J Supercomput (2009) 47: 76–109
DOI 10.1007/s11227-008-0201-5

Design and implementation of a workflow-based resource broker with information system on computational grids

Chao-Tung Yang · Kuan-Chou Lai · Po-Chi Shih

Published online: 12 April 2008
© Springer Science+Business Media, LLC 2008

Abstract The grid is a promising infrastructure that can allow scientists and engineers to access resources among geographically distributed environments. Grid computing is a new technology which focuses on aggregating resources (e.g., processor cycles, disk storage, and contents) from a large-scale computing platform. Making grid computing a reality requires a resource broker to manage and monitor available resources. This paper presents a workflow-based resource broker whose main functions are matching available resources with user requests and considering network information statuses during matchmaking in computational grids. The resource broker provides a graphic user interface for accessing available and the appropriate resources via user credentials. This broker uses the Ganglia and NWS tools to monitor resource status and network-related information, respectively. Then we propose a history-based execution time estimation model to predict the execution time of parallel applications, according to previous execution results. The experimental results show that our model can accurately predict the execution time of embarrassingly parallel applications. We also report on using the Globus Toolkit to construct a grid platform called the TIGER project that integrates resources distributed across five universities in Taichung city, Taiwan, where the resource broker was developed.

Keywords Resource broker · Job scheduling · Computational grid · Information monitoring · Workflow · Globus toolkit

C.-T. Yang (✉) · P.-C. Shih
High-Performance Computing Laboratory, Department of Computer Science and Information Engineering, Tunghai University, Taichung, 40704, Taiwan, Republic of China
e-mail: ctyang@thu.edu.tw

P.-C. Shih
e-mail: shedoh@gmail.com

K.-C. Lai
Department of Computer and Information Science, National TaiChung University, No. 140, Ming-Sheng Rd., Taichung, Taiwan, Republic of China
e-mail: kclai@ntcu.edu.tw

1 Introduction

A grid is an aggregation of geographically distributed resources perhaps belonging to separate organizations with different administrators, all working together over the Internet as a vast virtual computer [1–9, 11, 12, 14, 19–22]. A computational grid is a beacon to scientists for solving large-scale problems in gene comparison, high-energy physics, earthquake simulation, weather prediction, etc. Grid computing can be defined as coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations [7–9, 11, 12]. Grid computing involves sharing heterogeneous resources, based on different platforms, hardware/software, computer architecture, and computer languages, which are located in different places belonging to different administrative domains over a network using open standards.

As more grids are deployed worldwide, the number of multi-institutional collaborations is rapidly growing. However, for grid computing to realize its full potential, it is assumed that grid participants are able to use one another's resources. In the grid environment, applications make use of shared grid resources to improve performance. The target function usually depends on many parameters, e.g., the scheduling strategies, the configurations of machines and links, the workloads in a grid, the degree of data replication, etc. The subject of this paper is the resource management for a grid system that is primarily intended to support computationally expensive tasks like simulations and optimizations on a grid. Applications are represented as workflows that can be decomposed into single grid jobs. These jobs require resources from the grid that are described as accurately as necessary.

The main task of resource management is resource brokering to optimize a global schedule for all requested grid jobs and all requested resources [16–21]. With a resource broker, users are insulated from the grid middleware, thus avoiding communicative burdens between users and resources. Resource brokers generally map application requirements to appropriate resources over multiple administrative domains. However, conventional resource brokers cannot deal with a series of problems that may contain mutual dependencies. Furthermore, when resource brokers make decisions on matching application requirements to resources, frequently network information about the resources may be ignored [9–11, 19, 20]. This paper addresses solutions for solving these kinds of problems.

In this paper, a workflow-based grid resource broker is presented whose main function is to match available resources with user requests. The broker also solves the job dependency problem by proper topological sorting and execution of workflows. A resource broker portal makes it convenient for general users to submit jobs, query resource information, and monitor job statuses. And, in order to deal with communication-intensive applications, the broker considers network information statuses during matchmaking and allocates the appropriate resources, thus speeding up execution and raising throughputs.

The contributions of this paper are described as follows. First, we construct a computational grid platform, called Taichung Integrated Grid Environment and Resources (TIGER) [25], using Globus toolkits. TIGER consists of five different schools with a total of 37 computing nodes with 50 processors. The experience of constructing this computational grid will also be discussed. Second, we describe the design and

implementation of a resource broker whose main function is to match the available resources to the user's needs. The consideration and strategy of how to design an efficient resource broker will be discussed here. Then we combine network information and machine information of the grid platform for future scheduling or benchmarking use. Next, we propose a history-based execution time estimation model to predict the execution time of parallel applications, according to previous execution results. Finally, we provide a uniform friendly graphic user interface (GUI) to use our TIGER platform to achieve automatic resource discovery and efficient resource usage. This helps grid users to submit their jobs to the suitable grid resources without knowing any resource information.

The organization of this paper is as follows. Section 2 provides background review and related work. Section 3 gives a description concerning the architecture of a workflow system, the flowchart of job execution, and the interface of a resource broker portal. The experimental results and discussion are presented in Sect. 4. Finally, conclusions and related future work are discussed in Sect. 5.

