Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp19-24)

A Parallel Optimization Framework in Grid Environment

Kun Gao[1,2], Hongshan Yang[1], Kexiong Chen[2], Meiqun Liu[3], Jiaxun Chen[1]
[1]Information Science and Technology College, Donghua University, P.R.C
[2]Aviation University of Air Force, P.R.C
[3]Administration of Radio Film and Television of Jilin Province, P.R.C

Abstract: - Grid is a solution to computationally and data intensive computing problems. Since the distributed knowledge discovery process is both data and computational intensive, the Grid is a natural platform for deploying a high performance data mining service. The approach to efficient data mining is parallelization, where the whole computation is broken up into parallel tasks. Existing mechanisms of data mining parallelization are based on NOW or SMP, it is necessary to develop new parallel mechanism for grid feature. We propose a new framework for easily and efficiently parallelizing data mining algorithms on Grid. The framework decomposes tasks according to each of the existing computing power of grid.

Key-Words: - Optimization, Parallelism, Frameworks, Computational grids

1 Introduction

Knowledge Grid is a software architecture for geographically distributed PDKD (Parallel and Distributed Knowledge Discovery) systems [1]. This architecture is built on top of a computational Grid and Data Grid that provides dependable, consistent, and pervasive access to high-end computational resources[2] [3]. The Knowledge Grid uses the basic Grid services and defines a set of additional layers to implement the services of distributed knowledge discovery on world wide connected computers where each node can be a sequential or a parallel machine.

The K-Grid provides a specialized broker of Grid resources for PDKD computations: given a user's request for performing a DM analysis, the broker takes allocation and scheduling decisions, and builds the execution plan, establishing the sequence of actions that have to be performed in order to prepare execution, actually execute the task, and return the results to the user. The execution plan has to satisfy given requirements and constraints-available computing power. Once the execution plan is built, it is passed to the Grid Resource Management service for execution. Clearly, many different execution plans can be devised, and the Resource Allocation and Execution Management services have to choose the one which minimizes response time [4, 5]. In order to obtain the minimal response time, an efficient approach for data mining applications is to parallelize the tasks in grid environment.

However, many data mining applications have proved difficult to parallelize, because various pruning mechanisms are used extensively in data mining applications and a powerful pruning mechanism leads to a highly variable search process that conflicts with a uniform workload requirement for good performance [6]. Fortunately, the computation is coarse grain parallel, i.e., it can be parallelized into large, seldom interacting tasks. Coarse grain parallel computations are suitable computations for Grid [7].

In this paper, we propose a new parallel framework on grid for three classes of data mining problems, i.e. association rule mining, classification rule mining, and pattern discovery. According to each existing computing power of grid, the broker can decompose a data mining task to resources available. The rest of this paper is organised as follows: in section 2, we present the framework that parallelize data mining tasks in Grid environment. In section 3, we present how to optimize the parallel framework. Finally section 6 concludes this paper.

2 Parallelizing data mining tasks in Grid environment

2.1 Modeling Data Mining Applications

We surveyed three major classes of data mining applications, namely association rule mining, classification rule mining, and pattern discovery in combinatorial databases. We note the resemblance among the computation models of these three application classes. Table 1 is a comparison of specifications of these three classes of applications.

A task is the main computation applied on a pattern. Not only are all tasks of any one application of the same kind, but tasks of different applications

are actually very similar. They all take a pattern and a subset of the database and count the number of records in the subset that match the pattern. In the classification rule mining case, counts of matched records are divided into c baskets, where c is the number of distinct classes.

	Pattern Discovery	Assoc. Rule Mining	Class. Rule Mining
Data-base	Sequences	Transaction records	Database relation
Pattern	Partial sequence	Itemset	Attribute-value condition
Good pattern	$occurrence_{pattern}$ >min_occurrence	$support_{pattern}$ >min_support	$Info\_gain_{attribute}$= $Max_{sibling\_attributes}$ ($info\_gain_{attribute}$)
task	Counting occurrence of pattern in subset of database	Counting support of itemset over subset of database	Building histogram of pattern on classes over subset of database

Table 1 A comparison of specifications of three classes of data mining applications classes.

