coordinator node of S_i, Mem_i is the memory size and $Band_i$ is the bandwidth size of the network. Figure 1 shows the Grid system model. The proposed load balancing model is centralized intra-site, but distributed 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. The 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 site, the coordinator may calculate the frequency of candidate itemsets locally or transfer them to other sites. If the coordinator cannot fix the workload imbalance locally, the set of candidates will be sent to a remote site through the network. This site is chosen according to two features: ***(1)*** the global vector of workloads containing the workload information of each site;

(2) the nearest site available to receive this workload (i.e. least communication cost). Before performing any candidates migration between sites, or transactions migration between nodes within the same site, the coordinator makes use of the following two equations to choose the appropriate migration plan:

$$EET_{i,j} > 2*(CCN_{i,j,k} + EET_{i,k}) \qquad (1)$$

$$EET_{i,j} > 3*(CCS_{i,p} + EET_{p,q}) \qquad (2)$$

where $EET_{i,j}$ is the estimated required time for node $N_{i,j}$ of the site S_i to complete the processing of remaining transactions data, $EET_{i,k}$ is the estimated required time to process these transactions in another node $N_{i,k}$ of the same site S_i in the case of equation (1), $EET_{p,q}$ is the estimated required time to process these transactions 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, *and* $CCS_{i,p}$ is the communication cost between sites S_i and S_p. In other words, the coordinator guaranties, by the use of previously mentioned equations, if transactions or candidates migration will improve the performance of the Grid. The processing time at a local node must dominate 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 these transactions locally.

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

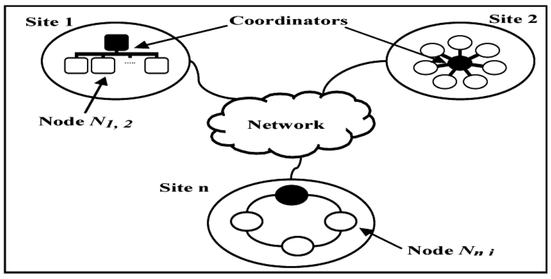

Figure 1. The system model of a Grid

4.2 Load balancing strategy

Our proposed load balancing strategy depends on three issues: (1) Database architecture (partitioned or not); (2) Candidate itemsets tree (duplicated or partitioned); (3) network communication parameter (bandwidth). Apriori algorithm for finding frequent itemsets [8] is the kernel for most data mining parallel algorithms. That is why we have chosen to address our strategy to Apriori-based algorithms [6]. This class of algorithms has an iterative nature (an itemset of length *K* is derived by scanning the transactional database *K* times), with a complete and bottom-up search. Our strategy balances the workload in two levels: the first level before execution, the second level during execution (at run time).

1. Before execution: As displayed in figure 2, the transactional database is partitioned over all Grid sites (clusters), where the size of each partition is determined according to the site

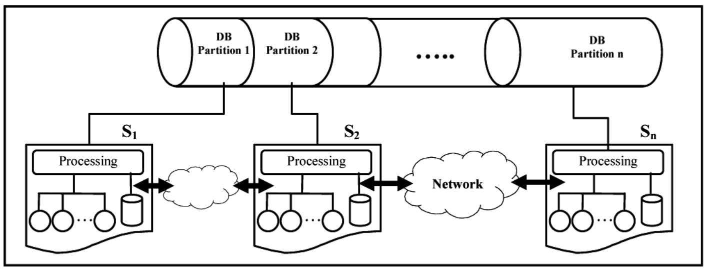

Figure 2. Database partitioning between different sites

processing capacity (i.e., different architecture, operating system, CPU speed, memory size and available disk space).

2. During execution: A hybrid approach between candidates duplication and candidates partitioning is used. The candidate itemsets are partitioned all over the sites of the Grid, but they are duplicated within each site nodes. This approach is chosen for the following reason: when the minimum support threshold is low the candidate itemsets overflow the memory space and incur a lot of disk I/O. So, partitioning the candidate itemsets over all sites, allows us to fully utilize the memory space and reduce disks I/O. At iteration *k*, 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 [5]. From the Grid level, the coordinators of different sites periodically exchange their global state information. From the intra-site level, the coordinator updates its global workload vector by acquiring workload information from each node in its local site. Using these information, each coordinator calculates its workload. In the case of imbalance, it proceeds to the following steps.
   - **(i)** The coordinator makes a plan for transactions migration intra-site (within the same cluster) using equation (1). If the imbalance still present, it also creates another plan for candidate migration inter-sites (between clusters of the Grid). The overhead of network transfer caused by transaction migration is larger then the overhead incurred by candidate migration. That is why we chose to reallocate transactions inside the cluster, and candidates between different clusters.
   - **(ii)** The coordinator sends migration plan to all processing nodes and instructs them to reallocate the workload.