2 Background and related work

2.1 Phases of a grid resource broker

Grid resource brokering involves five main phases as follows:

1. Resource discovery, which generates a list of potential resources.
2. Information gathering, which collects dynamical resource information.
3. Application modeling, which allows the user to define the characteristics of applications.
4. Resource selection, which filters the resources that do not satisfy user requirements and enables the selection of the optimal set of resources depending on system information.
5. Job execution, which includes file transferring, precompilation, job running, and result retrieving.

These phases and the steps that comprise them are shown in Fig. 1.

2.1.1 Phase 1—resource discovery

The first phase of resource brokering involves determining which resources are available to different users, respectively. The resource discovery phase is done in two steps: resource domain definition and authorization filtering.

The first step of resource discovery is to know how many resources the grid environment has. There is no manager component for controlling resources in a grid in the current Globus toolkit version. So, we implement resource domain definition by a file of the host list to indicate the domain name of those resources. This file contains all resources users can access, but it does not guarantee resource availability.

The second step is to give the user "authorization" to navigate the entire grid. The essence of a grid is that jobs can be submitted to anywhere from everywhere. In this step, the user will have a passport to access those resources defined in the previous step. We used Grid Security Infrastructure (GSI) in Globus [11] and account

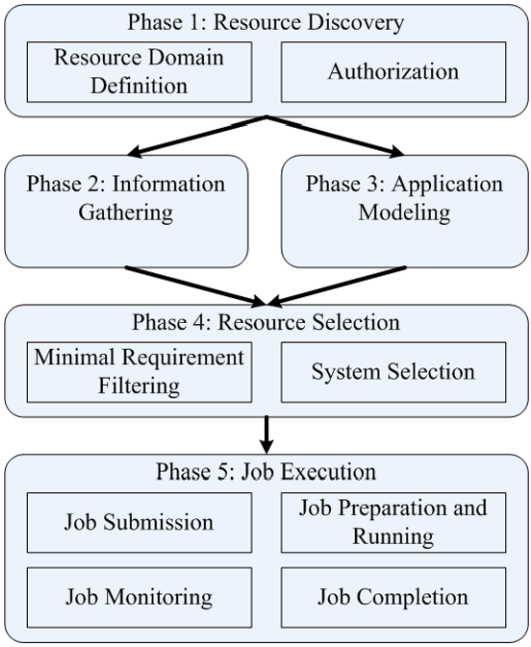

Fig. 1 Five main phases of a grid resource broker

controlling of different policies to implement this authorization. And, we used Java CoG Kit [13, 15] to develop a Java based GUI environment. The GSI uses public key cryptography (also known as asymmetric cryptography) as the basis for its functionality. A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service [11]. The Java CoG Kit [17, 18] provides access to grid services through the Java framework. Components providing client and limited server side capabilities are provided. The Java CoG Kit provides a framework utilizing the many Globus services as part of the Globus metacomputing toolkit. Many of the classes are provided as pure Java implementation, and using a client side applet without installing the Globus toolkit is possible.

2.1.2 Phase 2—information gathering

This phase is used for collecting resource information. This information is very important for the performance of the resource broker. System information collection is used to make the best possible job/resource match; detailed dynamic information about the resources is needed. That information will help the broker to determine whether the resource is available or to learn the status of the resource (Process CPU frequency, CPU utilization rate, network bandwidth, etc.). Because grid resources are dynamically changeable, real-time information allows for successful dynamical scheduling. We implement this step by using Ganglia [24] and NWS [22, 23]. The Network Weather Service (NWS) is a distributed system that detects network status by periodically monitoring and dynamically forecasting over a given time interval [23]. The service operates a distributed set of performance sensors (network

monitors, CPU monitors, etc.) from which it gathers system condition information. It then uses numerical models to generate forecasts of what the conditions will be for a given time period.

We develop a network measurement model for gathering network-related information (including bandwidth, latency, forecasting, error rates, etc.) without generating excessive system overhead. Then we consider inaccuracies in real-world network values in generating approximation values for future use. The details of the network information model will be discussed in [22, 23].

2.1.3 Phase 3—application modeling

The application model is used by users to describe characteristics and requirements of their jobs. A user can specify his preference during this phase. This model provides basic information that enables a broker scheduling algorithm to select the best strategy to distribute jobs among resources. To select suitable resources for a user's program, the user must be able to specify the minimal set of job requirements and program types. More specifically, we can divide this step into two substeps.

In the first substep, a user has to define application characteristics. Different jobs may have different running requirements. The more details are included, the less matching effort is made. Since various applications such as high energy physics and bioinformatics execute in grid computing, we must distinguish the unique factors that each application has. Most application-level scheduling models are based on the unique characteristic of each application. The following factors are the application characteristics that we consider in our resource selection strategy.

