J Supercomput (2010) 52: 171–198
DOI 10.1007/s11227-009-0286-5

Performance-based data distribution for data mining applications on grid computing environments

Wen-Chung Shih · Chao-Tung Yang · Shian-Shyong Tseng

Published online: 26 March 2009
© Springer Science+Business Media, LLC 2009

Abstract Effective data distribution techniques can significantly reduce the total execution time of a program on grid computing environments, especially for data mining applications. In this paper, we describe a linear programming formulation for the data distribution problem on grids. Furthermore, a heuristic method, named Heuristic Data Distribution Scheme (HDDS), is proposed to solve this problem. We implement two types of data mining applications, Association Rule Mining and Decision Tree Construction, and conduct experiments on grid testbeds. Experimental results show that data mining programs using the proposed HDDS to distribute data could execute more efficiently than traditional schemes could.

Keywords Heuristic data distribution scheme · Data mining · Grid computing · MPI

W.-C. Shih · S.-S. Tseng
Department of Information Science and Applications, Asia University, Taichung 41354, Taiwan, ROC

W.-C. Shih
e-mail: wjshih@asia.edu.tw

C.-T. Yang (✉)
High-Performance Computing Laboratory, Department of Computer Science, Tunghai University, Taichung 40704, Taiwan, ROC
e-mail: ctyang@thu.edu.tw

S.-S. Tseng
Department of Computer and Information Science, National Chiao Tung University, Hsinchu 30010, Taiwan, ROC
e-mail: sstseng@cs.nctu.edu.tw

1 Introduction

As inexpensive personal computers and Internet access become available, much attention has been directed to grid computing [4, 7, 24, 30, 31, 36, 41, 42, 45]. The basic goal of grid computing is to share the computing and storage resources all over the world via wide area networks. In this way, computation jobs can be distributed to idle computers far away, probably in other countries. Moreover, remote data can be accessed for large-scale analysis.

Data mining is an emerging technology which is applied to many fields including bioinformatics, e-business, etc. Basically, data mining is a data-intensive application, and it is time-consuming to execute data mining programs. In the past decade, a lot of efforts have been devoted to design efficient data mining algorithms [12, 14, 29]. Furthermore, the parallelization of these algorithms has also been investigated. However, the previous work is focused on traditional parallel architectures, such as multiprocessors, homogeneous clusters, heterogeneous clusters, etc. Little work is conducted on grid computing environments.

Traditional parallel data mining work assumes data is partitioned and transmitted to the computing nodes in advance. However, it is also usually the case in which a large database is generated and stored in some station [1]. Therefore, it is important to efficiently partition and distribute the data to other nodes for parallel computation.

In this paper, we focus on the problem of data distribution for data mining applications on grids. It is formulated as a linear programming problem, and a heuristic algorithm is proposed to solve it. The proposed scheme is applied to two kinds of data mining applications, Association Rule Mining and Decision Tree Construction. We implement the application with MPI directives, and execute them on grid testbeds, across three schools. Experimental results show that effective data partition can significantly reduce the total response time.

Our major contributions can be summarized as follows. First, we extend the linear programming formulation of this problem by considering memory constraints to be suitable for data mining applications. Second, this paper proposes a new heuristic algorithm to solve this problem. Finally, we implement the algorithm and apply it to two types of data mining applications on our grid testbeds. Consequently, experimental results show the obvious effectiveness of our approach. This paper enhances our previous work [48] by incorporating a new case study, Decision Tree Construction.

The rest of this paper is organized as follows. Section 2 reviews the related background. In Sect. 3, we describe the system model and formulate the data partition problem. Next, our heuristic scheme is proposed in Sect. 4. Then Sect. 5 shows the experimental results on our grid testbed. Finally, we conclude this paper in Sect. 6.

2 Background review and related work

In this section, we review related background for our research. First, background knowledge about data mining and grid computing is reviewed. Then we present the literature about load distribution and data mining on grids.

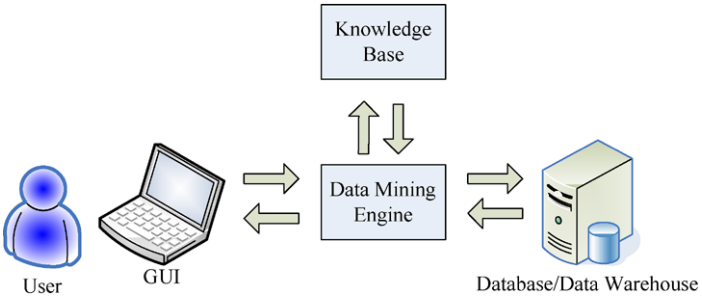


Fig. 1 Overview of a typical data mining system

2.1 Data mining

Due to the advances of information technologies, a tremendous amount of data has been generated by various kinds of applications. This is called the data explosion problem. Moreover, we need to extract knowledge or patterns from these data. Therefore, automatic tools for data processing and analysis are demanding.

Data mining, or known as knowledge discovery, is to acquire interesting knowledge from large-scale databases [26]. A typical architecture of a data mining system is illustrated in Fig. 1. In this architecture, the data mining engine utilizes various kinds of techniques to mine patterns from databases. These techniques include association rule mining, classification, cluster analysis, etc.

The objective of association rule mining is to discover correlation relationships among a set of items. The well-known application of association rule mining is market basket analysis. This technique can extract customer buying behaviors by discovering what items they buy together. The managers of shops can place the associated items at the neighboring shelf to raise their probability of purchasing. For example, milk and bread are frequently bought together.

The formulation of association rule mining problem is described as follows [3]. Let $I = \{I_1, I_2, I_3, \ldots, I_m\}$ be a set of items, and D a database of transactions. Each transaction in D is a subset of I. An association rule is a rule of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$ and $A \cap B = \{\}$. The well-known algorithm for finding association rules in large transaction databases is a priori. It utilizes the a priori property to reduce the search space.

Classification is a kind of data analysis technique which can extract models from categorical data. For example, we can build a classification model to classify animals according a small training database. Classification can be described as a two-phased process [26]. In the training phase, a model is constructed by analyzing a predetermined set of small dataset. Then this model is utilized to conduct classification in the testing phase. The common methods of classification include decision tree induction, k-nearest neighbor classifiers, case-based reasoning, genetic algorithms, etc.

Typical data mining has two goals of prediction and description [43]. In the former field, some people argue that the classification of data is the most important model of data mining and the most commonly applied technique [9]. Among classification algorithms, the decision tree shown in Fig. 2 is probably the most popular and practical

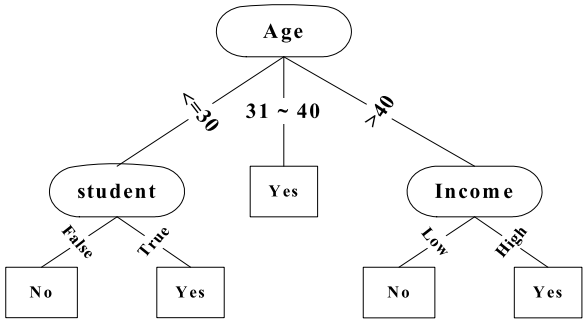


Fig. 2 A decision tree model for buying computers

method. A decision tree is composed of three types of nodes: the root node, the intermediate node, and the leaf node. Generally speaking, it is simple to be interpreted by human [34], also straightforwardly be converted into SQL statements that can be used to access databases efficiently [27].

A classification problem has an input dataset called the training data set. And the dataset consists of a number of records, and each consists of several fields called attributes. Attributes can be continuous, coming from an ordered domain, or categorical, coming from an unordered domain. One of the attributes, called the class attribute, indicates the class to which record belongs. A decision tree is built on two phases: a construction phase and a prune phase [6]. First, an initial tree is built until the leaf nodes belong to a single class only. Second, pruning is done to remove any over fitting to the training data. Typically, time is spent most on the construction phase, so we focus on the tree generation only.

While building the tree, there are three main steps [50] that must be performed for each node at each level of the tree:

1. Evaluate split points for each attribute.
2. Find the winning split-point for a node.
3. Split all attribute lists into two parts, one for each node.

Cluster analysis is a process of grouping objects into groups or clusters, in order that the objects in the same cluster have high similarity, while the objects belonging to different groups have small similarity. There are many methods of clustering, including partitioning methods, hierarchical methods, density-based methods, grid-based methods, model-based methods, etc.

As parallel processing rises, parallel data mining has been well investigated in the past decade; especially, much attention has been directed to parallel association rule mining. A good survey can be found in [49].

2.2 Grid computing

The concept of grid computing is first stated by Foster and Kesselman, and it has emerged as a promising technology [17–21]. However, it is a new field, and its standards and architectures are still being investigated. Conceptually, grid environments are a type of distributed systems. However, it includes many unique characteristics.