The similarities among the specifications of these applications are obvious, which inspired us to study the similarities among their computation models. As we can see from previous sections, they usually follow a generate-and-test paradigm-generate a candidate pattern, then test whether it is any good. Furthermore, there is some interdependence among the patterns that gives rise to pruning, i.e., if a pattern occurs too rarely, then so will any superpattern. These interdependences entail a lattice of patterns, which can be used to guide the computation.

In fact, this notion of pattern lattice can apply to any data mining application that follows this generate-and-test paradigm. We call this application class pattern lattice data mining. In order to characterize the computation models of these applications more concretely, we define them more carefully in Section 2.2.

2.2 Defining Data Mining Applications

In general, a data mining application defines the following elements.

1 A database D.

2 Patterns and a function len(pattern p) which returns the length of p. The length of a pattern is a non-negative integer.

For example, patterns of length k in the three application class are shown below:

Sequence pattern discovery	$*C_1C_2\ldots C_k*$ $C_1, C_2,\ldots,C_k$ are letters
Association rule mining	$\{i_1, i_2,\ldots, i_k\}$ $i_1, i_2,\ldots, i_k$ are items
Classification rule mining	$(A_1=v_{1i1})\Omega(A_2=v_{2i2})\Omega\ldots\Omega(A_k=v_{kik})$ $A_1, A_2,\ldots, A_k$ are attributes, and $V_{1i1}, v_{2i2},\ldots, v_{kik}$ are attribute values

We use **, {}, and f ; to represent zero-length patterns in sequence pattern discovery, association rule mining, and classification rule mining, respectively.

3 A function goodness(pattern p) which returns a measure of p according to the specifications of the application.

For example, the goodness of a pattern p in sequence pattern discovery is the occurrence number of p in the set of sequences; the goodness of a pattern p in association rule mining is the support of p over the set of items; the goodness of a pattern p in classification rule mining is the info_gain by partitioning the training set by p.

4 A function good(p) which returns 1 if p is a good pattern or a good subpattern and 0 otherwise. Zero-length patterns are always good.

For example, in sequence pattern discovery, a pattern p is good if goodness(p) is greater than or equal to some prespecified min_occurrence; in association rule mining, pattern p is good if goodness(p)>=some prespecified min support; in classification rule mining, a pattern p is good if goodness(p)>=goodness(p'), for any p' such that len(p') = len(p).

Sometimes there are such additional requirements as minimum and/or maximum lengths of good patterns (e.g., in sequence pattern discovery). Without loss of generality, we disregard these requirements in our discussion unless otherwise noted.

The result of a data mining application is the set of all good patterns. If a pattern is not good, neither will any of its superpatterns be. In other words, it is necessary to consider a pattern if and only if all of its subpatterns are good.

Let us define an immediate subpattern of a pattern q to be a subpattern p of q where len(p) = len(q)-1. Conversely, q is called an immediate superpattern of p.

For example, immediate subpatterns of a length k pattern p is as follows:

Sequence pattern discovery	All (k-1)-prefixes and All (k-1)-suffixes of p
Association rule mining	All (k-1)-itemsets
Classification rule mining	The pattern consisting of the first k-1 attribute-value pairs in p

Except for the zero-length pattern, all the patterns in a data mining problem are generated from their immediate subpatterns. In order for all the patterns to be uniquely generated, a pattern q and one of its immediate subpatterns p have to establish a

childparent relationship (i.e., q is a child pattern of p and p is the parent pattern of q). Except for the zero-length pattern, each pattern must have one and only one parent pattern. For example, in sequence pattern discovery, *FRR* can be a child pattern of *FR*; in association rule mining, {2, 3, 4} can be a child pattern of {2, 3}; and in classification rule mining, (C = c1)^(B = b2)^(A = a1) can be a child pattern of (C = c1)^(B = b2).