- CPU-intensive: An application requires mass computational power, and almost all the execution time is spent on CPU. It is limited by CPU speed and availability.
- Data-intensive: An application requires processing large amounts of data or generates large results, and may require distribution of those data to different resources. It is limited by network bandwidth needed to transfer those data to different resources.
- Communication-intensive: A process needs large network bandwidth or less network latency to communicate with other processes during job execution.

2.1.4 Phase 4—resource selection

This phase is responsible for selecting the most suitable resources for users. There are two main steps in this phase: minimal requirement filtering and system selection.

Step 1: Minimal requirement filtering: The first step of the resource selection phase is to filter out resources that do not satisfy the application requirements or user preferences described in phase 3. The main function of this step is to reduce the available and suitable resource set.

Step 2: System selection: As dynamic information of resources is available, system selection becomes easier and more scalable. While application requirement is defined and dynamic information of resources is available, everyone can make his own scheduling algorithm to handle different situations. We implement this step by supporting two kinds of application characteristics described in phase 3, CPU-intensive

and communication-intensive. Data-intensive characteristics will not be covered in our system Those two parts (computational grid and data grid) will be integrated in the future. Here, we describe the two resource selection strategies as follows:

- CPU-intensive: The broker selects resources based on the static CPU frequency multiplied by the dynamic average CPU free percentage over 1, 5, and 15 minutes (0% for fully busy). Therefore, the formula for CPU power is $\mathrm{CPU}_{\mathrm{power}} = \mathrm{CPU}_{\mathrm{freq}} \times (15/21\mathrm{CPU}_{1\,\mathrm{min}} + 5/21\mathrm{CPU}_{5\,\mathrm{min}} + 1/21\mathrm{CPU}_{15\,\mathrm{min}})$. We choose the best n resources according to CPU power; n is also the total number of CPU needed by users.
- Communication-intensive: The broker selects resources based on the network information modeling which will be described in Sect. 3. Therefore, the selection strategy will be covered in Sect. 3.

At the end of this step, a set of suitable resources is generated, saved as a machine list, and is ready to run jobs.

2.1.5 Phase 5—job execution

The fifth phase of grid scheduling is actually running jobs on available resources.

Step 1: Job submission: Before really running a job, the application must be submitted to the resources set described in phase 4. We perform this step by two actions. First, transfer the machine list needed for MPI [17, 19–22] to the first machine in our machine list via GridFTP, because the first is the best choice based on the application model. The second action is to transfer the application to every available resource.

Step 2: Job preparation and running: The preparation step may involve setup, compiling, or other actions needed to prepare the resource to run the application. And the program file, data file, argument file, and other settings file need to be located at the right place. Users can select uploading the application unless it has been uploaded before, which means the application is already placed appropriately. If a user selects to upload, the broker will send the compile operation to every machine and compile the source program simultaneously right after all the applications complete uploading to target resources. When compiling is finished, the broker starts to execute the applications.

Step 3: Job monitoring: While the job is running, users may want to monitor the progress of their application or may cancel or resubmit their job. Historically, such monitoring is typically done by repetitively querying the resource for status information. However, Globus is able to interrupt users when the job is finished. GRAM (Grid Resource. Allocation and Management) [11] provides basic status information such as failed, running, and finished. We also implement a job monitor web page that shows the job ID, submission time, status of job, finish time, and the result of each application.

Step 4: Job completion: When the job is finished, the user needs to be notified. The broker must be able to interrupt the user for job completion. We use the job monitor web pages to show the results to the user. Sending an e-mail or message to the cell phone of a user will be further implemented.

2.2 Related work

GRB was born in 1999 as a research project; it has been presented at Open Grid Forum and recognized by the Grid Computing Environment (GCE) research group since 2001. The GRB grid portal provides an integrated approach to grid resource management through a user-friendly web GUI and back-end GSI enabled scheduler. Among the research works focused on Grid Resource Broker (GRB) topics, in 2002 the authors in [26, 30] described the Grid Resource Broker (GRB) portal, an advanced Web gateway for computational grids in use at the University of Lecce. The portal allows trusted users seamless access to computational resources and grid services, providing a friendly computing environment that takes advantage of the underlying Globus Toolkit middleware, enhancing its basic services and capabilities. The authors' presentation shows that users do not need to learn how to use Globus, or rewrite their legacy applications.

In [27, 29], the authors describe a resource management system which is the central component of a distributed network computing system. There have been many projects focused on network computing that have designed and implemented resource management systems with a variety of architecture and services. In this paper, an abstract model and a comprehensive taxonomy for describing resource management architecture is developed. The paper presents a taxonomy for grid RMSs Resource Management Systems (RMSs). Requirements for RMSs are described and an abstract functional model has been developed. The requirements and model have been used to develop a taxonomy focused on types of grid systems, machine organization, resource model characterization, and scheduling characterization. Representative grid systems are surveyed and placed into their various categories.

A software layer that interacts with grid environments is needed to achieve these goals, i.e., middleware and its services. It is also necessary to offer resource management services to hide the underlying grid resource complexity from grid users. In [28], the authors present the design and implementation of an OGSI-compliant grid resource broker compatible with both GT2 and GT3. It focuses on resource discovery and management, and dynamic policy management for job scheduling and resource selection. The presented resource broker is designed in an extensible and modular way using standard protocols and schemas to become compatible with new middleware versions. The authors also gave experimental results to demonstrate the resource broker behavior.