First, resources within the grid are shared without centralized controlling and coordination. Users access these resources via distributed protocols and mechanisms. Second, the protocols, standards, and middleware are open. In other words, it is not a proprietary system. Finally, the performance of the grid is a critical issue. Because the resource can be located all over the world interconnected by the Internet, it is necessary for the grid infrastructure to provide a good enough quality of service.

The architecture of the grid involves many standards. The well-known one is the Open Grid Services Architecture (OGSA). OGSA is a common specification, and its objective is to standardize the services used by users and applications of a grid. This standard primarily involves:

- Program-to-program interface and messaging (SOAP)
- Data sharing (XML)
- Workload management (WS-Management)
- Other related specifications

In the OGSA standard, specification for implementing is provided by OGSI. A common implementation of OGSI is Globus Toolkit [46]. The software tools are provided by the Globus Project, and are used to build computational Grids and Grid-based applications. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability.

The Globus Toolkit consists of three components: Resource Management, Information Services, and Data Management. First, a resource management protocol is implemented by the Globus Resource Allocation Manager (GRAM). Second, the Metacomputing Directory Service (MDS) implements an information services protocol. Finally, a data transfer protocol is implemented by GridFTP. These protocols utilize the GSI security protocol to interconnection.

MPI is a message-passing interface library. This standard is based on the discussion in the MPI Forums [32]. Participants include vendors, researchers, academics, software library developers, and users. MPI offers portability, standardization, performance, and functionality [33]. MPICH-G2 [32, 33] is an implementation of the MPI v1.1 standard, but it is Grid-enabled. It uses services of the Globus Toolkit® to interconnect multiple machines, which may include different kinds of architectures. Existing parallelized programs written with MPI directives can be recompiled and executed on the Globus infrastructure [33].

The NWS (Network Weather Service) [35] is a distributed system that periodically monitors and dynamically forecasts the performance of various networking and computational resources over a given time interval. This service utilizes a distributed set of performance sensors (network monitors, CPU monitors, etc.) from which it can gather system status information. It then uses numerical models to generate forecasts of what the conditions will be for a given time period. It also uses mathematical models to forecast each condition and the Mean Absolute Error (MAE) and Mean Square Error (MSE) rates. NWS is a widely used measurement tool for grid environments.

The HPC laboratory of Tunghai University has developed a GUI [47] which can display bandwidth statistics of their grid environments. This tool utilizes API provided by NWS, and depicts real-time bandwidth statistics in web-based graphic user

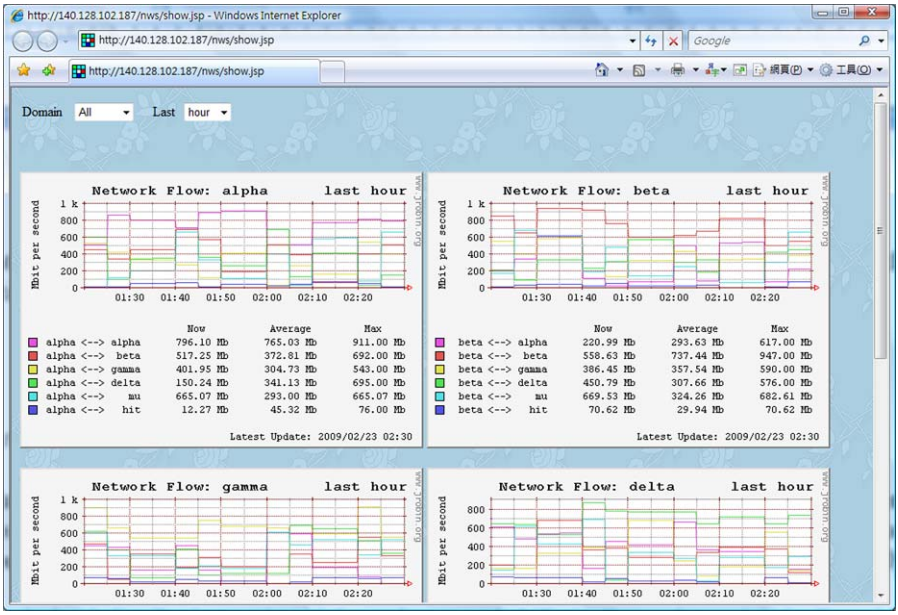


Fig. 3 Bandwidth statistics extracted from NWS information

interface, as shown in Fig. 3. In this paper, these bandwidth statistics will be used to estimate the performance of each grid node.

2.3 Load distribution

Load distribution has been studied for decades. As described in [16], it aims to improve the performance of a distributed system, usually in terms of response time or resource availability, by allocating workload amongst a set of cooperating hosts. This division of system load can take place statically or dynamically:

- Static load distribution assigns jobs to hosts probabilistically or deterministically, without consideration of runtime events. This approach is both simple and effective when the workload can be accurately characterized and where the scheduler is pervasive, in control of all activity, or is at least aware of a consistent background over which it makes its own distribution. Problems arise when the background load is liable to fluctuations, or there are jobs outside the control of the static load distributer.
- Dynamic load distribution is designed to overcome the problems of unknown or uncharacterizable workloads, nonpervasive scheduling and runtime variation (any situation where the availability of hosts, the composition of the workload, or the interaction of human beings can alter resource requirements or availability). Dynamic load distribution systems typically monitor the workload and hosts for any factors that may affect the choice of the most appropriate assignment and distribute jobs accordingly. This very difference between static and dynamic forms of load distribution is the source of the power and interest in dynamic load distribution.

Divisible load theory is a methodology involving the linear and continuous modeling of partitionable computation and communication loads for parallel processing. It adequately represents an important class of problems with applications in parallel and distributed system scheduling, various types of data processing, scientific and engineering computation, and sensor networks. Solutions are surprisingly tractable. Research in this area over the past decade is described.

Divisible Load Theory (DLT) [10, 11, 15, 38] addresses the case where the total workload can be partitioned into any number of independent subjobs. This problem has been discussed in the past decade, and a good review can be found in [8]. However, these previous models did not consider the memory storage constraint, which is critical for storing intermediate data structures in data mining processes. In [13], a data distribution method was proposed for host-client type of applications. Nevertheless, it was not formulated as a linear programming problem. In addition, their method was an analytic technique, and only verified on homogeneous and heterogeneous cluster computing platforms, not for grid environments.

2.4 Data mining on grids

Traditional parallel data mining work assumes data is partitioned and transmitted to the computing nodes in advance. However, it is also usually the case in which a large database is generated and stored in some station [1]. Grid computing has emerged as a computing infrastructure for implementing distributed high-performance applications and solving complex problems. Grid computing is receiving an increasing attention both from the research community and from industry and governments. A lot of researches have been conducted to address the execution of data mining applications on grid environments [12, 14, 29]. In [12], the KNOWLEDGE GRID was proposed to integrate data mining techniques and grid technologies. In the KNOWLEDGE GRID architecture data mining tools are integrated with generic and data grid mechanisms and services. Thus, the KNOWLEDGE GRID can be exploited to perform data mining on very large data sets available over grids, to make scientific discoveries, improve industrial processes and organization models, and uncover business valuable information. However, the problem of partitioning and distributing the data to other nodes for parallel computation has not been addressed. In this paper, we focus on the problem of data distribution for data mining applications on grids.

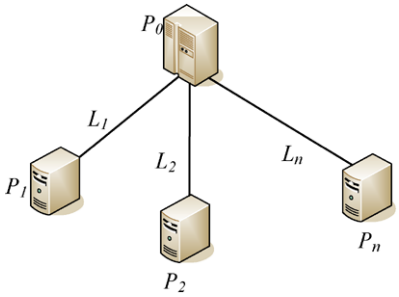
3 Problem formulation

In this section, the master/slave model for grid computing is described. Then we present the linear programming formulation of the data distribution problem.

3.1 The system and cost model

Our system model and cost model are extended from the framework in [8]. The master/slave model for a grid is represented by a star graph $G = \{P_0, P_1, \ldots, P_n\}$, as shown in Fig. 4. In this graph, P_0 is the master node and the other n nodes, $P_1, \ldots, P_n$,

Fig. 4 The system model



are slave nodes. In addition, there is a virtual communication link L_i connecting the master node and the slave node P_i.

In our cost model, each node P_i is associated with a computing capacity C_i, a memory capacity M_i, and a disk storage capacity D_i. Furthermore, each link L_i is also associated with a transmission capacity T_i. In our linear cost model, it takes $W \times C_i$ time units for the slave node P_i to conduct computation on W units of data. Besides, it takes $W \times T_i$ time units for link L_i to transmit W units of data. In this model, we assume that the master can only communicate with one slave node at the same time.