2.3 Solving Data Mining Applications

Having defined data mining applications as above, it is easy to see that an optimal sequential program that solves a data mining application does the following:

1. generates all child patterns of the zero-length pattern;
2. computes goodness(p) if all of p's immediate subpatterns are good;
3. if good(p) then generate all child patterns of p;
4. applies 2 and 3 repeatedly until there are no more patterns to be considered.

Because the zero-length pattern is always good and the only immediate subpatterns of its children is the zero-length pattern itself, the computation starts on all its children, which are all length 1 patterns. After these patterns are computed, good patterns generate their child sets. Not all of these new patterns will be computed-only those whose every immediate subpattern is good will be.

2.4 Mapping data mining application to DAG.

We propose to use a directed acyclic graph (dag) structure called exploration dag (E-dag, for short) to characterize pattern lattice data mining applications. We first describe how to map a data mining application to an E-dag.

The E-dag constructed for a data mining application has as many vertices as the number of all possible patterns (including the zero-length pattern). Each vertex is labeled with a pattern and no two vertices are labeled with the same pattern. Hence there is a one-to-one relation between the set of vertices of the E-dag and the set of all possible patterns. Therefore, we refer to a vertex and the pattern it is labeled with interchangeably.

There is an incident edge on a pattern p from each immediate subpattern of p. All patterns except the zero-length pattern have at least one incident edge on them. The zero-length pattern has an outgoing edge to each pattern of length 1. Figure 1 (Figure 2, Figure 3, respectively) shows an E-dag mapped from a simple sequence pattern discovery (association rule mining, classification rule mining) application.

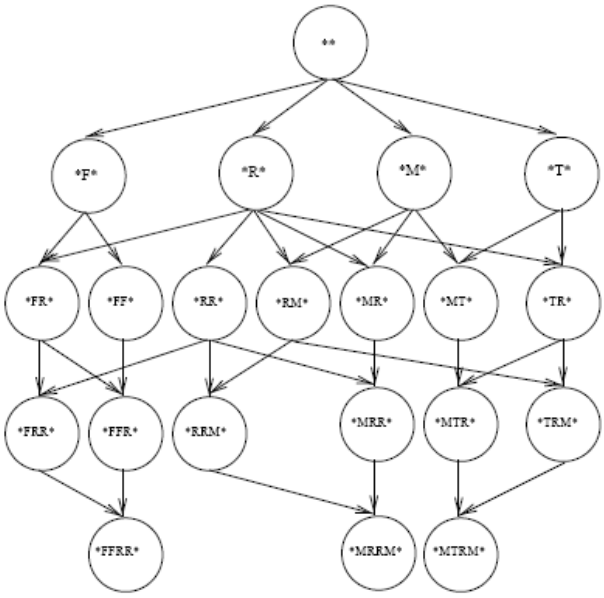

Figure 1: A complete E-DAG for a sequence pattern discovery application on sequences FFRR, MRRM, and MTRM.

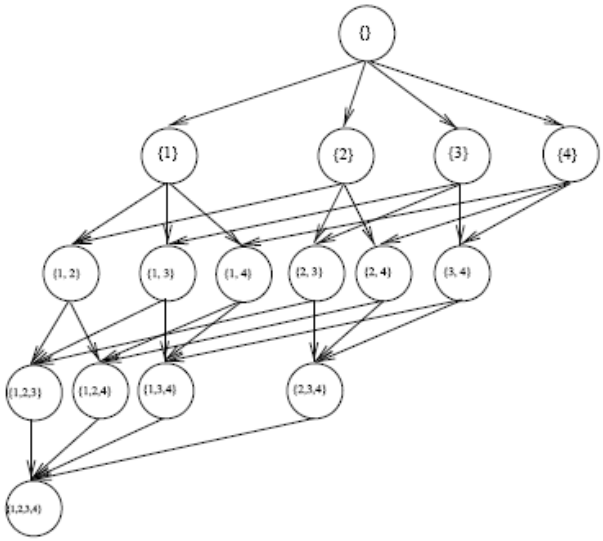

Figure 2: A complete E-DAG for an association rule mining application on the set of items {1, 2, 3, 4}.