Grid-enabled workflow management tools are crucial for successful building and deployment of bioinformatics workflows. We briefly review the following workflow systems: GRBWE [31], Pegasus [32], Pegasys [33]. In [31], the authors described the design and implementation of the Grid Resource Broker Workflow Engine (GRBWE) which deals with workflows described by arbitrary graphs and handles both cycles and conditions vertices; an important feature provided is recursive composition, i.e., the possibility to define a workflow vertex as a subworkflow or parameter sweep vertex instead of a batch task.

Pegasus [32] is a workflow management system designed to map abstract workflows onto grid resources, through Globus Replica Location Service (RLS), Monitoring and Discovery Service (MDS) to determine the available resources and data.

The difference between our GRB and Pegasus is that it lacks the interface with some de-facto bioinformatics tools and it does not support a monitoring tool for workflow. Pegasys [33] allows users to build direct acyclic graphs. The differences between Pegasus and our implementation are listed as below. First, the former supports only DAGs and does not include an editor for graphical composition of the workflow, but our can support an editor for graphical composition of the workflow; second, Pegasys does not exploit a Grid framework and must be installed on a cluster machine whereas our GRB scheduler supports dynamic assignment of resources belonging to a computational grid environment.

3 System design and implementation

3.1 Conceptual overview

The use of the resource broker provides a uniform interface to access any of the available and appropriate resources using the user's credentials. The resource broker runs on top of the grid middleware (such as Globus Toolkit) and serves as a link to the diverse systems available in the grid. Our grid resource broker discovers and evaluates grid resources, and makes informed job submission decisions by matching a job's requirements with an appropriate grid resource to meet user and deadline requirements. Figure 2 shows the resource broker system architecture and component relationships; the functions of each component are listed in the relation link.