3.2 Linear programming formulation

The goal is to minimize the total response time of a given workload W on the system. The Data Distribution Problem (DDP) can be formulated as a linear program as follows

Minimize T_{finish}

Subject to

$$W_i \geq 0, \quad 1 \leq i \leq n \tag{1}$$

$$\sum_{i=1}^{n} W_i = W \tag{2}$$

$$W \cdot s \leq D_0 \tag{3}$$

$$W_i \cdot s \leq D_i, \quad 1 \leq i \leq n \tag{4}$$

$$DS(W_i) \leq M_i, \quad 1 \leq i \leq n \tag{5}$$

$$W_1 \cdot T_1 + W_1 \cdot C_1 \leq T_{\text{finish}} \tag{6}$$

$$\sum_{j=1}^{i} W_j \cdot T_j + W_1 \cdot C_1 \leq T_{\text{finish}}, \quad 2 \leq i \leq n \tag{7}$$

where

- T_{finish} is the response time for the master node to finish executing the data mining application program. For simplicity, we assume T_{finish} is a linear function.

- W_i is the work load dispatched to the slave P_i.
- s is the size of one unit data.
- DS is a linear function which returns the size of the intermediate data structure during data mining process. This intermediate data structure is generated during the process of local computation. For example, in association rule mining, the intermediate data structure could be a hash tree for frequent itemsets. For simplicity, we assume DS is a linear function.

3.3 An example

We describe an example to clarify the model and the problem formulation. The master/slave model for the example grid consists of four nodes, P_0, P_1, P_2, and P_3, as shown in Fig. 5. In this graph, P_0 is the master node and the other three nodes are slave nodes. In addition, L_1, L_2, and L_3 connect the master node to P_1, P_2, and P_3, respectively. The total workload of this example database contains 256 transactions. The related attributes are listed in Table 1.

The objective is to minimize the total response time of a given workload of 256 transactions on the system. The data mining application is association rule mining. We try to partition the database into three subsets, and transmit them to the three slave nodes, respectively. However, inappropriate partition size or transmission order could affect the total execution time. Therefore, a good distribution scheme is essential.

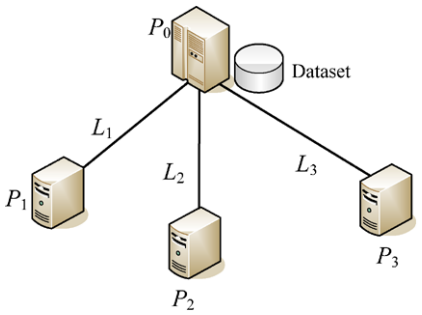


Fig. 5 An example grid

Table 1 Parameters in the example grid

Attribute	Node			
	P_0 (Master)	P_1	P_2	P_3
C_i	1.5 GHz	2.0 GHz	1.0 GHz	1.5 GHz
M_i	1 GB	512 MB	512 MB	256 MB
D_i	80 GB	80 GB	40 GB	40 GB
L_i		1 Mbps	256 Kbps	512 Kbps

4 HDDS (heuristic data distribution scheme)

The proposed heuristic algorithm is based on the performance of each slave node and each link to distribute the corresponding workload. In this section, the concept of performance ratio is explained first. Then we present the heuristics which the algorithm is based on. Finally, the algorithm is formally described and illustrated by an example.

4.1 Performance ratio

We propose to partition the workload according to the performance ratio of all slave nodes. Therefore, the effectiveness of this approach depends on the accuracy of estimating the performance ratio. To estimate the performance of each slave node, we define a Performance Function (*PF*) for a slave node j as

$$PF_j(V_1, V_2, \ldots, V_M) \tag{8}$$

where V_i, $1 \leq i \leq M$, is a parameter of the performance function. In more detail, the parameters could include CPU speed, networking bandwidth, memory capacity, etc. In this paper, our *PF* for node j is defined as

$$PF_j = w_1 \times \frac{1/t_j}{\sum_{\forall node_i \in S} 1/t_i} + w_2 \times \frac{B_j}{\sum_{\forall node_i \in S} B_i} \tag{9}$$

where

- S is the set of all slave nodes.
- t_i is the execution time (s) of node i for some data mining application program, such as association rule mining.
- B_i is the bandwidth (Mbps) between node i and the master node.
- w_1 is the weight of the first term.
- w_2 is the weight of the second term ($w_2 = 1 - w_1$).

The Performance Ratio (*PR*) is defined to be the ratio of all performance functions. For instance, assume the *PF*s of three nodes are 1/2, 1/3, and 1/4, respectively. Then *PR* is $1/2 : 1/3 : 1/4$; i.e., the *PR* of the three nodes is $6 : 4 : 3$. In other words, if there are 13 transactions to be processed, 6 transactions will be assigned to the first node, 4 transactions will be assigned to the second node, and 3 transactions will be assigned to the last one.

4.2 Performance-based heuristics

Our algorithm is based on two heuristics to dispatch workload to slave nodes.

1. The total workload is divided in n chunks according to the *PR* of the n slave nodes.
2. Send the data chunk to the node with faster network bandwidth first. The network bandwidth is estimated by $\frac{B_j}{\sum_{\forall node_i \in S} B_i}$.

In this paper, B_j is obtained from NWS (Network Weather Service) statistics [35]. Specifically, our network bandwidth estimation is extracted directed from [47].

4.3 Algorithm

Our algorithm is also a master/slave type of application. In the MASTER module, the total dataset is divided according to the *PR* of slaves, and the subdatasets are transmitted accordingly. In the SLAVE module, the subdataset is computed. The algorithm of our approach is described as follows.

Module MASTER

```
1. Initialization
2. Calculate performance ratio of all slave nodes
3. Partition the total data according to the PR
4. Get network bandwidth B_i of the link to node i
5. Send data to slaves in nonincreasing order of B_i
6. //Master could does its own computation work here
7. Gather results from all slave nodes
8. Print the results
9. Finalization
```

Module SLAVE

```
1. Receive data from the master node
2. Conduct data mining computation on its local database
3. Send the result to the master
```

Without loss of generality, we assume the master node does not participate in computation in our algorithm. However, the algorithm can be modified to utilize the computing power of the master node by removing the comment notation (//) in line 6.

4.4 An example

To clarify our algorithm, we use the example in Fig. 4 to go through the MASTER algorithm.

1. The master conducts initialization work.
2. The *PR* is obtained by (9), and is assumed to be $6:4:3$.
3. The dataset is partitioned by $6:4:3$.
4. The ratio of B_i values of three links is obtained from the NWS information, and is assumed to be $4:1:2$.
5. The data chunks are sent in the order: P_1, P_3, and then P_2, as shown in Fig. 6.
6. The master collects results from slaves.
7. The master outputs the answer.
8. The master conducts finalization work.

For comparison, the partition sizes by different partition schemes are listed in Table 2. In this table, "EQ" means to distribute the workload to slaves equally, and "CPU-Weighted" means to distribute the workload to slaves according to the ratio of CPU speed values of slaves. And, HDDS is our scheme.

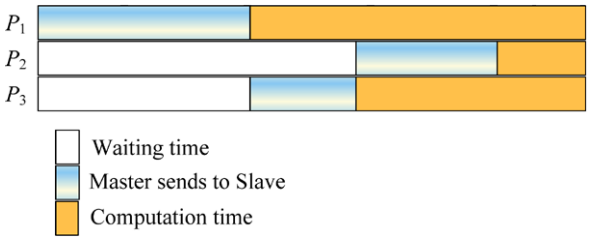


Fig. 6 Example time line

Table 2 The partition sizes by different partition schemes

Scheme	Node			
	P_0 (Master)	P_1	P_2	P_3
EQ	256	85	85	86
CPU-Weighted	256	114	57	85
HDDS	256	118	79	59

4.5 A grid-based data mining architecture

We can set up a local Grid instead of global, because the basic idea is sharing information, not necessarily depend on the scale. The purpose is to avoid wasting resource. The main advantage is in the resource integration. For example, through Grid computing technology, we can find the idle computers on the Internet and divide task into these computers. Because of the efficient usage of resources, we can spend only one hour in finishing the tasks that originally needs 10 hours to complete by one computer.

Since, Grid computing has caught a lot of attention. Some applications on Grid-based data mining infrastructure have been proposed, such as the Knowledge Grid (K-Grid) [37] and NASA's Information Power Grid (IPG) [28]. Both are the input on the data mining algorithms work to extract new knowledge. The objective is to build the next generation computing infrastructure providing intensive analysis and integration over World Wide Web. In addition, the DataGrid Project developed by European Union is focused on core Grid data services, like data access and metadata access. It can process the data sets from hundreds of TeraBytes to PetaBytes [5].

We present the "Top-Down" closed Grid architecture for data mining, because it is normally used for e-commerce. Finding the best split point and performing the split are the main tasks in data mining. The former is a very important as well as a very tedious job, specifically the continuous attribute, for which many algorithms use gini index [25] to get the split position. The split point is a mid-point between every two sorted consecutive attribute values. Therefore, it requires an ongoing calculation based on two adjacent values. When the data amount is large, using Grid technology to split up the workload is very useful. So, our main task in our Grid architecture is to calculate the split point.