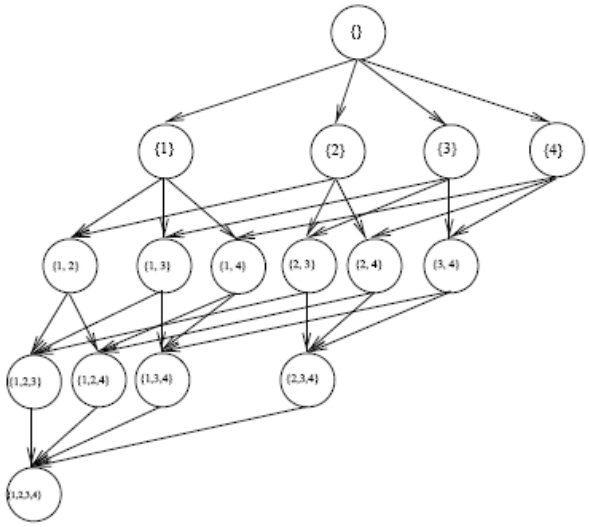

Figure 3: A complete E-DAG for a classification rule mining application on a simple database with attributes A (possible values a1 and a2), and B (possible values b1, b2, and b3).

2.5 Data mining task parallel decomposition

Definition 2.8 Let P be a set. An equivalence relation on P is a binary relation Ί such that for all X, Y, ZΞP, the relation is:

1) Reflexive: XΊ X.
2) Symmetric XΊ Y implies YΊ X.
3) Transitive: XΊ Y and YΊ Z, implies XΊ Z.

The equivalence relation partitions the set P into disjoint subsets called equivalence classes. The equivalence class of an element XΞP is given as [X]={YΞP| XΊ Y}.

Define a function p: P (I)a P(I) where p(X; k) = X[1:k], the k length prefix of X. Define an equivalence relation q_k on the lattice P(I) as follows: " X, Y ΞP(I), XΊ q_k YY p(X, k) = p(Y, k). That is, two itemsets are in the same class if they share a common k length prefix. We therefore call q_k a prefix-based equivalence relation.

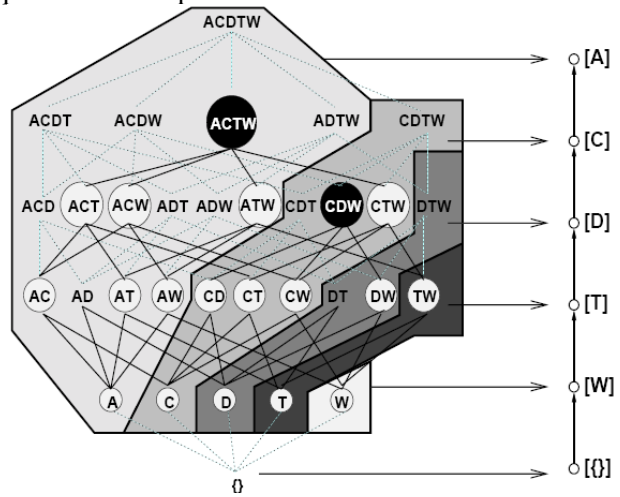

Figure 4: Equivalence Classes of P(I) Induced by q_1

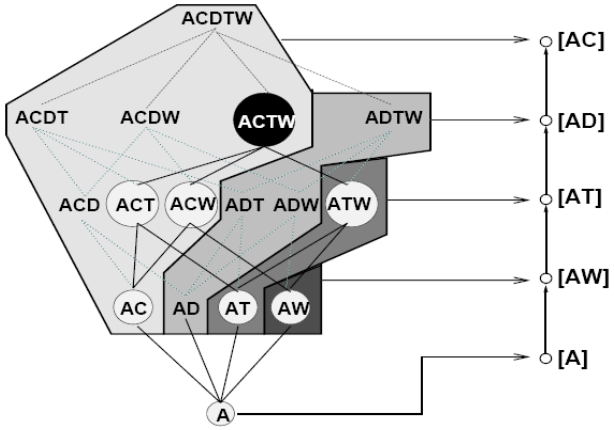

Figure 5: Equivalence Classes of $[A]_{q_1}$ Induced by q_2

Figure 4 shows the lattice induced by the equivalence relation 1 on P(I), where we collapse all itemsets with a common 1 length prefix into an equivalence class. The resulting set or lattice of equivalence classes is {[A], [C], [D], [T], [W]}.

