Distributed Data Mining in Grid Computing Environments

Ping Luo[a b], Kevin Lü[c*], Zhongzhi Shi[a] and Qing He[a]

[a]Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China

[b]Graduate School of the Chinese Academy of Sciences, 100080, Beijing, China

[c]Brunel University, Uxbridge, U. K. UB8 3PH

The computing-intensive data mining for inherently Internet-wide distributed data, referred as Distributed Data Mining (DDM), calls for the support of a powerful Grid with an effective scheduling framework. DDM often shares the computing paradigm of *local processing* and *global synthesizing*. It involves every phase of Data Mining (DM) processes, which makes the workflow of DDM very complex and can be modelled only by a Directed Acyclic Graph (DAG) with multiple data *entries*. Motivated by the need of a practical solution of the Grid scheduling problem for the DDM workflow, this paper proposes a novel two-phase scheduling framework, including *External Scheduling* and *Internal Scheduling*, on a two-level Grid architecture (InterGrid, IntraGrid). Currently a DM IntraGrid, named DMGCE (Data Mining Grid Computing Environment), has been developed with a dynamic scheduling framework for competitive DAGs in a heterogeneous computing environment. This system is implemented in an established Multi-Agent System (MAS) environment, in which the reuse of existing DM algorithms is achieved by encapsulating them into agents. Practical classification problems from oil well logging analysis are used to measure the system performance. The detailed experiment procedure and result analysis are also discussed in this paper.

1. Introduction

With the vast improvements in wide-area network performance and powerful yet low-cost computers, Grid computing has emerged as a promising attractive computing paradigm. The underlying principle of computational Grid is the notion of providing computing power transparently in an analogy with electrical power. It aims to aggregate distributed computing resources, hide their specifications and present a homogeneous interface to end users for high performance or high throughput computation. Thus, instead of computing locally, users dispatch their tasks to the Grid and use the remote computing resourses. To achieve the promising potentials of computational Grids, an effective and efficient scheduling framework within Grids is fundamentally important.
Recently, DDM has attracted lots of attention among the data mining community [1]. DDM refers to the mining of inherently distributed datasets, aiming to generate global patterns from the union set of locally distributed data. However, the security issue among different local datasets and the huge communication cost in data migration prevent moving all the datasets to a public site. Thus, the algorithms of DDM often adopt a computing paradigm of local processing and global synthesizing, which means that the mining process takes place at a local level and then at a global level where local data mining results are combined to gain global findings. Furthermore, the local processing often concerns multiple phases of data mining, including preprocessing, training and evaluation. The diversity of algorithms in each mining phase makes the DDM workflow so complex that it requires a DAG to model it.

This paper concerns the development of a scheduling framework on a two-level Grid architecture illustrated in Figure 1 for complex DDM workflows. In the two-level Grid the low level is

*Corresponding author. Tel:+44-1895203122; Fax:+44 1895 203 149; E-mail addresses: kevin.lu@brunel.ac.uk, luop@ics.ict.ac.cn

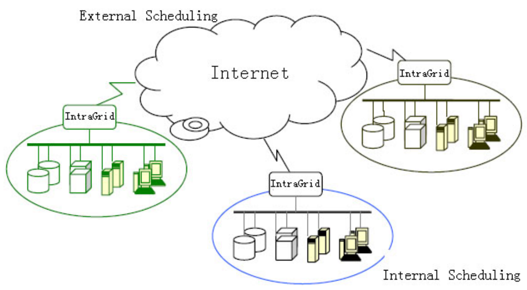


Figure 1. Two-Level Architecture of InterGrid [2]

IntraGrids while the high level is an InterGrid.

IntraGrid A typical IntraGrid topology exists within a single organization. This organization could be made up of many computers, which are connected by a private high-speed local network. The primary characteristic of an IntraGrid is the bandwidth guarantee on the private network.

InterGrid An InterGrid is an Internet-wide Grid, consisting of multiple IntraGrids connected by WAN. Due to WAN connectivity the communication speed between IntraGrids could be comparably slow.