In general, basic Grid technology services include security service, scheduling, data service, database service, user service, application management service, autonomy and monitoring service, information service, composition service, and message

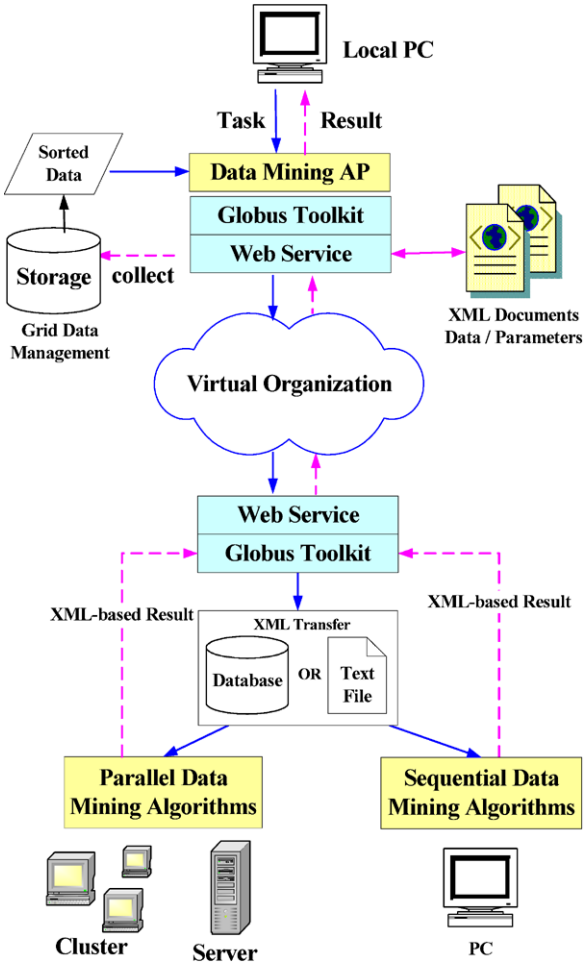


Fig. 7 Grid-based data mining architecture

service. Currently, Grid architecture is mature, and we will not elaborate the various service details here. Our architecture is shown in Fig. 7; we just use Open Source Globus Toolkit as Grid middleware, it provides three functions, that is, resource management, data management and information services. They are built in GSI (Grid Security Infrastructure) [22]. The data exchange protocol, allocation, control and security in the Grid environment are all dependent on this middleware. Web Service provides XML and HTTP services. It will be a corresponding suite of XML schema describing the object and services associated with the application [23]. The particular application is built on the top of them.

Users submit instructions to the data mining application and through Globus find the suitable resources. Workload is estimated according to the resource efficiency. Following, attribute data and description will be specified in XML. Note the attribute data maybe have been sorted by task type. All the work is processed through Web Services. Those XML documents not only contain attribute data, but also describe

the task including some parameters for computing split point. Finally, Globus sends the resulting documents to the client side.

In order to fully utilize the network resources, no matter using PCs or servers, there may be different application functions. But they all use the same Grid mechanism while the differences lie at the algorithm of parallel or sequential process. The client resource is processed as follows: through Web Services, XML files are transformed into database table data or text file by client attributes. If clients are using the parallel mechanism, such as servers or Clusters, the software application will execute with parallel data mining algorithms. On the other hand, if it is a PC, it will compute by the sequential algorithms.

The calculation results are then transformed into XML-based format which do not include original attribute data because the data mining application just needs the candidate split point values. The same process, send back to the local PC by the Grid mechanism. The local PC receives the result, via Web Service, and then stores into the storage for data management. Through these repeating processes of resource search and allocation task, a large and various data mining will proceed faster than the single resource process.

Through collaboration, it can extend the application scope of data mining using Grid technologies. Parallel and sequential mechanisms both can be used. With the enhancement of calculation speed and Grid-related technologies, programmers do not need to modify too many codes to transfer the original system onto the Grid environment. Some algorithms are very useful and accurate, but the fact that during calculation all the tasks need to be fit into a single site is requiring further enhancement or replacement with the increasing number of data. Therefore, if data can be distributed to every computer in the world and participates in the Grid with each host processing part of the data and sending back response result. There is no need to modify the algorithm in order to reach the same objective with enhanced speed and performance and lower cost. Of course, the supercomputers or Clusters resources can also be included in the architecture.

5 Experimental results

To verify our approach, a grid testbed is built, and one type of data mining application programs is implemented with MPI (Message Passing Interface) to be executed on this testbed. To begin with, our grid environment is illustrated, and terminologies for our programs are described. Next, performance of our scheme is compared with that of other schemes on this grid, with respect to association rule mining. Based on experimental results in this section, we could conclude that our HDDS got performance improvement on previous schemes for most cases.

5.1 Data mining application 1: association rule mining

In this section, we implement the a priori algorithm, and apply our HDDS to conduct data distribution. Specifically, the parallelized version of a priori we adopt is Count Distribution (CD) [2].

Fig. 8 The grid testbed

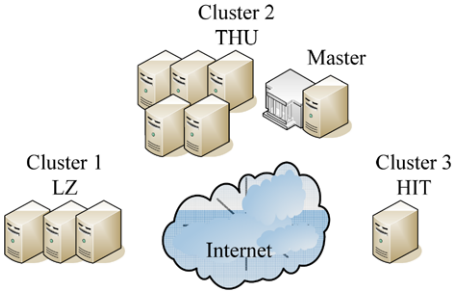


Table 3 Hardware configuration

	Host	CPU type	CPU speed	RAM
Master				
	gamma3	Intel Pentium^{TM} III	866 MHz	512 MB
Cluster 1: LZ (Li Zen High School)				
	lz01	Celeron^{TM}	900 MHz	256 MB
	lz02	Celeron^{TM}	900 MHz	256 MB
	lz03	Celeron^{TM}	900 MHz	256 MB
Cluster 2: THU (Tunghai University)				
	beta1	Celeron^{TM}	1.7 GHz	512 MB
	beta2	Celeron^{TM}	1.7 GHz	512 MB
	beta3	Celeron^{TM}	1.7 GHz	256 MB
	gamma1	Intel Pentium^{TM} III	866 MHz	512 MB
	gamma2	Intel Pentium^{TM} III	866 MHz	512 MB
Cluster 3: HIT (Hsiuping Institute of Technology)				
	gridhit3	Intel Pentium^{TM} 4	2.8 GHz	512 MB

Hardware configuration and terminology We have built a grid testbed, which consists of one master and three domains. This testbed is illustrated in Fig. 8. We have built this grid testbed by the following middleware:

- Globus Toolkit 4.0.6 [46]
- Mpich library 1.2.7 [32, 33]

The hardware configuration of this testbed is shown in Table 3.

In this experiment, the performance function and the performance ratio are the same as those defined in Sect. 3. The assumption behind the assignment is that each type of application has its suitable weight assignment. For Association Rule Mining, we have experimented on different weight combination, and the ratio of 7/3 got best performance for w_1/w_2. Therefore, w_1 is assigned as 0.7 and w_2 is assigned as 0.3. In the future work, the impact of weight assignment will be investigated and rules will be worked out. Furthermore, T_i for node i is obtained by executing a priori, a representative algorithm for association rule mining, for input size 1000 transactions, while B_i for node i is obtained by NWS statistics [35, 47]. The resulting performance

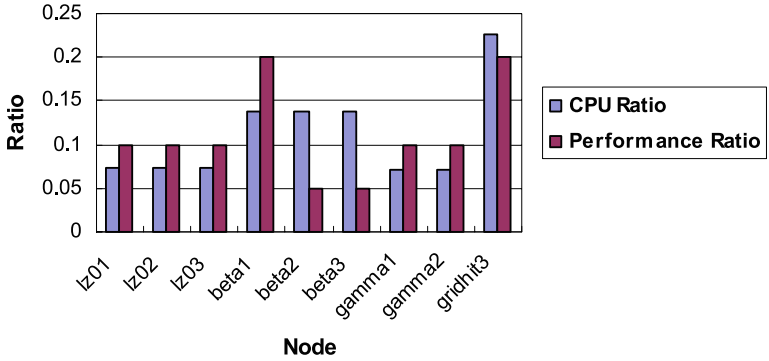


Fig. 9 Performance ratio of 9 slave nodes for our grid testbed

Table 4 Description of our implementation for all programs

AP	Name	Scheme	Description
Association rule mining (count distribution)	cd_eq	EQ	Equal data partition
	cd_cpu	CPU_Weighted	CPU-Weighted data partition
	cd_hdds	HDDS	Our HDDS data partition

Table 5 Description of our dataset

Dataset	Number of transactions	Average transaction length	Number of items
D10KT5I10	10,000	5	10
D50KT5I10	50,000	5	10
D100KT5I10	100,000	5	10

ratio is shown in Fig. 9. For example, node 4 and node 5 have the same CPU speed. However, our HDDS assigns higher PR to node 4 because of its higher network bandwidth.