Because the computation nodes in grid are supercomputer, cluster or PC, their computing capability is different [8, 9, 10]. According to each existing computing power of grid, the broker can decompose a data mining task to resources available. Figure 5 shows the decomposition results according to different computing resources.

3 Optimal Implementation of Parallel Tasks

The implementation of the parallel data mining generates the correct result for any data mining application because it faithfully implements a parallel E-dag traversal. But what would be an optimal implementation of a parallel E-dag traversal on Grid? Would an optimal implementation have an equivalent execution as the optimal sequential program on any input? In the Grid environment, with the presence of complicated communication and synchronization costs, the first question is unanswerable in general. And the answer to the second question is no. So what is our approach to optimality? In order to answer this question, we first introduce a tree structure that is closely related to the E-dag structure. The tree structure is called exploration tree (E-tree, for short).

3.1 Exploration Tree

An E-tree is a tree transformed from an E-dag. The transformation is very simple. If we make edges from all patterns to their non-child immediate superpatterns dashed, all the vertices in the E-dag and

all the solid edges constitute the E-tree. The root of the E-tree is the zero-length pattern. Figure 6, Figure 7, and Figure 8 are E-trees transformed from the corresponding E-dags.

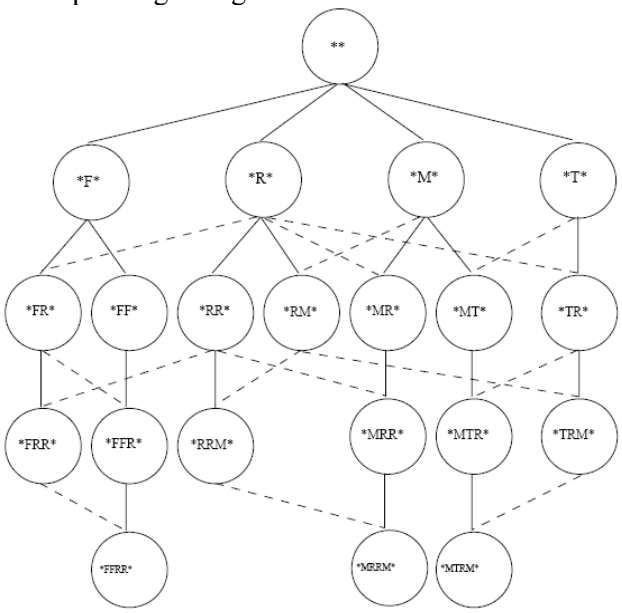

Figure 6: An E-tree for a sequence pattern discovery application on sequences FFRR, MRRM, and MTRM.

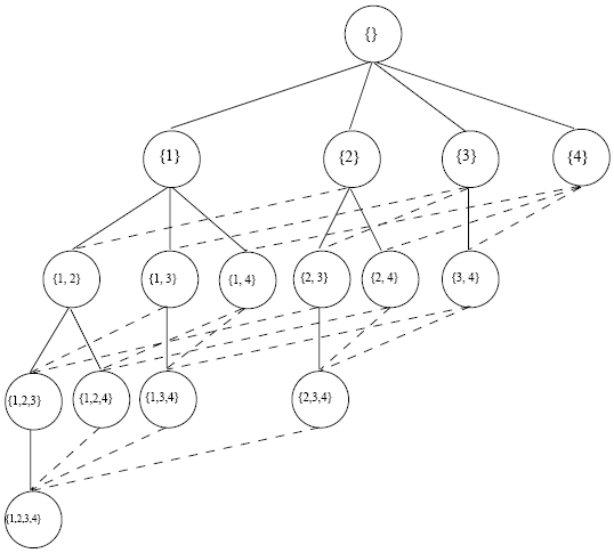

Figure 7: An E-tree for an association rule mining application on the set of items {1, 2, 3, 4}.