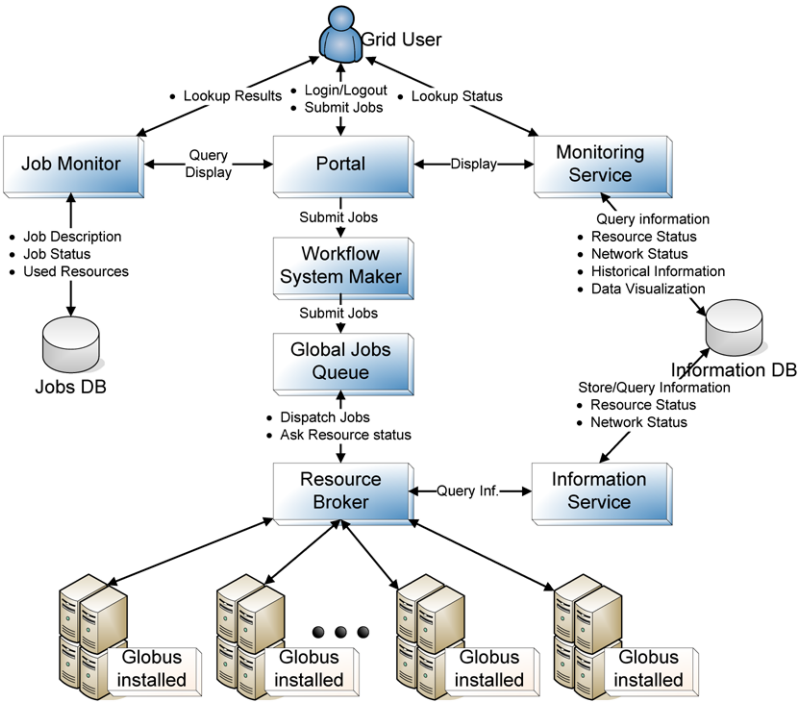

Fig. 2 System architecture

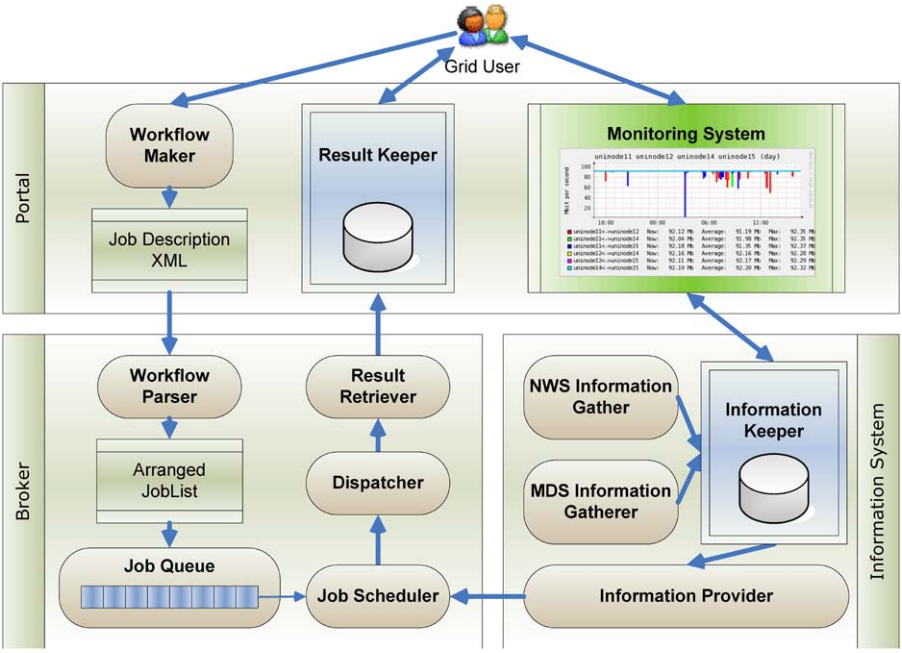

Fig. 3 Workflow system architecture

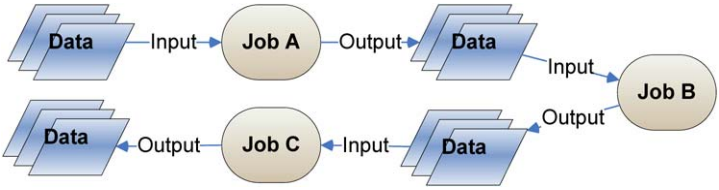

Fig. 4 Simple dependencies

A common grid portal allows easy access to the system [17, 21]. A schematic diagram of the complete workflow system is shown in Fig. 3. Most general resource brokers cannot handle jobs with dependencies, which means, for example, that Job B may have to be executed after Job A because Job B needs output from Job A as input data, as shown in Fig. 4.

The workflow-based resource broker presented in this paper copes with this in two phases: client-side phase and server-side phase. The client-side phase is a GUI Java applet that is provided in the Grid Portal for users to create workflows in workflow description language (WDL), which allows jobs with dependency and sets the following attributes for each job. We also list the names used as stored in an XML file, shown as Fig. 6, in parenthesis:

- Job name (jobname)—name of the job
- Broker sorting algorithm (sortkey)—select preferred algorithm
- Job type, parallel MPI or general sequential (runtype)—select job type

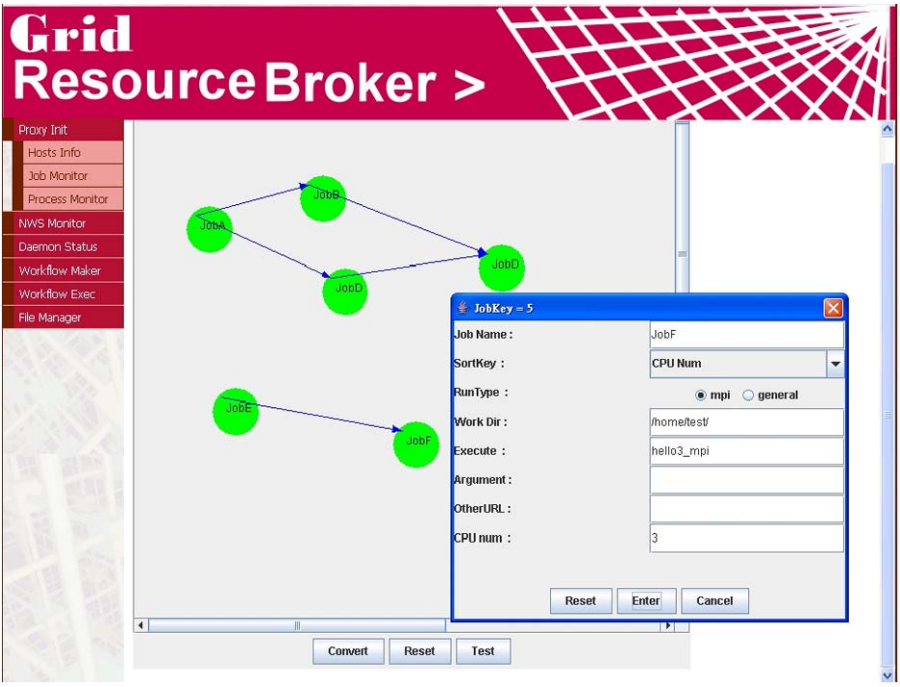

Fig. 5 Composing a workflow abstract with the Java-based workflow maker

- Job dependencies (pointed)—represent as DAG
- Working directory (workdir)—working directory of user program
- Program name (filename)—the program file name
- Argument (argu)—the arguments for the program
- Number of processors (cpuno)—number of CPU required

The workflow maker, as shown in Fig. 5, converts this workflow abstract into an actual XML file, and then delivers it to the resource broker. In the server-side phase, the resource broker portal provides a web page for receiving XML workflow files. The resource broker parses the XML file, checking all job information and dependency relationships, and then adds the job to the global job queue for execution.

3.2 Global job queue

The global job queue is responsible for holding all pending subjobs delivered to the resource broker. When the scheduler retrieves a subjob from the global job queue, it checks all node statuses, sets busy nodes to "occupied" to prevent overloading, allocates available nodes to satisfy subjob requirements, and sets these nodes to "occupied." The scheduler then gets the next subjob and repeats the procedure.