Our approach for scheduling complex DDM DAG is performed in two phases: external scheduling and internal scheduling. They are involved with InterGrids and IntraGrids, respectively. Issues such as scalability, flexibility, adaptability are critical for a practical wide-area deployment of Grid systems, which require an efficient and effective scheduling framework. That is the motivation of this study.
The arrangement of the rest of this paper is as follows. Section 2 describes the workflow of distributed classification and formalizes the scheduling problem. Section 3 presents the two-phase scheduling framework, including the external scheduling at the InterGrid level and the internal scheduling within an IntraGrid. Section 4 evaluates the performance of the developed DM IntraGrid by real-world datasets for classification. The related work and conclusions will be given in Section 5. The implementation issues of this DM IntraGrid in the multi-agent system environment is omitted due to the space limitation.

2. Workflow of DDM: a Computing Paradigm of Local Processing and Global Synthesizing

DDM is the process of performing data mining in distributed computing environments, where users, data, hardware and data mining software are geographically distributed. It emerges as an area of research interests to deal with naturally distributed and heterogeneous databases and then to address the scalability bottlenecks of mining very large datasets [3]. A number of distributed algorithms have been developed for different DDM tasks, including distributed classification, clustering and association [1]. The fundamental concept of these algorithms is that each local dataset is mined individually and the local patterns obtained are then combined to produce global patterns of the entire data. Thus, they mostly adopt the computing paradigm of local processing and global synthesizing for different DDM tasks. Because this computing paradigm is generally adopted in DDM, we aim to present an effective Grid scheduling framework for it. In the following, a workflow of distributed classification is given as the running example of our Grid application.

2.1. Distributed Classification: a Running Example

The DM workflow for classification in Figure 2 aims to find the optimal local classification pattern for the local dataset. It is a complex, highly dynamic, and resource-intensive process, which consists of several different phases. In each phase, many different algorithms are available with different parameters. The workflow in Figure 2 consists of preprocessing, training&testing and evaluation phase. The preprocessing phase can be subdivided into three sequential sub-phases of normalization, discretization, and attribute reduction based on rough set. The mining steps within a phase are optional operations with different per-

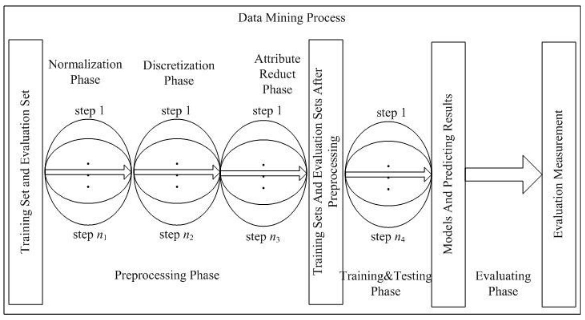


Figure 2. Data mining process in local processing

formances. For convenience and clarity, we give the following definitions.

Definition 2.1 (DM Step)**.** *A DM step corresponds to a particular algorithm to be executed, provided a dataset and a certain set of input parameters for it. Each DM step* Λ *is described as a quad:*

$$\Lambda = (A, F, D, \vec{P})$$

where A *is the data mining algorithm,* F *is the data mining phase that contains the algorithm* A*,* D *is the input dataset and* $\vec{P}$ *is the vector of algorithm parameters.*

Let $\Lambda_1 = (A_1, F_1, D_1, \vec{P_1})$ and $\Lambda_2 = (A_2, F_2, D_2, \vec{P_2})$, $\Lambda_1 = \Lambda_2$ if and only if $A_1 = A_2, F_1 = F_2, D_1 = D_2, \vec{P_1} = \vec{P_2}$.