We have implemented one kind of data mining application programs in C language, with message passing interface (MPI) directives for parallelizing code segments to be processed by multiple CPUs. For readability of experimental results, the description of our implementation for all programs is listed in Table 4. In this table, "cd_eq" means to distribute the workload to slaves equally, and "cd_cpu" means to distribute the workload to slaves according to the ratio of CPU speed values of slaves. And, cd_hdds is our scheme. Our datasets are generated by the tool as in [3]. The parameters of the synthetic datasets are described in Table 5.

Relative performance for different dataset sizes First, execution time on the grid for the three schemes is investigated. Figure 10 illustrates execution time of cd_eq, cd_cpu, and our cd_hdds, with input size 10 K, 50 K, and 100 K transactions, respectively. Experimental results show that our scheme got better performance than cd_eq

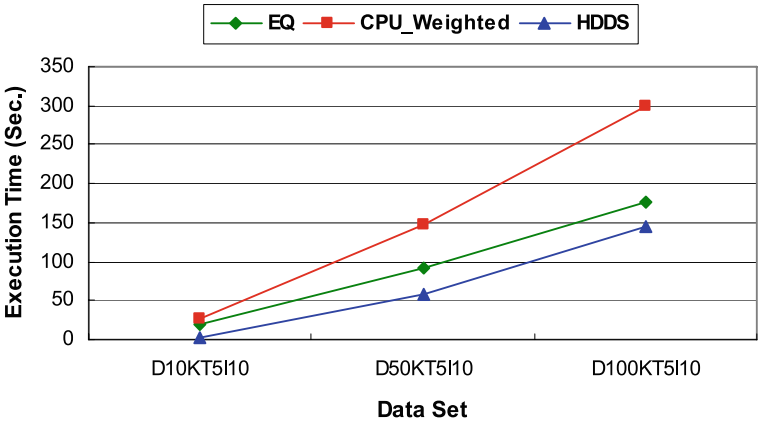


Fig. 10 Performance of data partition schemes for different datasets

and cd_cpu. In this case, our scheme for input size 100 K transactions got 18% and 52% performance improvement over cd_eq and cd_cpu, respectively.

From this experiment, we can see the significant influence of partition schemes on the total response time. In grid environments, network bandwidth is an important criterion to evaluate the performance of a slave node. Cd_eq and cd_cpu are static data partition schemes. Therefore, they can not adapt to the practical network status. When communication cost becomes a major factor, our HDDS scheme would be well adaptive to the network environment.

Moreover, the reason why cd_cpu got the worst performance can be contributed to the inappropriate estimation of node performance. In grid computing environments, CPU speed is not the only factor to determine the node performance. A node with the fastest CPU is not necessary the node with optimal performance. This has been illustrated in Fig. 10.

Speedup In order to see how well our HDDS scheme speeds up, we keep the dataset constant to be D10KT5I10 and vary the number of nodes. Figure 11 shows that the response time of HDDS is decreasing as the number of nodes increases. This means our HDDS can choose available computing power to optimize its execution time. However, the curves of cd_eq and cd_cpu fluctuate as the number of nodes increases.

The effect of transmission order In this experiment, we address the effect of transmission order. As described in Sect. 4.2, our HDDS sends data chunks in the decreasing order of link bandwidths. For comparison, we implement two schemes with different transmission order, cd_inc and cd_rand. In contrast to our HDDS, cd_inc sends data chunks in the increasing order of network bandwidths. On the other hand, cd_rand sends data chunks in the arbitrarily chosen order of network bandwidths. As shown in Fig. 12, HDDS outperforms the other two schemes. The reason might be that our decreasing heuristic can reduce the overall waiting time of all slave nodes.

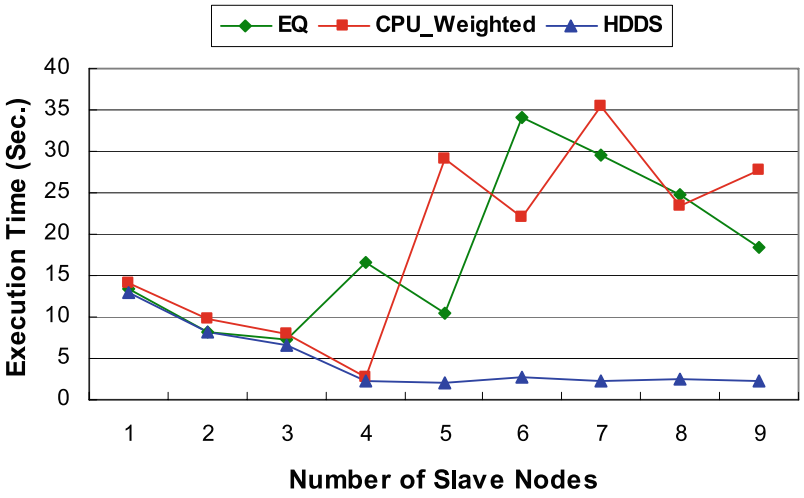


Fig. 11 Speedup performance of data partition schemes

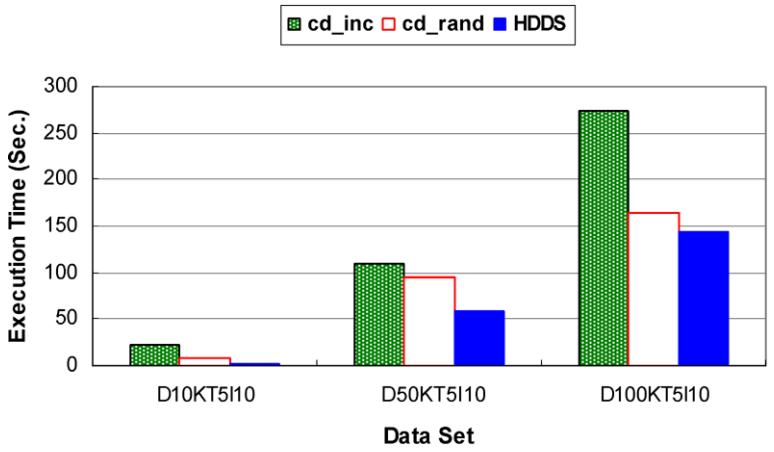


Fig. 12 Performance of data partition schemes for different transmission order

5.2 Data mining application 2: decision tree construction

The SPRINT algorithm We employ a parallel SPRINT (Scalable Parallel Induction of Decision Tree) algorithm that implemented its parallelization on an IBM SP2 [39]. We exploit the same program into Cluster and Grid environments, and try not to modify the originally program of Cluster. Also we want to know SPRINT algorithm whether suitable for Grid computing or not. Then we introduce it briefly in next section and focus on the continuous attribute. Note this algorithm was running SMP architecture, but we run in Cluster architecture. So, it can affect the experimental result, maybe different from the original paper of SPRINT.

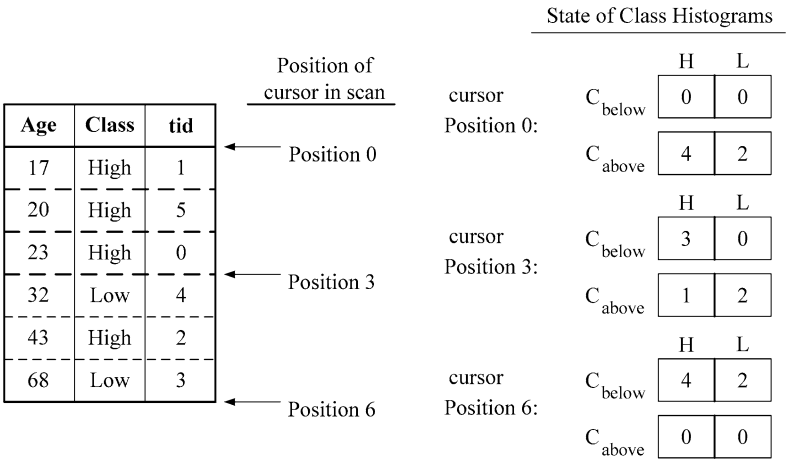


Fig. 13 Evaluating continuous split points

SPRINT algorithm removes all of the memory restrictions, and it is fast and scalable. It is designed for being easily parallelized, allowing many processors to work together to build a single consistent model. The algorithm assumes a share-nothing parallel environment where each processor has private memory and disk. The processors are connected by a communication network and can communicate only by passing messages [39]. In SPRINT program [44], we downloaded that has been written by C++ and MPI, compiler with mpiCC v2.96. It is implementation of the SPRINT parallel synchronous decision tree construction.