If the scheduler does not find sufficient nodes to meet job requirements, it pauses until sufficient nodes are available. When a subjob finishes, the scheduler frees the respective resources by changing their statuses from "occupied" to "available." The

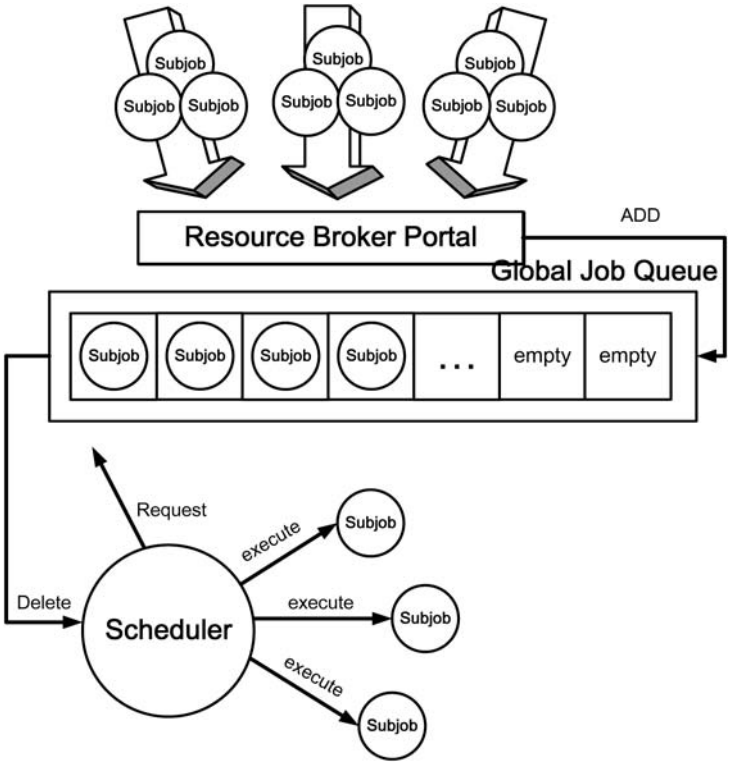

Fig. 6 Job queue architecture

scheduler also sets job run limits according to grid system capacities. For instance, if the run limit is fifty and a fiftieth job is begun, the scheduler stops retrieving subjobs from the global job queue until the number of jobs drops below fifty. Figure 6 shows the architecture of our job queue.

Figure 7 shows an example of the workflow system operation. When the job series A $\sim$ F containing dependencies is submitted, the client-side Java applet applies a topological sort. Suppose that Jobs A and E are independent of each other. The workflow system simply adds them to the global job queue for execution in parallel. When Job A finishes, it resolves its dependencies with Jobs B and C, and the workflow system adds them to the global job queue, removing Job A. When Jobs B and C finish, the workflow system then adds Job D to the global job queue for execution.

3.3 Execution time estimation model

Consider the problem of the time estimation model. First, we have to know how many variables will affect execution time when running a parallel job. Secondly, we need to know the amount of information we can obtain from historical data. In our model, there are three variables that might affect the estimated execution time: job size, quantity of processors, and processor power. In our experimental computing environment,

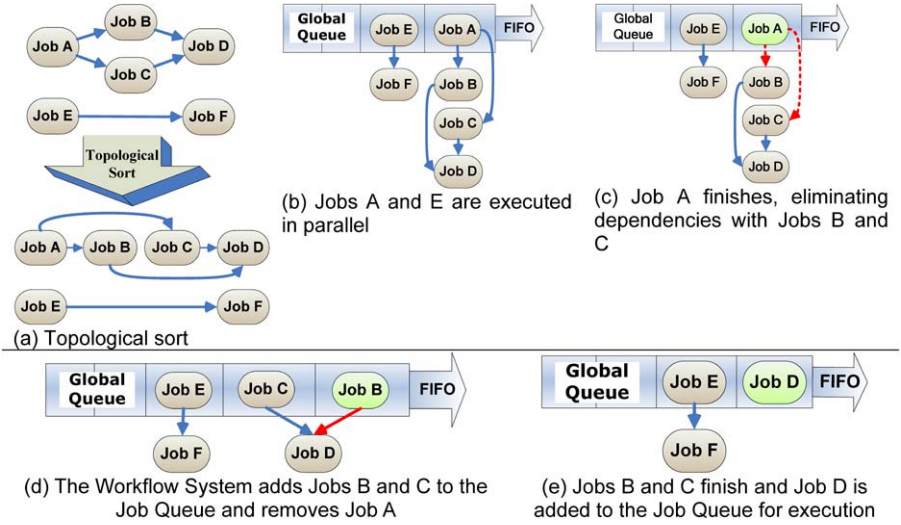

Fig. 7 The detailed steps of workflow operations

as the number of MPI jobs is divided by the number of processors, each processor will get coherent job sizes to execute. To meet the concept of heterogeneous environments, we propose the estimation model that considers the different processing power of processors to formulate total execution time (TET). We first define some terminologies that could affect the TET.