Definition 2.2 (DM Path)**.** *Let* $\Lambda_1 = (A_1, F_1, D_1, \vec{P_1}), \cdots, \Lambda_k = (A_k, F_k, D_k, \vec{P_k})$*, DM Path is* $\vec{\Lambda} = (\Lambda_1, \cdots, \Lambda_k)$*, where* $F_i (1 \leq i \leq k)$ *is the i-th phase of the whole k-phase data mining process.*

In Figure 2, a DM path can be easily obtained after we select a DM step from each mining phase. If there are n_1, n_2, n_3, n_4 different DM steps in the four phases of normalization, discretization, attribute reduction and training&testing respectively, the number of all possible DM paths would be $n_1 \times n_2 \times n_3 \times n_4$ according to the Multiply Theorem. Along a DM path, a mining step transfers its output to the following step until the path terminates and the final result would be obtained. Then, using the training and validation datasets as an input of the DM path, a measurement will be obtained for this path according to certain evaluation criterion. For classification problems, the evaluation measurements could be accuracy, weighted accuracy and AUC (Area Under Curve), etc. After exhaustively evaluating all the DM paths, ranks of all resultant patterns for all DM paths are generated.
After local processing, the global synthesizing begins. The combining techniques include voting, arbitrating, combining and stacked generalizer, etc. However, the computing flow in this process is much simpler than that in local processing.

2.2. Workflow Model of Distributed Data Mining

We model the DM workflow as a weighted DAG, $G = G(V, E)$, where $V = \{v_1, \cdots, v_n\}$ is a set of weighted node and E is a set of weighted directed edges, representing data dependencies and communications between nodes. A node in the DAG represents a job (referred to as the corresponding DM step), which must be executed without preemption on a host. The weight of a node is referred to as the standard computation cost, representing its execution time on a standard computer, denoted by $\Delta_{standard}(v_i)$. $e_{ij} = (v_i, v_j) \in E$ indicates data transportation from job v_i to v_j, and $|e_{ij}|$ represents communication cost between these two jobs if they are not executed on the same machine. The precedence constraints of a DAG require that a node should not start executing before it gathers all the data from its predecessors. The node without predecessors is called the *entry* of G. The node without successors is called the *end* of G. The *critical path* of G is the longest path (there can be more than one longest path) from an entry to an end of G. The weight of this path is the sum of the weights of the nodes and edges along this path. In the following, a *task* refers to a DAG and a *job* refers to a node in a DAG.
Figure 3 is the corresponding un-weighted DAG of the DM process in Figure 2. The direction of all the edges in Figure 3 is from the node in

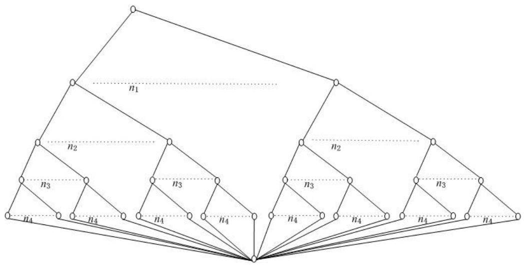


Figure 3. The DAG of classification workflow

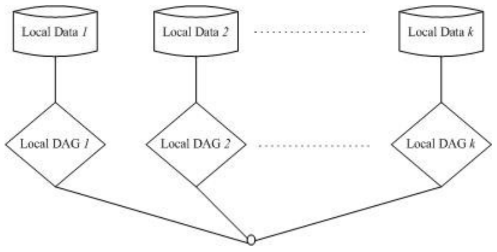


Figure 4. The DAG of the whole distributed classification

the upper layer to the one in the lower layer. If we feed the datasets to the uppermost node in Figure 3, after the whole computation the lowermost node in this figure will output the rank of all patterns for all DM paths, indicating the optimal local pattern. Figure 4 depicts the DAG of the whole distributed classification with k local data sites. Thus, it is a k-entry DAG with k local sub-DAGs. Each local sub-DAG represents the complex local processing, pictured in Figure 3. The lowermost node in this figure corresponds to the synthesizing processing of local patterns and ultimately output the global pattern.