For continuous attributes, there are two histograms are associated with each node for splitting. The example is shown in Fig. 13 [39]. All attributes will partition to each own attribute list. The attribute list will average to every processor when process. C_{below} maintains the attribute records that have already been processed, whereas C_{above} for remainder that has not. Both C_{below} and C_{above} have all the necessary information to compute the gini index. For the root node, it is obtained at the time of sorting, the other nodes is obtained when the node is created. Since attribute lists are processed one at a time, memory is required for only one set of such histograms. It is one characteristic of SPRINT algorithm. Note this algorithm needs to sort the continuous attribute first before the put into the attribute list.

For performing the split, it used a hash table that collected the attribute value, class and its record number, namely attribute list. You can refer to Fig. 13. So, there is nothing to which child the record was moved. The retrieved information told us with which child to place the record. Therefore, the memory is less used. If the hash-table is too large for memory, splitting is done in more than one step. The attribute list for the splitting attribute is partitioned according to the attribute record for which the hash table will fit in memory, and the process is repeated for the remainder.

As data placement, the partitioning is achieved by first distributing the records equally among all the processors; it is shown in Fig. 14. Each processor generates its own attribute list partitions in parallel. Also, they can parallelize computing to find the split point for each own record.

Fig. 14 Parallel data placement

Processor 0

Age	Class	rid
17	High	1
20	High	5
23	High	0

Car Type	Class	rid
Family	High	0
Sports	High	1
Sports	High	2

Processor 1

Age	Class	rid
32	Low	4
43	High	2
68	Low	3

Car Type	Class	rid
Family	Low	3
Truck	Low	4
Family	High	5

Table 6 The dataset list

Dataset	Attributes			Record	Size
	Continuous	Categorical	Class		
cov4	4	0	1	581,012	15.7 MB
big4	4	0	1	1,162,024	31.7 MB

In a parallel environment, each processor can work independently for processing 1/N of the total data. Each processor has a separate contiguous section of a global attribute list. C_{below} must initially reflect the class distribution of all sections of an attribute-list assigned to processors of lower rank. C_{above} must likewise initially reflect the class distribution of the local section as well as all sections assigned to processors of higher rank. Once all the attribute list sections of a leaf have been processed, each processor will have what it considers besting the best split for that leaf. The processors then communicate to determine which of the N split points has the lowest cost.

Datasets The dataset we use synthetic datasets with text file. We focus on the continuous attribute computing. We try two different datasets: cov4 and big4, big4 is almost double size of cov4. Both have 4 continuous attributes and 1 class, and their sizes are shown in Table 6. In these synthetic datasets, the range of attributes are as follows: the class ranges from 1 to 7 (integer), attribute 1 ranges from 200 to 3999, attribute 2 ranges from 1 to 500, attribute 3 ranges from 1 to 99, attribute 4 ranges from 0 to 99.

Cluster Our Cluster C149 is a low cost Beowulf-type supercomputer that utilizes multicomputer architecture for parallel computations, as shown in Table 7. It consists of 8 PC-based connected by three 24-port 100 Mbps Ethernet switches with Fast Ethernet interface. Each node has two AMD Athlon MP 2600+ or 2400+ 1 GHz processors. There are total 16 processors. The disks are made a NFS shared local disk. The processors of Cluster connected with a private network. It is a less heterogeneous Cluster architecture.

Table 7 Hardware configuration of Cluster C149

Cluster C149			
No	Host name	Processor (CPU = 16)	Memory
1	amd1*	AMD Athlon MP 2600+ × 2	2048 MB
2	amd2	AMD Athlon MP 2600+ × 2	1024 MB
3	amd3	AMD Athlon MP 2600+ × 2	1024 MB
4	amd4	AMD Athlon MP 2400+ × 2	1024 MB
5	amd5	AMD Athlon MP 2400+ × 2	1024 MB
6	amd6	AMD Athlon MP 2400+ × 2	1024 MB
7	amd7	AMD Athlon MP 2400+ × 2	1024 MB
8	amd8	AMD Athlon MP 2400+ × 2	1024 MB

*Stands for Master node of the cluster, the others are slave nodes

All computers in this Cluster run the RedHat Linux release 8 operating system. Program is developed using C++ language, compiler within MPI library LAM 6.5.8-4 and gcc 3.2.2. The program and dataset are just put in the master node, also compiler, and run here.

Grid Our Grid architecture is implemented on top of Globus Toolkit, name grid-cluster. It has built three Clusters to form a multiple cluster environment. Cluster1 and Cluster2 consist of 4 PCs that each has one master node and three slave nodes. Cluster3 consists of 3 PCs, has one master node and two slave nodes, as shown in Table 8. Each node is interconnected through 3COM 10/100 Fast Ethernet Card to Accton Switch HUB. SGE QMaster daemon is run on the master node of each Cluster, and SGE execute daemon is run to manage and monitor the incoming job and Globus Toolkit v2.4. Each slave node is running SGE execute daemon to execute income job only.

The operating system is RedHat Linux release 9. The parallel application we use is MPICH-G2 v1.25 for message passing. The program and dataset have to be put in the master node of each Cluster and also need to be put into the same directory. But we can run just in Cluster1. Cluster1 is a master of grid-cluster. If we use mpirun over 5 processors, it means that the process will communicate current Cluster with another Cluster. The order of mpirun is Cluster1, Cluster2, and Cluster3. Note the order will affect the result of run. During execution, if mpirun needs to cross the other processor, it will run sequentially by machine file that we have written.

Performance result There are two kinds of environment and two datasets in our experiment. We first examined the original cluster C149, the result is shown in Fig. 15. The turn-around time is the total response time. We found the best performance is 6 processors, because the former 3 CPU of nodes are higher. The latter 5 nodes are lower, so the performance is getting down. There is just a little difference in CPU, but the performance is sensitive. Although the result is not good for increasing processors, it needs more communication as the processor is increased. Fortunately, the execution time is not double when the amount of dataset is double. In the experiment of original SPRINT paper is implemented in IBM SP2, it is all in the SMP architecture with intracommunication. Our Cluster has external communication with each node.

Table 8 Hardware configuration of Grid-cluster

Cluster1*				
No.	IP	Host name	Processor (CPU = 4)	Memory
1	beta1.hpc.csie.thu.edu.tw	beta1*	Intel Celeron 1.7 GHz	512 MB
2	beta2.hpc.csie.thu.edu.tw	beta2	Intel Celeron 1.7 GHz	256 MB
3	beta3.hpc.csie.thu.edu.tw	beta3	Intel Celeron 1.7 GHz	256 MB
4	beta4.hpc.csie.thu.edu.tw	beta4	Intel Celeron 1.7 GHz	256 MB
Cluster2				
No.	IP	Host name	Processor (CPU = 8)	Memory
1	gamma1.hpc.csie.thu.edu.tw	gamma1*	Pentium III × 2	512 MB
2	gamma2.hpc.csie.thu.edu.tw	gamma2	Pentium III × 2	512 MB
3	gamma3.hpc.csie.thu.edu.tw	gamma3	Pentium III × 2	512 MB
4	gamma4.hpc.csie.thu.edu.tw	gamma4	Pentium III × 2	512 MB
Cluster3				
No.	IP	Host name	Processor (CPU = 6)	Memory
1	alpha1.hpc.csie.thu.edu.tw	alpha1*	AMD Athlon MP 2400+ × 2	1024 MB
2	alpha2.hpc.csie.thu.edu.tw	alpha2	AMD Athlon MP 2000+ × 2	1024 MB
3	alpha3.hpc.csie.thu.edu.tw	alpha3	AMD Athlon MP 1800+ × 2	512 MB

*Stand for Master node of the cluster, the others is slave node

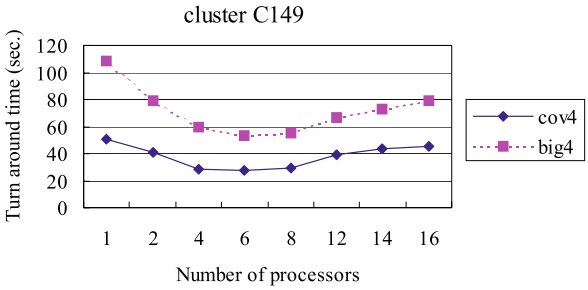


Fig. 15 The turn-around time of C149 diagram

Figure 16 shows our experimental result of the Grid environment. During the experiment test, for the same case, we got the different turn-around time every time, but the interval within one minute. We tried to do our best to get the result in the same condition. It is still changeable. It is comprehensible because the interchange communication is unsettled in the Internet. Therefore, we chose the least time of every processor for this experimental presentation. So, the results of 6 processors and 12 processors are higher suddenly, because they just cross the external node. After that, the workload is average because the speedup is increasing. We obtain the amazing discovery that we can get the lower turn-around time for the maximum processor. As seen from the figure, all the execution times are very close after 4 processors, but big4

