A Heterogeneous Computing System for Data Mining Workflows

Ping Luo[1,2], Kevin Lü[3], Qing He[2], and Zhongzhi Shi[1]

[1] Key Laboratory of Intelligent Information Processing,
Institute of Computing Technology, Chinese Academy of Sciences,
P.O. Box 2704-28, Beijing 100080 China
[2] Graduate School of the Chinese Academy of Sciences, Beijing, China
[3] Brunel University, Uxbridge, U.K. UB8 3PH
luop@ics.ict.ac.cn

Abstract. The computing-intensive Data Mining (DM) process calls for the support of a Heterogeneous Computing (HC) system, which consists of multiple computers with different configurations, connected by a high-speed LAN, for increased computational power and resources. DM process can be described as a multi-phase pipeline process, and in each phase there could be many optional methods. This makes the workflow of DM very complex and can be modelled only by a Directed Acyclic Graph (DAG). An HC system needs an effective and efficient scheduling framework, which orchestrates all the computing hardware to perform multiple competitive DM workflows. Motivated by the need of a practical solution of the scheduling problem for the DM workflow, this paper proposes a dynamic DAG scheduling algorithm according to the characteristics of execution time estimation model for DM jobs. Based on an approximate estimation of job execution time, this algorithm first maps DM jobs to machines in a *decentralized* and *diligent* (defined in this paper) 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. We implement this system 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 are used to test and measure the system performance. The detailed experiment procedure and result analysis are also discussed in this paper.

Keywords: Data mining, heterogeneous computing, directed acyclic graph, multi-agent system environment.

1 Introduction

Current Data Mining (DM) tools contain a plethora of algorithms, but lack the guidelines to appropriately select and arrange these algorithms according to the nature of the problem under analysis. Given a practical DM problem,

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 177–189, 2006.
© Springer-Verlag Berlin Heidelberg 2006

an expedient solution is to evaluate all the possible DM schemes modeled as a Directed Acyclic Graph (DAG), and rank them according to certain performance metrics. This yields a Grid computing problem, which aims to construct an Heterogeneous Computing (HC) system, supporting the executions of DM workflows.

An HC system, which consists of multiple computers with different configurations connected by a high-speed LAN, responses multiple computational requests of DM simultaneously. This system emerges as the provider of Internet-based data mining services, and offers an attractive option for small to medium range organizations, which are the most constrained by the high cost of data mining software, and consequently, stand to benefit by paying for software usage without having to incur the costs associated with buying, training and maintenance.

This study aims to construct such an HC system and mainly focuses on the effective and efficient scheduling framework to orchestrate all the computing hardware in it to perform multiple competitive DM workflows. According to the characteristics of execution time estimation model for DM jobs, we propose a *dynamic* scheduling framework for DM workflows. It has the following features:

- The scheduling operation performs in a totally *decentralized* and *diligent* manner, which avoids the computation bottleneck for centralized scheduling and increases the system robustness.
- This scheduling framework supports simultaneous computing of multiple competitive DAGs. The execution sequence of DM jobs considers the factors of both the precedence constraints in a DAG and the arrival order of these DAGs.
- This scheduling framework is tolerant to approximate time estimations of DM jobs. The initial mapping, based on the approximate running time estimations, will be improved by *job migrations*.

The arrangement of the rest of this paper is as follows. Section 2 describes the DM workflow for classification as a running example and formalizes the scheduling problem. In Section 3 we propose the dynamic scheduling algorithm for competitive DM workflows. Section 4 evaluates the performance of the data mining HC system with the presented scheduling algorithm by real-world datasets. The related work and conclusions will be given in Section 5. The discussions about the execution time estimation model for DM jobs, the implementation issues of this DM HC system in a Multi-Agent System (MAS) environment, and the details about the approximate execution time estimation method used in the experiment are omitted due to the space limitation. The full version of this paper can be downloaded from [1].