- T_mpi_{nop}: MPI start up time and Globus overhead of total *nop* processors
- S_now: The total job size for current execution
- Np_now: Number of processors for current execution
- P_now_{pn}: Processing power for processor pn, $pn = 1 \sim Np\_now$
- T_now: Total execution time (TET) for current execution
- N_pr: Number of previous results used for estimation
- S_pre_{pr}: The total job size for previous pr times execution, pr from 1 to N_pr
- Np_pre_{pr}: Number of processors for previous pr times execution, pr from 1 to N_pr
- $P\_pre_{pn,pr}$: Processing power of processor pn for previous pr times execution, pn from 1 to Np_pre_{pr}, pr from 1 to N_pr
- T_pre_{pr}: TET for previous pr times execution, pr from 1 to N_pr

The part of jobs that every processor works on is calculated by:

$$Job\_pre = \frac{S\_pre_{pr}}{Np\_pre_{pr}}. \tag{1}$$

The TET can be calculated by (the term "alpha" is defined below):

$$T\_pre_{pr} = \text{Max}\left(\frac{Job\_pre}{P\_pre_1 \times \alpha}, \frac{Job\_pre}{P\_pre_2 \times \alpha}, \ldots, \frac{Job\_pre}{P\_pre_{Np\_pre} \times \alpha}\right) + T\_mpi_{nop}$$

$$= Job\_pre \times \mathrm{Max}\left(\frac{1}{P\_pre_1 \times \alpha}, \frac{1}{P\_pre_2 \times \alpha}, \ldots, \frac{1}{P\_pre_{Np\_pre} \times \alpha}\right) + T\_mpi_{nop}$$

which can be simplified to

$$T\_pre = \frac{Job\_pre}{\mathrm{Min}(P\_pre_1, P\_pre_2, \ldots, P\_pre_{Np\_pre}) \times \alpha} + T\_mpi_{nop}. \quad (2)$$

This formula can be used to predict the next execution time. This means that the slowest processor will slow down the progress of the entire work. The TET is almost equivalent to the time that the slowest processor takes completing its job. Therefore, our estimation model is based on this particular idea. Here, we define the symbol α in the previous result.

$$\alpha = \mathrm{Ave}\left(\frac{Job\_pre_{pr}}{\mathrm{Min}(P\_pre_{1,pr}, P\_pre_{2,pr}, \ldots, P\_pre_{Np\_pre,pr}) \times (T\_pre_{pr} - T\_mpi_{nop})}\right). \quad (3)$$

The function Ave() calculates the average value of α. Finally, the estimated TET for current execution is:

$$T\_now = \frac{Job\_now}{\mathrm{Min}(P\_now_1, P\_now_2, \ldots, P\_now_{Np\_now}) \times \alpha} + T\_mpi_{nop}. \quad (4)$$

Our estimation model first gets rid of the influence of MPI and Globus overhead, in order to get the actual processor execution time. Finally, we add this overhead according to how many processors are running.

3.4 Resource selection in a communication intensive application

We used a methodology to select resources when a user requests n CPUs. Here, we describe the notation and assumptions used throughout our algorithm:

- n: total number of CPUs required
- m: total number of domains
- Num_i: number of CPUs in domain i
- $B\_in_{avg,i}$: average inner-domain bandwidth in domain i
- $B\_out_{i,j}$: bridge bandwidth from domain i to domain j

Our algorithm is based on two principles:

- Select as few domains as possible
- Select relatively higher bandwidth

The first principle is because LAN bandwidth is larger than WAN in most cases. The second is to select better network bandwidth to reduce the time complexity of the selection algorithm. Figure 8 shows the resource selection flow of our algorithm. First, we sort $B\_in_{avg,i}$ to select resources starting from high average inner-domain bandwidth. Then we search all domains for those owning more than n CPUs. If one