2.3. Problem Definition and Assumptions

Consider the following computation problem of DDM. The local site, which owns local data for DDM, can be scattered anywhere on the Internet. However, it has not enough computing power to support the complex local processing. This task of DDM is then fed to a dedicated InterGrid. The InterGrid simultaneously supports the computation of multiple competitive DDM DAGs. Our research objective is to propose an effective and efficient scheduling framework for DDM DAGs. We assume that the InterGrid is connected via a two-level hierarchical network as illustrated in Figure 1. The first level is an Internet-wide network (WAN) that connects local area networks (LANs) at the second-level. A group of machines, connected by LAN, form an IntraGrid. The local communication cost between computers within an IntraGrid is ignored due to the following reasons: 1) the network bandwidth within an IntraGrid is high speed and 2) even if the volume of the transferred data is large, its corresponding processing time on a computer is much longer than its communication time. However, the communication cost between IntraGrids is considered because of the limited and dynamic bandwidth on a WAN.

3. Two-Phase Scheduling for Distributed Classification Workflow in an InterGrid

The Grid scheduling process of the workflow of distributed classification consists of four steps: *partition of distributed classification workflow*, *external scheduling*, *internal scheduling* and *synthesization of local patterns*. Partition of distributed classification workflow divides the whole k-entry DAG into k sub-DAGs, each of which represents the corresponding local processing. The external scheduling involves the process of mapping the resultant sub-DAGs onto suitable IntraGrids according to some criterion, considering communication costs and IntraGrid credibilities. It is a WAN-wide and DAG-level scheduling. After an IntraGrid receives a sub-DAG it maps the jobs in the sub-DAG onto the computers in it, while keeping the job precedence constraints. This is the process of internal scheduling, which is a LAN-wide and job-level scheduling. After all the IntraGrids send their local patterns to a public site, the synthesization process begins and eventually outputs the global pattern. The entire scheduling process is described in Algorithm 1.
The partition algorithm for DDM workflow in our running example is straightforward. After seg-

menting the edges between the lowermost node in Figure 4 and all its predecessors, the sub-DAGs are generated. The synthesization process is a computation atom on a machine and does not concern about task scheduling. So we omit the description of these two processes.

Algorithm 1 Scheduling for the Whole DAG of DDM

```
1:  partition the $k$-entry DAG into $k$ sub-DAGs
2:  send the $k$ sub-DAGs to the nearest external scheduler $i$
3:  the external scheduler maps these $k$ sub-DAGs onto suitable IntraGrids by external scheduling algorithm
4:  after receiving a sub-DAG for executing the IntraGrid processes it by internal scheduling algorithm
5:  if a sub-DAG finished notification received then
6:    store this notification
7:    if all sub-DAGs finished then
8:      select a IntraGrid $j$ with the minimal communication cost for moving all local patterns to this IntraGrid
9:      synthesization of local patterns on IntraGrid $j$
10:     return
11:   end if
12: end if
```

3.1. External Scheduling Algorithm

In our scheduling framework we adopt the modification of the external scheduling algorithm presented in [4]. This algorithm processes through the sealed-bid auction and is decentralized since an *external scheduler* resides on each IntraGrid. Once an external scheduler receives a DAG, it sends its bidding request with the Requested Task Response Time (RTRT) to the other external schedulers for task auction. Those bidders (external scheduler) reply to this request and send back its Estimate Task Response Time (ETRT). After receiving all replies it chooses the best IntraGrid with the minimal ETRT.

RTRT is the approximate estimation of the execution time for a DAG. In our algorithm RTRT is estimated by the weight of the critical path of a DAG. Because the DDM sub-DAG is executed within an IntraGrid the communication cost is omitted and then only the weights of nodes are counted for this critical path. The ETRT is determined by three issues: RTRT, Network Transfer Rate (NTR) and Average IntraGrid Credibility (AIC).

IntraGrid Credibility (IC) represents the computing reliability of that IntraGrid. After a DAG is completed we can obtain the Actual Task Response Time (ATRT). Then IC is computed by (1)

$$IC_{i,j} = \frac{ATRT_{i,j}}{ETRT_{i,j}} \tag{1}$$

where $IC_{i,j}$ is the IC of IntraGrid i for DAG$_j$, $ATRT_{i,j}$ is the actual task response time of DAG$_j$ on IntraGrid i, $ETRT_{i,j}$ is the estimate task response time of DAG$_j$ on IntraGrid i.