2 Data Mining Workflow for Heterogeneous Computing

2.1 Data Mining Workflow for Classification: A Running Example

The DM workflow for classification in Figure 1, used as a running example in this paper, aims to find the optimal classification pattern for the input dataset.

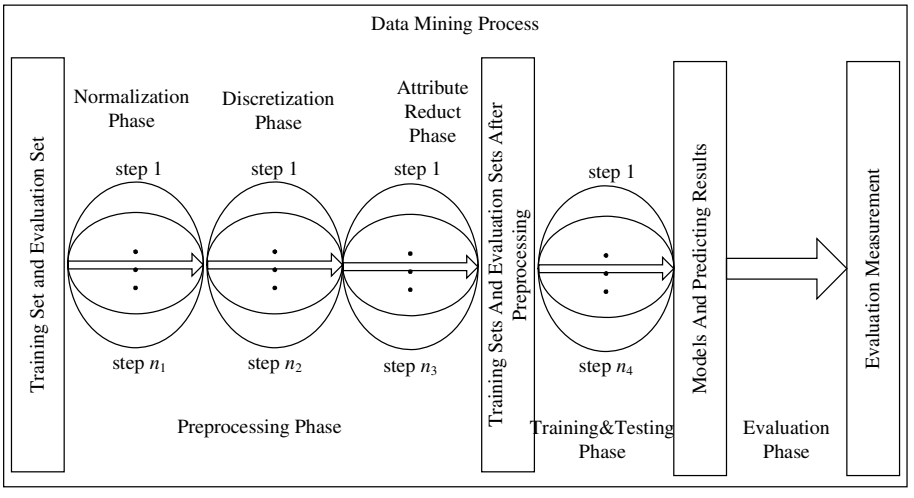


Fig. 1. Data mining process for classification

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 1 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. The mining steps within a phase are optional operations, which would output different results. For convenience and clarity, we give the following definitions.

Definition 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 quadruple:*

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

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

Definition 2 (DM Path). *Let Λ_1=$(A_1, F_1, D_1, \boldsymbol{P_1}), \cdots, \Lambda_k$=$(A_k, F_k, D_k, \boldsymbol{P_k})$, DM Path is $\boldsymbol{\Lambda} = (\Lambda_1, \cdots, \Lambda_k)$, where $F_i (1 \leq i \leq k)$ is the i-th phase of the whole k-phase data mining process.*

In Figure 1, 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 each of 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.

2.2 Workflow Model of 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 nodes 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. Consider the HC system consisting of l machines $m_1, \cdots, m_l$, the weight vector of a node v is referred to as the computation cost vector $\boldsymbol{\Delta(v)} = \{\Delta(v, m_1), \cdots, \Delta(v, m_l)\}$, where $\Delta(v, m_i)$ represents its execution time on a machine m_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.

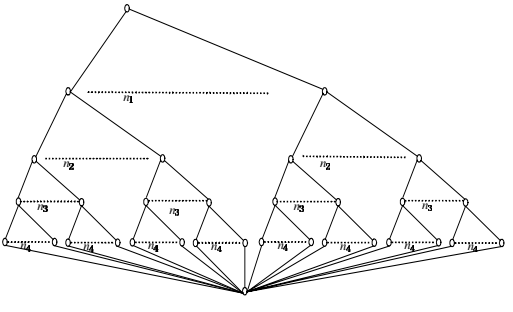


Fig. 2. The DAG of classification workflow

Figure 2 is the corresponding un-weighted DAG of the DM process in Figure 1. The direction of all the edges in Figure 2 is from the node in the upper layer to the one in the lower layer. If we feed the dataset to the uppermost node in Figure 2, after the whole computation the lowermost node in this figure will output the rank of all patterns for all DM paths, indicating the optimal classification pattern.

3 Dynamic Scheduling for Competitive DAGs in an HC System

We consider the following 4 issues in developing the scheduling algorithm within an HC system.