For each site S_i, the coordinator will execute the algorithm displayed in figure 3. Where: GL_i = global vector of workloads of the nodes $N_{i,j}$ in the Site S_i ; $L_{i,j}$ = local workload of the node $N_{i,j}$ of the site S_i ; $Limit_{i,j}$ = a pre-fixed threshold of the workload of the node $N_{i,j}$ of the site S_i ; $CCN_{i,j,k}$ = communication cost between nodes $N_{i,j}$ and $N_{i,k}$ of the site S_i ; $EET_{i,j}$ = estimated required time for node $N_{i,j}$ of the site S_i to complete the processing of remaining transactions; $CCS_{i,p}$ = communication cost between sites S_i and S_p ; V_i= vector of the state of all the other coordinators in the Grid. The state of each site coordinator is stored in the vector with these informations: *Id-site*, $CCS_{i,j}$ and L_i. This vector is sorted by $CCS_{i,j}$ and L_i.

```
Start
    For $N_{i,j}$ in $S_i$ do
        // Update the global vector of workload of each site
        Update $GL_i(L_{i,j})$
        If ($L_{i,j} > Limit_{i,j}$) Then     //An overload detection
            If Exists ($N_{i,k}$ in $S_i$) Such that ($L_{i,k} < Limit_{i,k}$ and $EET_{i,j} > 2*(CCN_{i,j,k} + EET_{i,k})$)
                Then
                    Migration intra-site($N_{i,j}$, $N_{i,k}$)  // Load balancing intra-Site
            Else  // Load balancing inter-Site
                // the migration inter-site is more expensive (in time) than the migration intra-site.
                Search in $V_i$
                If exists ($N_{p,q}$ in $S_p$) where ($EET_{i,j} > 3*(CCS_{i,p} + EET_{p,q})$) Then
                    Migration inter-site($S_i$, $S_p$)  // Workload balancing inter-Site
                End If
            End If
        End If
    End For
End
```

Figure 3. The run time work load balancing strategy

5. Performance evaluation

We evaluated the performance of the load balancing strategy proposed in section 4 by using the GridSim toolkit proposed by Buyya and Murshed [9]. By the help of the previously mentioned simulator, we can specify the desired Grid configuration: number of sites, number of nodes in each site, their characteristics, time period for sending the state vector, network bandwidth, etc. In order to do our tests, we implemented a parallel version of the sequential Apriori and we added to it our work load balancing strategy. The program code of the parallel version was first implemented, tested and ameliorated on an IBM SP2 distributed memory machine of 8 clusters before running it under the GridSim toolkit environment. We conducted many experimentations (with different Grid configurations and with different datasets). Due to pages limitation, we will present in what follows only the results obtained by using fours sites, each site containing eight nodes. The dataset used in tests is synthetic. The database size is 100 MB, it contains 1300000 transactions and 4000 items, and the average size of each transaction is 25. Figure 4 displays 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. We can clearly see that the parallel execution time with 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.

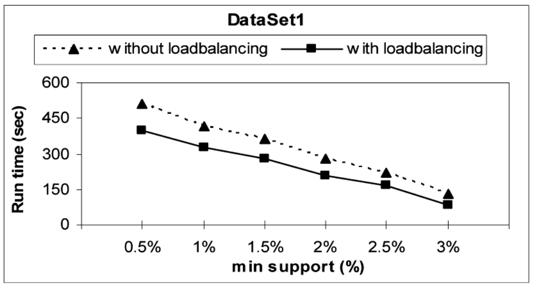

Figure 4: Run time with and without load balancing for different support values

Our work load balancing strategy has reduced the execution time of about 30%.

6. Conclusion

In this article we developed a dynamic load balancing strategy for association rule mining algorithms under a Grid environment. The first experimentations of our strategy have showed that we reduce the execution time of association rule algorithms and we obtain a good distribution of the workload among processors of a Grid. In the future, we plan to experiment our strategy on dense and sparse databases in order to study the effect of the database type on our strategy.

7. References

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

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

[3] 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.

[4] M. J. Zaki. Parallel and distributed association mining: a survey. IEEE, Concurrency, 7(4): pp14-25, 1999.

[5] 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.

[6] R. Agrawal and J. C. Shafer. Parallel mining of association rules. IEEE Transactions on Knowledge and Data Engineering , 8:962-969, 1996.

[7] R. Buyya and M. Murshed. GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing. Concurrency Computat.: Pract. Exper., 2002.

[8] 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.