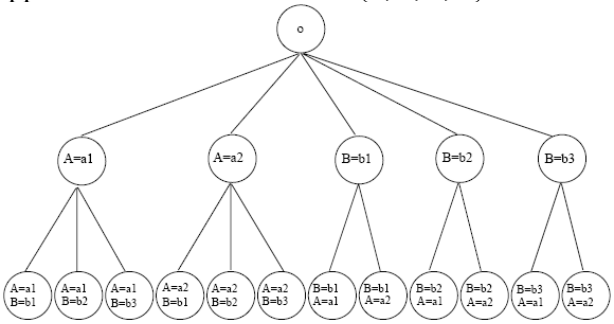

Figure 8: An E-tree for a classification rule mining application on a simple database with attributes A (possible values a1 and a2), and B (possible values b1, b2, and b3).

The optimal implementation of the parallel data mining is a combination of PLED (parallel E-dag traversal) and PLET (parallel E-tree traversal).

We observe that one can benefit more from the pruning due to a non-good subpattern in early stages of an E-dag traversal than in later stages. One possibility is that the optimal program will start with PLED, in the middle of which, will switch to PLET. When to switch is the crucial question in this scheme. The answer will depend on the particular environment on which the program is running and the data mining application itself.

Another possibility for the optimal program is a hybrid from PLED and PLET, in which, when a visiting worker becomes free, it visits the non-pruned node as high as possible in the E-tree. If the result is not good, then it prunes all its superpatterns. This algorithm would be optimal under the assumptions of free network bandwidth and free access to shared storage.

4 Conclusion

We proposed the E-dag framework for finding pattern lattices based on analysis of computation models of three classes of data mining problems. We present the framework that parallelize data mining tasks in Grid environment and how to optimize the parallel framework. E-dag construction and traversal can be done efficiently in parallel. However, network latency can slow down parallel E-dag traversal if there is a fair amount of interprocess communication. In an E-tree traversal, much communication is eliminated by giving up some pruning opportunities. We observe that an optimal parallel program is some form of combination of E-dag and E-tree traversal.

References:

[1] D. Talia and M. Cannataro, Knowledge grid: An architecture for distributed knowledge discovery, Communications of the ACM, 2002.

[2] I. Foster and C. Kasselman, The Grid: blueprint for a future infrastructure. Morgan Kaufman, 1999.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, The Data Grid: towards an architecture for the distributed management and analysis of large scientific datasets. J. of Network and Comp. Appl., (23):187–200, 2001.

[4] Kun Gao, Kexiong Chen, Meiqun Liu, Jiaxun Chen, Rough Set Based Data Mining Tasks

Scheduling on Knowledge Grid, Lecture Notes in Computer Science, Volume 3528, May 2005, Pages 150 - 155

[5] Kun Gao, Youquan Ji, Meiqun Liu, Jiaxun Chen, Rough Set Based Computation Times Estimation on Knowledge Grid, Lecture Notes in Computer Science, Volume 3470, July 2005, Pages 557 – 566

[6] D. Talia, "Parallelism in Knowledge Discovery Techniques", Proc. 6th Int. Conference on Applied Parallel Computing, Helsinki, LNCS 2367, pp. 127-136, June 2002.

[7] M. Cannataro, P. K. Srimani and D. Talia, Parallel Data Intensive Computing in Scientific and Commercial Applications, Parallel Computing, Vol. 28, No. 5, pp. 673-704, May 2002.

[8] D. B. Skillikorn. Strategies for parallel data mining. IEEE Concurrency, 7, 1999.

[9] Cannataro M., Clusters and Grids for Distributed and Parallel Knowledge Discovery, LNCS 1823, pp. 708-716, Springer, 2000.

[10] Cannataro M., D. Talia, Parallel and Distributed Knowledge Discovery on the GRID: A Reference Architecture, 4th Int. Conference on Algorithms and Architecture for Parallel Processing (ICA3PP 2000), Hong Kong, Dec. 11-13, 2000, pp. 662-673. World Scientific, 2000.