- The time estimation model for DM jobs. It is assumed to be provided in advance as a function with three parameters: 1) DM algorithm, 2) feature vector of input data and 3) user-specified algorithm attributes. Thus, the execution time of the node is not known a priori until its input datasets are all gathered. When a DAG is being processed, only if all the predecessors of a node are finished, the estimation model then can use the gathered immediate results to predict the execution time of this node. Therefore, the mapping process must be performed during the job executions and only *dynamic* scheduling can be adopted under this situation.
- Because it is hard to induce an accurate execution time estimation model of DM jobs, this scheduling algorithm should be tolerant to approximate time estimations of DM jobs.
- To avoid the the computation bottleneck for centralized scheduling and increase the system robustness, this algorithm should be totally decentralized.
- When multiple competitive DAGs arrive at an HC system, the execution sequence of DM jobs should consider the factors of both the precedence constraints in a DAG and the arrival order of these DAGs.

Therefore, the scheduling, in fact, can be described as a problem of *dynamic scheduling for competitive DAGs*. The scheduling objective is to minimize the average makespan (the time when the last job of a DAG finishes) of competitive DAGs. This problem has been proved, in general, to be NP-complete [2], thus requiring the development of heuristic techniques [3, 4] for practical usage. In this paper we propose a scheduling framework, which satisfies the aforementioned issues.

It should be noted that the communication cost between computers within an HC system is ignored due to the following reasons: 1) the network bandwidth within an HC system 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.

3.1 Decentralized and Diligent Job Mapping

We propose a *decentralized* and *diligent* scheduling algorithm, compared with the algorithm in [3], which performs in a *centralized* and *lazy* manner. A scheduler resides on each machine. When a job Λ is finished, the scheduler on the same machine will find all the *ready* jobs (A job is *ready* when all the input data from its predecessors are available) in the successive nodes of Λ, and then map them to suitable machines immediately. The heuristic *min-min* [5] for mapping a class of independent jobs can be used to map this group of ready jobs. We call this scheduling paradigm diligent in the sense that the mapping decision is made as soon as a job is ready. The pseudo-code for scheduling algorithm on each machine of an HC system is presented in Algorithm 1.

Algorithm 1. Scheduling Algorithm on Each Machine of an HC system

```
1: if a job Λ finished on the same machine then
2:    S = {Λ' | Λ' is the successor of Λ}
3:    S' = {Λ'' | Λ'' is ready and Λ'' ∈ S}
4:    while S' ≠ Φ do
5:       according to min-min, find the best pair of job Λ'' ∈ S' and machine m, based
         on current job pending queues of each machine
6:       map job Λ'' to machine m
7:       update the job pending queue on machine m
8:       S' = S' − {Λ''}
9:    end while
10: end if
```

3.2 Job Execution Control with Priority

Usually, the pending job queue E on each machine, which stores all the waiting jobs for executing, is processed in a FIFO manner. To consider the critical path factor of a DAG, the jobs from a DAG will be executed in descending order of their estimation times. Thus, Algorithm 2 for job execution control is proposed, which supports the execution of multiple DAGs. In this algorithm when a new job arrives at a machine and suppose at the same time the machine is executing another job, it will be inserted into the job queue E at a suitable position, to keep that the jobs from *the same* DAG are arranged in descending order of their estimation times while the positions of the jobs from the other DAGs in the queue E will not be changed. Then, the jobs in the pending job queue E are processed in a FIFO manner.

Altogether Algorithms 1 and 2 form the whole heuristic scheme, which consider both the minimal completion time criterion and the critical path of a DAG. These two factors are integrated and implemented in job mapping process and job execution control, respectively.