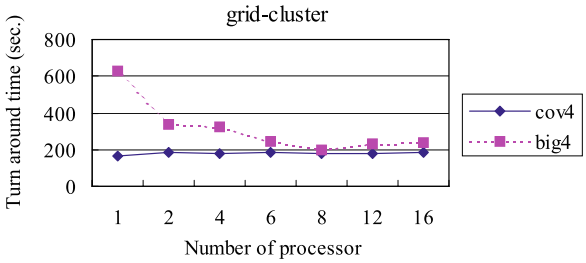


Fig. 16 The turn-around time of grid-cluster diagram

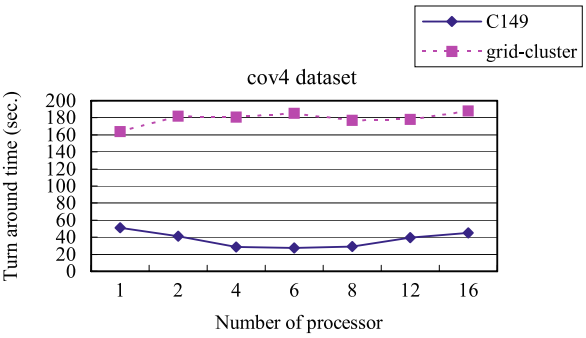


Fig. 17 Cluster and Grid comparison diagram for cov4 dataset

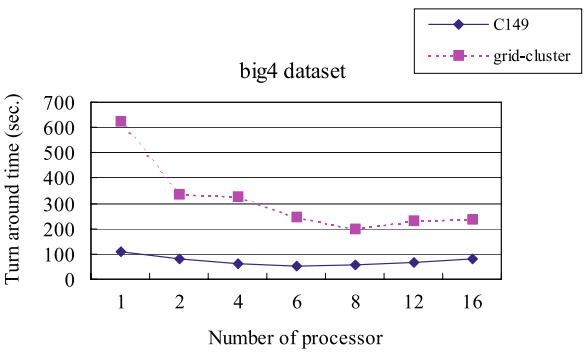


Fig. 18 Cluster and Grid comparison diagram for big4 dataset

dataset is double than cov4 dataset. All prove that Grid computing is available in the particular condition. It means the proposed method is scalable for large dataset.

Next, we compare Cluster with Grid. We know their hardware are different, we just put their results together and compare their curve varieties. Furthermore, their CPU gap is not too much. The data values as the same former experiment are shown in Fig. 17 and Fig. 18. Obviously, for the speedup, Cluster is much better than Grid, even just one processor. We believe that the reason is due to the different MPI library; MPICH-G2 has more of a communication layer with Globus toolkit. But on the other hand, the increasing processor is good for Grid. As seen in the figure, the execution time is just a little decreasing when the processor is increasing in the Grid, and the Cluster is not. We can see that the performances of a large dataset are near gradual. Particularly, the processor is increasing.

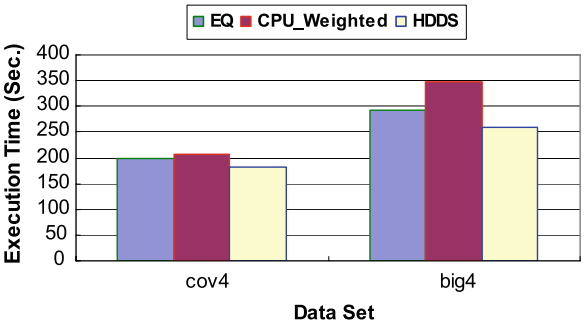


Fig. 19 Performance of data partition schemes for different datasets

Relative performance for different dataset sizes First, execution time on the grid for the three schemes is investigated. Figure 19 illustrates execution time of EQ, CPU_Weighted and HDDS, with dataset cov4 and big4, respectively. Experimental results show that the HDDS scheme got better performance than the others.

From this experiment, we can see the significant influence of partition schemes on the total response time. In grid environments, network bandwidth is an important criterion to evaluate the performance of a slave node. EQ and CPU_Weighted are static data partition schemes. Therefore, they cannot adapt to the practical network status. When communication cost becomes a major factor, the proposed HDDS scheme would be well adaptive to the network environment.

Moreover, the reason why CPU_Weighted got the worst performance can be contributed to the inappropriate estimation of node performance. In grid computing environments, CPU speed is not the only factor to determine the node performance. A node with the fastest CPU is not necessarily the node with optimal performance. This has been illustrated in Fig. 19.

Experiment discussion Obviously, for the speedup, Cluster is much better than Grid in our case. On the other hand, we do not schedule the workload into the heterogeneous configuration. It also affects the entire performance. We believe that more processors will result in more communication overhead between the slaves and the master. However, if the large dataset is appropriately distributed, the benefit of saving computation time will override the communication overhead. Frankly, we have bottlenecks for more attributes and larger dataset in the Grid environment. The dataset is too large to Grid segmentation. We plan use larger synthetic dataset in next experiment and try the other way to solve it, even to modify the current program or try another parallel decision tree algorithms. After all, we demonstrate that Grid is not bad in the multiple processors.

Our experiment reveals the other appearance that the speedup will be a little down when the processor first time touches the other Cluster processor in the Grid. But it speeds up later. Of course, it will affect the Grid type that consists of plenty single workstations. It is worth noting.

The turn-around time of a multiple processor does not double increase when the dataset is double; oppositely, the speedup is closed. It is also amazing and satisfying. All the experimental results can make us get going to research Grid computing for data mining.

6 Conclusion

In this paper, we have proposed a performance-based heuristic to solve the data partition problem for data mining applications, and have compared it with other algorithms by experiments for two types of application programs on our Grid environment. In each case, our approach can obtain performance improvement on other schemes. In our future work, we will implement more types of application programs to verify our approach. Furthermore, we hope to study theoretical analysis to find better solutions, and consider more status information.

The main goals of this paper are providing the idea architecture for decision tree model in the Grid environment, and analyzing Grid roles of user equality perspective in the data mining application. We hope the Grid-based decision tree architecture plus the roles of Grid can help the emerging Grid community. Also, expecting the evaluation report of the experimental performance in different platforms is interesting to the computer researchers.

This preliminary experiment is the first step to achieve our Grid-based decision tree architecture. After all, it is a big plan; we need to know what arduous problem we will meet, and step-by-step achieve it. We think if the transplant is easy about Cluster into Grid, then we can smoothly go to the next work that implements the decision tree application. This ideal application can assign separately the task to single PC and parallel computer according to the response of Globus Toolkit. Further, we will involve the different data media, database, or text file based on computer side needs.

In this paper, we just want to know the difference between Cluster and Grid in efficiency and try not to modify the original program. Our preliminary experimental results show that Cluster is better than Grid in our datasets, but increasing processor advantage Grid. Therefore, our next work is to implement the large dataset and more attributes in Grid, and maybe try to use another decision tree algorithm. Finally, we will have an application that can manage and assign the work to a remote workstation to achieve Grid-based decision tree architecture that we propose.

Our experiment reveals Grid computing is expansibility; we believe it is expected that Grid computing will be a trend and common. Now, there are more and more researchers and large industries investing much time and money in Grid investigation. We will have more tools to assist in achieving our goal that is integrated Grid-based decision tree application.

It is a good idea to consider MIPS in the Performance Ratio calculation. We will incorporate this attribute into the Performance ratio formula in the future work. Indeed, in another paper [40] of ours about Workload Distribution, CPU speed has been considered in the Performance Ratio calculation. And the experimental results show good performance. We believe that the MIPS attribute should have the same effect as the CPU speed.

This work assumes that the scalability problem can be alleviated by such fault-tolerant technologies as server replication. Our future work will extend the master/slave architecture to peer-to-peer models to overcome scalability problems.

We hope, through Grid collaboration, it can make data mining tasks originally thought too difficult or complex to become possible. And all can be implemented in every different research field, despite the fact that current Grid-based data mining

applications still have many problems and limit to different scientific communities. But from the perspective of the network development and global common objectives, Grid is an ideal of the human being and a feasible platform.

Acknowledgements This paper is supported in part by the National Science Council, Taiwan, under grants no. NSC 96-2221-E-029-019-MY3, NSC 97-3114-E-007-001, and NSC 97-2511-S-468-003.