The AIC is a weighted average that is shown in (2). The initial AIC is set to be 1.

$$AIC_j = old\_AIC_j \cdot (1-\alpha) + IC_{i,j} \cdot \alpha \tag{2}$$

where AIC_j is the AIC of IntraGrid j, old_AIC_j is the previous AIC of IntraGrid j, $IC_{i,j}$ is the IC of IntraGrid i for DAG j. α ($0 \leq \alpha \leq 1$) is the coefficient, indicating the tradeoff between previous and current credibilities. The more α is set to be, the more AIC_j represents the current credibility of IntraGrid j. In [4] α is set to be 0.01. However, we suggest α be a much bigger value to let AIC_j indicate the current computing reliability of IntraGrid j more.

The external scheduler decides which IntraGrid is selected by ETRT, computed by (3)

$$ETRT_{i,j} = (RTRT_i + \frac{Task\_Data\_Size_i}{NTR_{k,j}}) \cdot AIC_j \tag{3}$$

where $ETRT_{i,j}$ is the estimate task response time of task i on IntraGrid j, $RTRT_i$ is the requested task response time of task i, AIC_j is the AIC of IntraGrid j, $Task\_Data\_Size_i$ is the data size of task i, $NTR_{k,j}$ is the network transfer rate between IntraGrid k and IntraGrid j. The pseudocode of external scheduling for a DAG is presented in Algorithm 2.

Algorithm 2 External Scheduling

```
 1: if a DAG submitted then
 2:   for all external schedulers participating in bidding do
 3:     send bidding request with RTRT to external schedulers
 4:   end for
 5:   if a bidding reply received then
 6:     store the bidding reply
 7:     if all bidding replies for this task are received then
 8:       IntraGrid $i$ = IntraGrid with the minimal ETRT
 9:       send task to IntraGrid $i$
10:     end if
11:   end if
12: end if
```

3.2. Internal Scheduling Algorithm

An IntraGrid is a heterogeneous computing (HC) environment, which consists of multiple computers with different configurations, connected by a high-speed LAN. Thus, the existing research results from the field of HC can be adopted in scheduling jobs from DAGs. The research focus of HC is the design of an algorithm, which orchestrates all the computing hardware to perform an application that has diverse computational requirements [5] so as to minimize the completion time, i.e., the overall execution time of the application.

This internal scheduling, in fact, can be described as a problem of *dynamic scheduling for competitive DAGs*. It has been proved, in general, to be NP-complete [6], thus requiring the development of heuristic techniques [7,8] for practical usage. The adopted internal scheduling algorithm in this paper is based on our previous work on scheduling data mining workflows in a heterogeneous computing environment [9], which is designed to satisfy the issues on the characteristic of DM workflows. Based on an approximate estimation of job execution time, this algorithm first maps DM jobs to machines in a *decentralized* and *diligent* manner. Then the performance of this initial mapping can be improved through *job migrations* when necessary. The scheduling heuristic used in it considers the factors of both the *minimal completion time* criterion and the *critical path* in a DAG. These two aspects are integrated and implemented in the initial job mapping process and the job execution control process, respectively. The detail of this internal scheduling algorithm is presented in [9].

4. Experiment Procedure and Results

We first focus on the implementation of the DM IntraGrid, involving internal scheduling only. This IntraGrid, named DMGCE (Data Mining Grid Computing Environment), is developed in a MAS environment MAGE [10] so as to measure the system performance and then to provide this Grid service practically. The evaluation of this system is carried out with practical DM data from well logging analysis. Well logging analysis plays an essential role in petroleum exploration and exploitation. It is used to identify the pay zones of gas or oil in the reservoir formations. The performance metrics in the experiments include task response-time, system throughput and system efficiency defined in the following.