3.3 Job Migration After Initial Mapping

The system efficiency of an HC system is defined in (1)

$$\eta = \frac{t_{computation}}{t_{total}} \tag{1}$$

where $t_{computation}$ is the system CPU time for computation and t_{total} is the total system CPU time. Then the system waste μ is defined in (2)

$$\mu = \frac{t_{idle}}{t_{total}} \tag{2}$$

where t_{idle} is the system blocking time and t_{total} is the same as the one in (1). It is clear that $\eta + \mu = 1$ because $t_{computation} + t_{idle} = t_{total}$. Furthermore, the system waste μ can be divided into two parts: the intrinsic system waste μ_i and the system waste μ_s caused by approximate execution time estimations of DM jobs.

Algorithm 2. Algorithm for Job Execution Control with Priority on machine m

```
1: loop
2:   E is the job queue on machine m
3:   if a new job Λ arrives at machine m and the machine is executing one another
     job then
4:     if |E|==1 then
5:       append Λ to the end of E
6:     else
7:       newPosition = |E| + 1
8:       for i = |E| to 2 do
9:         Λ' is the i-th element of E
10:        if Λ and Λ' are from the same DAG then
11:          if Δ(Λ', m) < Δ(Λ, m) then
12:            move Λ' to the newPosition-th position of E
13:            newPosition = i
14:          else
15:            break
16:          end if
17:        end if
18:      end for
19:      move Λ to the newPosition-th position of E
20:    end if
21:  end if
22:  if a job-finished notification received then
23:    remove the front job of E
24:    if E ≠ Φ then
25:      execute the new front job of E
26:    end if
27:  end if
28: end loop
```

Consider the HC system consisting of l machines $m_1, \cdots, m_l$ and the job pending queue (including the current executing job) on each machine is $E_1, \cdots, E_l$, respectively. $|E_i|$ $(0 \leqslant i \leqslant k)$ is the number of jobs in E_i. μ_i counts at the time that $\exists E_i$ such that $|E_i| = 0$ and $\nexists E_j$ such that $|E_j| > 1$. This kind of system waste is intrinsic, because a job is the computation atom, representing the minimal granularity for parallelization, and can be only executed on a single machine. The other kind of system waste μ_s increases while $\exists E_i, E_j$ such that $|E_i| = 0$ and $|E_j| > 1$. It is caused by the mapping decision based on inaccurate execution time estimations of DM jobs. μ_i is intrinsic, so it is unavoidable. And μ_s is seemingly also unavoidable because the task for accurate time estimation of DM jobs is so difficult. However, the technique of job migration after initial mapping can decrease μ_s. The key point of the job migration is that when $|E_i| = 0$ and $|E_j| > 1$ a suitable job Λ in E_j would migrate from m_j to m_i and begins executing immediately on m_i. The satisfying condition for migration is that $t_{completion}(\Lambda, m_j) > \Delta(\Lambda, m_i)$, which

means that the completion time of Λ on m_i is early than that on m_j. Conformed to the job execution priority in 3.2, the job in the front of the job pending queue is firstly selected to check the migration condition. Thus, a system monitor is created for the whole HC system, checks the job pending queue on each machine every T_m time units and is responsible for job migrations. The pseudo-code for this system monitor is in Algorithm 3.