References

1. Agrawal R, Jagadish H (1988) Partition techniques for large-grained parallelism. IEEE Trans Comput 37(12):1627–1634
2. Agrawal R, Shafer JC (1996) Parallel mining of association rules. IEEE Trans Knowl Data Eng 8(6):962–969
3. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proc 20th very large data bases conf, pp 487–499
4. Allcock B, Tuecke S, Foster I, Chervenak A, Kesselman C (2000) Protocols and services for distributed data-intensive science. ACAT2000 Proceedings, pp 161–163
5. Allcock W, Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S (2001) The data grid: towards an architecture for the distributed management and analysis of large scientific datasets. J Netw Comput Appl 23:187–200
6. Alsabti K, Ranka S, Singh V (1998) CLOUDS: a decision tree classifier for large datasets. In: Proc KDD'98, 4th intl conf on knowledge discovery and data mining, New York City, pp 2–8
7. Baker MA, Fox GC (1999) Metacomputing: harnessing informal supercomputers. In: High performance cluster computing. Prentice-Hall, New York. ISBN 0-13-013784-7
8. Beaumont O, Casanova H, Legrand A, Robert Y, Yang Y (2005) Scheduling divisible loads on star and tree networks: results and open problems. IEEE Trans Parallel Distrib Syst 16(3):207–218
9. Benoît G (2002) Data mining. In: Cronin B (ed) Annual review of information science and technology, vol 36. American Society for Information Science and Technology, Silver Spring, pp 265–310
10. Bharadwaj V, Ghose D, Mani V, Robertazzi TG (1996) Scheduling divisible loads in parallel and distributed systems. IEEE Press, New York
11. Bharadwaj V, Ghose D, Robertazzi TG (2003) Divisible load theory: a new paradigm for load scheduling in distributed systems. Cluster Comput 6(1):7–18
12. Cannataro M, Congiusta A, Pugliese A, Talia D, Trunfio P (2004) Distributed data mining on grids: services, tools, and applications. IEEE Trans Syst Man Cybern B 34(6):2451–2465
13. Comino N, Narasimhan VL (2002) A novel data distribution technique for host-client type parallel applications. IEEE Trans Parallel Distrib Syst 13(2):97–110
14. Di Fatta G, Berthold MR (2006) Dynamic load balancing for the distributed mining of molecular structures. IEEE Trans Parallel Distrib Syst 17(8):773–785
15. Divisible Load Theory, http://www.ee.sunysb.edu/~tom/MATBE/index.html
16. Dynamic Load Distribution, http://homepages.mcs.vuw.ac.nz/~kris/thesis/node11.html
17. Foster I (2002) The grid: a new infrastructure for 21st century science. Phys Today 55(2):42–47
18. Foster I, Karonis N (1998) A grid-enabled MPI: message passing in heterogeneous distributed computing systems. In: Proc 1998 SC conference, November 1998
19. Foster I, Kesselman C (1997) Globus: a metacomputing infrastructure toolkit. Int J Supercomput Appl 11(2):115–128
20. Foster I, Kesselman C (eds) (1999) The grid: blueprint for a new computing infrastructure, 1st edn. Morgan Kaufmann, San Mateo
21. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. Int J Supercomput Appl 15(3)
22. Foster I, Kesselman C, Nick J, Tuecke S (2002) The physiology of the grid: an open grid services architecture for distributed systems integration. Globus project
23. Fox G (2003) Education and the enterprise with the Grid. In: Berman F, Fox G, Hey T (eds) Grid computing: making the global infrastructure a reality. Wiley, New York
24. Grimshaw AS (1992) Meta-systems: an approach combining parallel processing and heterogeneous distributed computing systems. Workshop on heterogeneous processing, international parallel processing symposium, pp 54–59

25. Hagiwara J, Doi T, Shindo T, Yaginuma Y, Maeda K (1997) Commercial applications on the AP3000 Parallel Computer. IEEE massively parallel programming models'97
26. Han J, Kamber M (2001) Data mining: concepts and techniques. Morgan Kaufmann, San Mateo
27. Hinke TH, Novotny J (2000) Data mining on NASA's information power grid. HPDC
28. Hinke TH, Novotny J (2000) Data mining on NASA's information power grid. HPDC
29. Huang F, Li Z, Sun X (2008) A data mining model in knowledge grid. In: The 4th international conference on wireless communications, networking and mobile computing (WiCOM'08), pp 1–4, 12–14 Oct 2008
30. Introduction to Grid Computing with Globus, http://www.ibm.com/redbooks
31. KISTI Grid Testbed, http://Gridtest.hpcnet.ne.kr/
32. MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/
33. MPICH-G2, http://www.hpclab.niu.edu/mpi/
34. Narlikar G (1998) A parallel, multithreaded decision tree builder. Tech Report CMU-CS-98-184, December 1998
35. Network Weather Service, http://nws.cs.ucsb.edu/
36. Open Grid Forum, http://www.ogf.org/
37. Orlando S, Palmerini P, Perego R, Silverstri F (2002) Scheduling high performance data mining tasks on a data grid environment. Proceedings of Europar
38. Robertazzi TG (2003) Ten reasons to use divisible load theory. Computer 36(5):63–68
39. Shafer J, Agrawal R, Mehta M (1996) SPRINT: a scalable parallel classifier for data mining. In: Proc of VLDB
40. Shih W-C, Yang C-T, Tseng S-S (2009) Using a performance-based skeleton to implement divisible load applications on grid computing environments. J Inf Sci Eng (JISE) 25(1):59–81
41. Sun ONE Grid Engine, http://wwws.sun.com/software/Gridware/
42. Sunderam VS (1990) PVM: A framework for parallel distributed computing. Concurr Pract Exp 2(4):315–339
43. Talia D (2002) High-performance data mining and knowledge discovery. Euro-Par, Paderborn, Germany, August 2002
44. Taniar's D Homepage, http://www-personal.monash.edu.au/~dtaniar/VPAC/parsprint.zip
45. TeraGrid, http://www.teraGrid.org/
46. The Globus Project, http://www.globus.org/
47. THU Bandwidth Statistics GUI, http://140.128.102.187/nws/show.jsp
48. Yang C-T, Shih W-C, Tseng S-S (2008) A heuristic data distribution scheme for data mining applications on grid environments. In: IEEE international conference on fuzzy systems, 2008 (FUZZ-IEEE 2008), Jun 1–6, 2008, Hong Kong, pp 2398–2404
49. Zaki MJ (1999) Parallel and distributed association mining: a survey. IEEE Concurr 7(4):14–25
50. Zaki MJ, Ho C-T, Agrawal R (1999) Parallel classification for data mining on shared-memory multiprocessors. ICDE 1999, pp 198–205

Wen-Chung Shih received the Ph.D. degree in Computer Science from National Chiao Tung University in 2008. Since 2004, he has worked as a librarian in the National Chi Nan University library, Taiwan. In August 2008, he joined the faculty of the Department of Information Science and Applications at Asia University, where he is currently an assistant professor. His research interests include e-learning, ubiquitous learning, grid computing, and expert systems.

Chao-Tung Yang is a professor of computer science at Tunghai University in Taiwan. He was born on November 9, 1968 in Ilan, Taiwan, ROC and received a B.S. degree in computer science from Tunghai University, Taichung, Taiwan, in 1990, and the M.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 1992. He received the Ph.D. degree in computer science from National Chiao Tung University in July 1996. He won the 1996 Acer Dragon Award for an outstanding Ph.D. dissertation. He has worked as an associate researcher for ground operations in the ROCSAT Ground System Section (RGS) of the National Space Program Office (NSPO) in Hsinchu Science-based Industrial Park since 1996. In August 2001, he joined the faculty of the Department of Computer Science and Information Engineering at Tunghai University. He got the excellent research award by Tunghai University in 2007. He got the excellent research award by Tunghai University in 2007. In 2007 and 2008, he got the Golden Penguin Award by Industrial Development Bureau, Ministry of Economic Affairs, Taiwan. His researches have been sponsored by Taiwan agencies National Science Council (NSC), National Center for High Performance Computing (NCHC), and Ministry of Education. His present research interests are in grid and cluster computing, parallel and multi-core computing, and Web-based applications. He is both a member of the IEEE Computer Society and ACM. He has published more than 30 journal papers.

Shian-Shyong Tseng received the Ph.D. degree in Computer Engineering from the National Chiao Tung University in 1984. Since August 1983, he has been on the faculty of the Department of Computer and Information Science at National Chiao Tung University, and is currently a professor there. From 1988 to 1991, he was the Director of the Computer Center at National Chiao Tung University. From 1991 to 1992 and 1996 to 1998, he acted as the Chairman of Department of Computer and Information Science. From 1992 to 1996, he was the Director of the Computer Center at the Ministry of Education and the Chairman of Taiwan Academic Network (TANet) management committee. From 1999 to 2003, he has participated in the National Telecommunication Project and acted as the Chairman of the Network Planning Committee and National Broadband Experimental Network (NBEN). From 2003 to 2006, he has acted as the principal investigator of the Taiwan SIP/ENUM trial project and the Chairman of the SIP/ENUM Forum Taiwan. In December 1999, he founded the Taiwan Network Information Center (TWNIC) and was the Chairman of the board of directors of TWNIC from 1999 to 2005. Since August 2005, he has been the Dean of the College of Computer Science, Asia University. He is also the Director of the e-learning and application research center at National Chiao-Tung University. His current research interests include expert systems, data mining, computer-assisted learning, and Internet-based applications. He has published more than 100 journal papers.

Copyright of Journal of Supercomputing is the property of Springer Science & Business Media B.V. and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.