4.1. Experiment Procedure

In these experiments 9 machines with different configurations are used. The main configurations of these machines are listed in Table 1. To measure the system performance based on the metrics mentioned above, a DM task for classification denoted by G^*, is constructed for the whole experiment process. The corresponding DAG of this task, which contains 16 jobs, is isomorphic with the DAG in Figure 3. After removing the end node of the DAG it becomes a tree, which indicates that all the successors of an internal node in the tree can be mapped once its execution is completed. The input data for this DAG is from

Table 1
Machine Configuration List

Machine Type Index	CPU	Main Memory	Machine Amount
1	3 GHz	512 M	5
2	2.8 GHz	512 M	1
3	2.4 GHz	1024 M	1
4	2.2 GHz	512 M	1
5	731 MHz	448 M	1

the well logging analysis. This data contains 2000 labeled examples with 10 numeric condition attributes.

An approximate running time estimation for each job in G^* is generated by the following process. We regard one of the machine in Machine Type 1 of Table 1 as the standard computer. First we execute a group of benchmark DM jobs on each machine i and record the actual total execution time of these jobs, denoted by Δ_i, which would give a performance assessment of the machine i. Next, G^* is executed on the standard machine and the actual running time of each job Λ, denoted by $\Delta_{standard}(\Lambda)$, is recorded. Thus, the average execution time of DM jobs of certain algorithm A on the standard machine can be computed in (4)

$$\Delta_{standard}(A) = \frac{\sum_{\Lambda \in \vec{A}} \Delta_{standard}(\Lambda)}{|\vec{A}|} \quad (4)$$

where $\vec{A}$ contains all the jobs performing algorithm A with different parameters or input data in G^*. And the running time of the average execution time of DM jobs of algorithm A on the other machine i can be estimated by (5)

$$\Delta_i(A) = \rho_i \cdot \Delta_{standard}(A) \quad (5)$$

where $\rho_i = \frac{\Delta_i}{\Delta_{standard}}$. In our system the *machine heterogeneity*, measured by the standard deviation of all ρ_i, is 2.3835. Then, in our experiment the running time of a job $\Lambda(A, F, D, \vec{P})$ on machine i is approximately estimated by $\Delta_i(A)$ as shown in (6)

$$\Delta_i(\Lambda(A, F, D, \vec{P})) = \Delta_i(A). \quad (6)$$

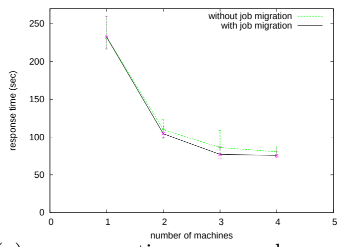
This estimation method considers only the algorithm type it performs, and ignores the other elements in the quad of Λ, so as to check the tolerant performance of the internal scheduling algorithm on the approximate time estimations of DM jobs. These experiments are performed in two parts. In the first part, the 4 machines from Machine Type 1 are used to form a homogeneous system, in order to measure task response time and system throughput versus the number of joining machines with the same configuration. Let the arrival time of the task G be $a(G)$, the completion time of G be $c(G)$, then the response-time of G is $r(G) = c(G) - a(G)$. The system throughput is defined by the number of G^*, which is completed by the system in a fixed time.
The second part of the experiments is to evaluate the scheduling performance in a heterogeneous system, which contains all the 9 machines listed in Table 1. In these experiments exponential distribution is used to generate the task sequence, including 100 tasks of G^*. These tasks are assigned under two inter-arrival times, $t_l = 25$ seconds and $t_h = 50$ seconds. The task arrival time is generated, which satisfies $\frac{|t_a - t|}{t} < 0.06$, where t_a is the actual average inter-arrival time of the task sequence and t is the expected inter-arrival time. We record the average response time of the tasks in the sequence and compute the *weighted* system efficiency in (7), which considers the machine heterogeneity in an HC system.

$$\eta_{weighted} = \frac{t_{computation}}{t_{total}} = \frac{\sum_{i=1}^{l} \frac{t_{computation}(i)}{\rho_i}}{\sum_{i=1}^{l} \frac{t_{total}(i)}{\rho_i}} \quad (7)$$