Algorithm 3. Algorithm for Job Migration after Initial Mapping

while (t=the current system time) *mod* T_m=0 **do**
  receive the copy of job pending queues on each machine $\boldsymbol{E} = \{E_1, \cdots, E_l\}$
  $\boldsymbol{E}_{idle} = \{E_i | E_i \in \boldsymbol{E}$ *and* $|E_i| = 0\}$ {In $\boldsymbol{E}_{idle}$, E_i is arranged in decrease order of the computing speed of the corresponding host, which E_i is from}
  $\boldsymbol{E}_{busy} = \{E_i | E_i \in \boldsymbol{E}$ *and* $|E_i| > 1\}$ {In $\boldsymbol{E}_{busy}$, E_i is arranged randomly}
  if $|\boldsymbol{E}_{idle}| > 0$ and $|\boldsymbol{E}_{busy}| > 0$ **then**
    $\boldsymbol{E}(i)$ is the i-th element of $\boldsymbol{E}$
    for $i = 1$ to $|\boldsymbol{E}_{busy}|$ **do**
      pop the front of $\boldsymbol{E}_{busy}(i)$ {the front is running on machine i}
    end for
    $q = \sum_{E_i \in \boldsymbol{E}_{busy}} (|E_i| - 1)$
    $E_{candidate}$=null {$E_{candidate}$ is the queue of candidate jobs for migration}
    while $|E_{candidate}| < q$ **do**
      for $i = 1$ to $|\boldsymbol{E}_{busy}|$ **do**
        if $\boldsymbol{E}_{busy}(i)$ is not empty **then**
          Λ =pop the front of $\boldsymbol{E}_{busy}(i)$
          add Λ to $E_{candidate}$
        end if
      end for
    end while
    while $|E_{candidate}| > 0$ and $|\boldsymbol{E}_{idle}| > 0$ **do**
      Λ =pop the front of $E_{candidate}$
      m is the machine which owns Λ
      m' is the machine which owns $\boldsymbol{E}_{idle}(0)$
      if $t_{completion}(\Lambda, m) > \Delta(\Lambda, m')$ **then**
        Λ migrates from m to m'
        pop the front of $\boldsymbol{E}_{idle}$
      end if
    end while
  end if
end while

3.4 The Overall Scheduling Framework

In summary, the scheduling framework consists of three parts: 1) a scheduler on each machine, 2) a job execution controller on each machine, and 3) a system monitor for job migration. The job mapping process starts immediately after a job on the same machine is finished. All the ready jobs in the successors of the

finished job are mapped, by the scheduler on the same machine, to the machines according to Algorithm 1. According to Algorithm 2 the job execution controller is responsible for inserting a mapped job into a suitable position and executing them one after another. The system monitor migrates a job according to Algorithm 3. Therefore, this scheduling framework first maps jobs to machines in a decentralized and diligent manner, based on a approximate estimation of job execution time. Then the performance of this initial mapping can be improved through the use of job migration. The scheduling heuristic considers both the minimal completion time criterion and the critical path in a DAG. These two aspects are integrated and implemented in job mapping process and job execution control process, respectively.

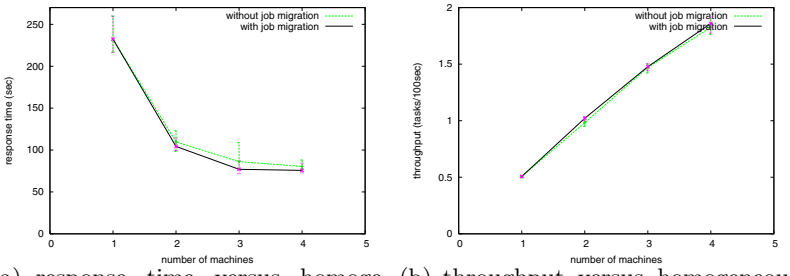
4 Experiment Procedure and Results

Based an established MAS environment named MAGE [6], we have developed a data mining HC system with the newly proposed scheduling framework. In our experiment 9 machines with different configurations are used to form this HC system. The main configurations of these machines are listed in Table 1. The performance metrics measured in the experiments include task response-time, system throughput and system efficiency defined in the following. To measure these metrics, 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 2. 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 a practical classification problem, well logging analysis to identify the pay zones of gas or oil in the reservoir formations. It contains 2000 labeled examples with 10 numeric condition attributes.

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 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^*s, which are completed by the system in a fixed time.