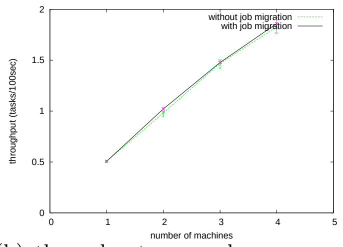
where $t_{computation}(i)$ is the system CPU time for the computation on machine i, $t_{total}(i)$ is the total system CPU time on machine i, l is the number of machines in our system, and ρ_i is the same as the one in (5).
All the above experiments are performed under two situations, with and without job migrations after initial mapping, and repeated five times. The average values of these metrics are listed in subsection 4.2.

4.2. Experiment Results

Figure 5(a) and Figure 5(b) show the results from the first part of experiments. Figure 5(a) illustrates that the response time of a single task

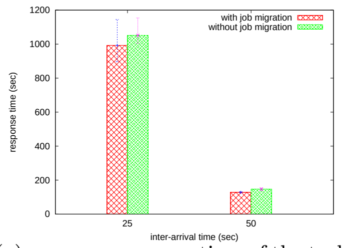


(a) response time versus homogeneous machines

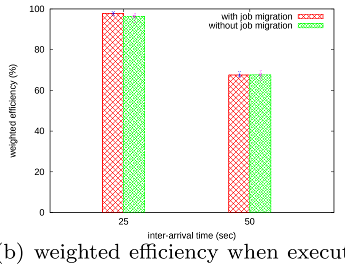


(b) throughput versus homogeneous machines

Figure 5. The experimental results for homogeneous computing



(a) average response time of the tasks in task sequence



(b) weighted efficiency when executing the tasks in task sequence

Figure 6. The experimental results for heterogeneous computing

G^* decreases along with the increase of the number of machines. However, the response time decreases in a non-linear manner and eventually reaches at a minimal level, because in our application the minimal computing granularity is a job, which could not be broken down any further for parallelization. In theory, the minimum response time of a DAG is the weight sum of the critical path in the DAG. Figure 5(b) shows that the throughput of the HC system increases close to linear along with the increase of the number of joining machines. These two figures also show that the use of job migration could improve the system performance in terms of task response time and system throughput.

The results from the second part of the experiments can be seen in Figure 6(a) and Figure 6(b). In Figure 6(a) it can be found that through the use of job migration technique the average response times of the 100 tasks decrease 5.58% and 13.21% for the cases of 25-second inter-arrival and 50-second inter-arrival, respectively. The weighted efficiency of the HC system is also improved through job migration technique, as shown in Figure 6(b).

5. Related Work and Conclusions

5.1. Related Work

The issues of building a computational Grid for Data Mining have been recently addressed by a number of researchers. WEKA4WS [11] adapts the Weka toolkit to a Grid environment and exposes all the 78 algorithms as WSRF-compliant Web Services. FAEHIM (Federated Analysis Environment for Heterogeneous Intelligent Mining) [12] is Web Services based on a toolkit of DM and mainly focuses on the composition of existing DM Web Services by Triana problem solving environment [13]. The Knowledge Grid [14,15] is a reference software architecture for geographically distributed knowledge discovery systems. It is built on top of a computational Grid of Globus and uses basic Grid services to implement the DM services on connected computers. A visual environment for Grid application (VEGA) is developed in this system, supporting visual DM plan generation and automatic

DM plan execution. GridMiner [16] focuses its effort on data mining and On-Line Analytical Processing (OLAP), two complementary technologies, which, if applied in conjunction, can provide a highly efficient and powerful data analysis and knowledge discovery solution on the Grid. Discovery Net [17] builds the world's first e-Science platform for scientific discovery in various fields. To make good use of the computing hardware in heterogeneous systems for DM workflow a scheduling framework is urgently needed. Although this computing paradigm can be achieved by exposing all the DM algorithms as Web Services on every host in this system or by dynamic Web Service deployment, however, the scheduling framework for DM DAG applications, in general, has drawn a very little attention except for the scheduling heuristics mentioned in [15]. Paper [15] also emphasizes the importance of scheduling algorithm in Knowledge Grid and uses the concept of *abstract hosts* to represent any computing host.