(a) response time versus homogeneous machines (b) throughput versus homogeneous machines

Fig. 3. The experimental results for homogeneous computing

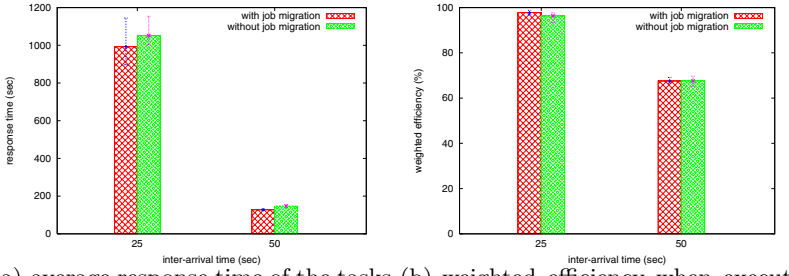
The second part of the experiments is to evaluate the scheduling performance in a heterogeneous system, which contains all the 9 machines listed in Table 1. In these experiments exponential distribution is used to generate the task sequence, including 100 G^*s. These tasks are assigned under two average task arrival intervals, t_l and t_h, where $t_l = 25$ seconds, $t_h = 50$ 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 (3), which considers the machine heterogeneity in an HC system.

$$\eta_{weighted} = \frac{t_{computation}}{t_{total}} = \frac{\sum_{i=1}^{l} \frac{t_{computation}(i)}{\rho_i}}{\sum_{i=1}^{l} \frac{t_{total}(i)}{\rho_i}} \tag{3}$$

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 performance coefficient for machine i. All the above experiments are performed under two situations, with and without job migrations after initial mapping, and repeated five times.

Figure 3(a) and Figure 3(b) show the results from the first part of experiments. Figure 3(a) illustrates that the response time of a single task 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 3(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 4(a) and Figure 4(b). In Figure 4(a) it can be found that through the use of job



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

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

Fig. 4. The experimental results for heterogeneous computing

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, as shown in Figure 4(b).

5 Related Work and Conclusions

The issues of building a computational Grid for Data Mining have been recently addressed by a number of researchers. WEKA4WS [7] 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) [8] 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 [9]. The Knowledge Grid [10, 11] 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.

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 [11]. Paper [11] also emphasizes the importance of scheduling algorithm in Knowledge Grid and uses the concept of *abstract hosts* to represent any computing host.

To the best of our knowledge, the study in this paper is the first attempt in developing a data mining HC system with an efficient and effective scheduling framework. It is formalized as a problem of dynamic scheduling for competitive DM DAGs in a heterogeneous computing system. According to the char-

acteristics of execution time estimation model of DM jobs, a new scheduling framework is presented with three features: totally decentralized, the hybrid heuristic scheme, and the use of job migration after initial mapping. The DM computing platform with this scheduling framework has been implemented in a multi-agent system environment. Its performance has also been tested by real-world datasets, which is demonstrated by our experiments. It should also be noted that the scheduling framework in this paper is a generic dynamic scheduling algorithm for DAGs, and thus has wide applicability in other fields besides data mining.

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 gratefully acknowledges the support of K.C.Wong Education Foundation, Hong Kong.

References

1. Ping Luo, Kevin Lü, Qing He, and Zhongzhi Shi. A heterogeneous computing system for data mining workflows. Technical report, Institute of Computing Technology, Chinese Academy of Sciences, 2006. http://www.intsci.ac.cn/users/luop/.
2. D. Fernandez-Baca. Allocating modules to processors in a distributed system. *IEEE Transaction on Software Engineering*, 15(11):1427–1436, 1989.
3. 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.
4. Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Poceedings of the 13th Heterogeneous Computing Workshop*, 2004.
5. Tracy D. Braun, Debra Hensgen, Richard F. Freund, Howard Jay Siegel, Noah Beck, Lasislau L. Boloni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, and Bin Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.
6. 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.
7. 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.
8. 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.

9. The Triana Problem Solving Environment. http://www.trianacode.org.
10. M. Cannataro and D. Talia. Knowledge grid an architecture for distributed knowledge discovery. *Communication of the ACM*, 46(1), 2003.
11. 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.