5.2. Conclusions

In this paper a novel two-phase scheduling framework, based on the two-level architecture of an InterGrid, is presented for the Internet-wide distributed data mining, which shares the computing paradigm of local processing and global synthesizing. The external scheduling for DDM sub-DAGs operates through the sealed-bid auction, with the consideration of communication costs and IntraGrid credibilities. The internal scheduling for jobs in a sub-DAG is formalized as a problem of scheduling for competitive DM DAGs in heterogeneous computing environments. According to the characteristics of DM workflows, a new internal scheduling framework is adopted based on our previous work [9] with three features: totally decentralized, the hybrid heuristic scheme, and the technique of job migration after mapping. The DM IntraGrid with this internal scheduling algorithm has been implemented in a multi-agent system environment. Its performance has also been tested by real-world DM data, which is demonstrated by our experiments.

Acknowledgements

Our work is supported by the National Science Foundation of China (No.60435010), the national 863 Project (No.2003AA115220), the national 973 Project (No.2003CB317004) and the Nature Science Foundation of Beijing (No.4052025). Kevin Lü would like to show his appreciation to Wang Kuan Cheng Science Foundation, Chinese Academy of Sciences for the funding to enable him to conduct this research.

REFERENCES

1. Y. Fu. Distributed data mining: An overview. *IEEE TCDP newsletter*, 2001.
2. Yanmin Zhu. A survey on grid scheduling systems. Technical report, Computer Science Department of Hong Kong University of Science and Technology, 2003.
3. S. Krishnaswamy, S. Loke, and A. Zaslavsky. Supporting the optimization of distributed data mining by predicting application run times. In *Proceedings of the Fourth International Conference on Enterprise Information Systems*, pages 374–381, Ciudad Real, Spain, 2002.
4. H. Chen and M. Maheswaran. Distributed dynamic scheduling of composite tasks on grid computing systems. In *Proceedings of the 11th IEEE Heterogeneous Computing Workshop*, 2002.
5. Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.
6. D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Transaction on Software Engineering*, 15(11):1427–1436, 1989.
7. Michael Iverson and Fusun Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. In *Proceedings of the Eighth Heterogeneous Computing Workshop*, 1999.
8. Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Poceedings of the 13th Heteroge-*

neous Computing Workshop, 2004.

9. Ping Luo, Kevin Lü, Qing He, and Zhongzhi Shi. Scheduling for data mining workflows in a heterogeneous computing system. Technical report, Institute of Computing Technology, Chinese Academy of Sciences, 2006. http://www.intsci.ac.cn/users/luop/.
10. Zhongzhi Shi, Haijun Zhang, Yong Cheng, Yuncheng Jiang, Qiujian Sheng, and Zhikung Zhao. Mage: An agent-oriented programming environment. In *Proceedings of IEEE International Conference on Cognitive Informatics*, pages 250–257, 2004.
11. D. Talia, P. Trunfio, and O. Verta. Weka4ws: a wsrf-enabled weka toolkit for distributed data mining on grids. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Porto, Portugal, 2005.
12. Ali Shaikh Ali, Omer F. Rana, and Ian J. Taylor. Web services composition for distributed data mining. In *Proceedings of International Conference on Parallel Processing Workshops*, pages 11–18, 2005.
13. The Triana Problem Solving Environment. http://www.trianacode.org.
14. M. Cannataro, D. Talia, and P. Trunfio. Distributed data mining on the grid. *Future Generation Computer Systems*, 18(8):1101–1112, 2002.
15. M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, and P. Trunfio. Distributed data mining on grids: Services, tools, and applications. *IEEE Transactions on Systems, Man and Cybernetics*, 34(6):2451– 2465, 2004.
16. GridMiner. www.gridminer.org.
17. Discovery Net. www.discovery-on-the.net.