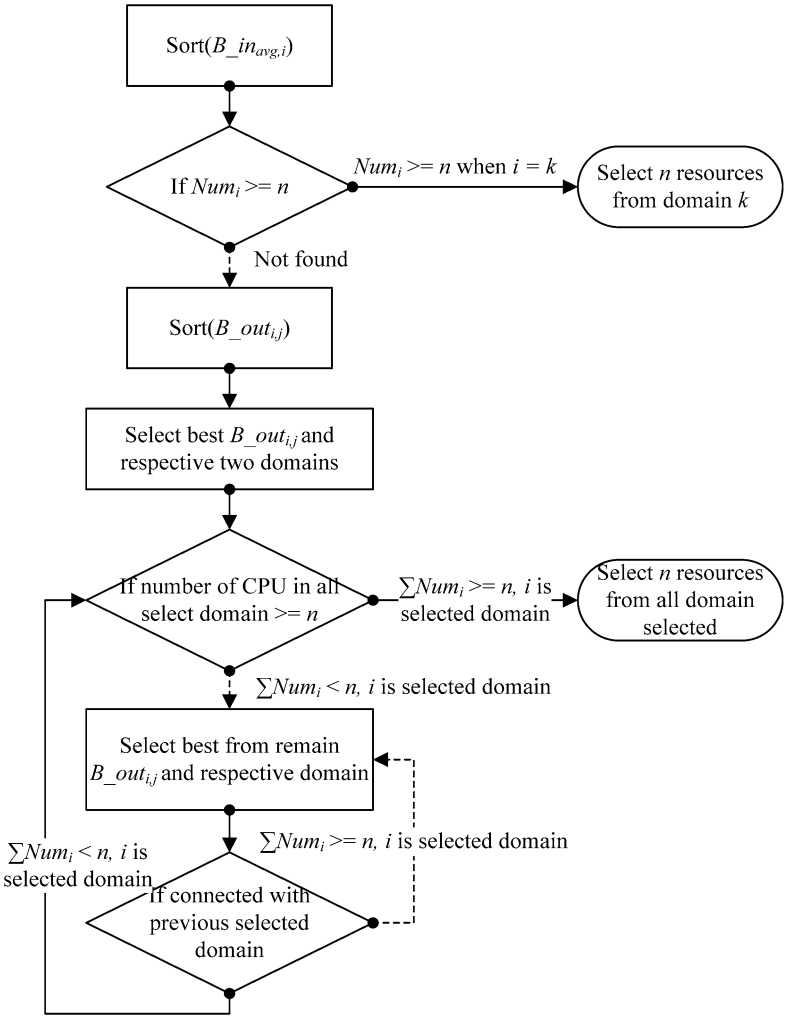

Fig. 8 Resource selection flow in a communication-intensive application

is found, select n resources from that domain. If no single domain can satisfy n, then we adapt Kruskal's minimal cost spanning tree algorithm sort to find "maximal" bandwidth with "circle."

First we sort $B\_out_{i,j}$. Then the best $B\_out_{i,j}$ and two connected domains are selected. If the number of total CPUs satisfies n, then select n resources from these two domains. Otherwise, select next best $B\_out_{i,j}$, and check whether this bridge is connected with the selected domains. If not, re-select next best $B\_out_{i,j}$. If yes, then add the original domain with a new domain connected with $B\_out_{i,j}$, and recheck whether the number of CPUs satisfies n. If yes, then select CPU from those domains. If not, select next best $B\_out_{i,j}$ and repeat those steps until it satisfies n.

3.5 Job scheduling strategy

An efficient scheduling strategy, as shown in Fig. 11, is essential for a resource broker to match jobs and resources. Our scheduling algorithm was domain-based in the experimental testbed environment shown in Fig. 12; each independent cluster was a domain. We divided a grid into n domains. Suppose a grid consists of domain 1 to domain n. Here we define some terminology:

- X: Number of processors required to execute a job
- Y: Total processors available for allocations
- S_i: Number of sites (domains) in the grid environment, $i = 1 \sim n$
- S_{RB}: Grid site where the resource broker is located
- $N(S_i)$: Number of nodes available at site i, $N(S_i) = 1 \sim m$.
- $P(S_i)$: Number of processors available at site i, where $N_i \leq P_i$, and total processors available for job execution are summed as $Y = \sum_{i=1}^{n} P(S_i)$
- E_{ij}: Graph constructed between domains i and j, the edge corresponding to the current available bandwidth reported by NWS

As shown in Fig. 9, after querying information services to get the processor loading and networking status of all grid nodes, the resource broker eliminates busy nodes to get Y. If $X > Y$, meaning the number of available processors is insufficient, matching pauses and the resource broker continues querying information services until $X \leq Y$. When $X \leq Y$, the resource broker first allocates the available processors to the S_{RB} site and initiates $S = \{S_{RB}\}$. It then sorts the weights of edges (weights correspond to current average network transmission speeds) linked to the S_{RB} site to find sites S_j and $S_j \notin S$ such that $\sum E_{ij}$ is maximum, where $\forall S_i \in S$. The resource broker then gets new site sets $S = S \cup S_j$; if $\sum P(S_i) = P(S) < X$, the resource broker continues maximizing $\sum E_{ij}$ for $\forall S_i \in S$ until $P(S) \geqq X$. Finally, the resource broker allocates X processors to S in order of speed.

```
// RB_Net Algorithm
// Parameters:
//    X is the job which requires a CPU amount
//    S_RB: Resource Broker is in this site.
Scheduler( )
{
//    Calculate the number of total available processors on all sites in grid.
      Y=∑_{i=1}^{n} P(S_i) ; for  ∀S_i ∈ G , G is a grid.
      if (X > Y) then break;
//    S is a set including the sites which should be allocated.
      S = {S_RB};
// P(S) =  ∑P(S_i)  for  ∀S_i ∈ S
      While (P(S) < X) {
            Find a site  S_j ∉ S , such that  ∑E_ij  is maximum for  ∀S_i ∈ S ;
            S = S ∪ S_j;
      }
      Allocate processors ranked in top X speed of the S.
}
```

Fig. 9 Scheduling algorithm

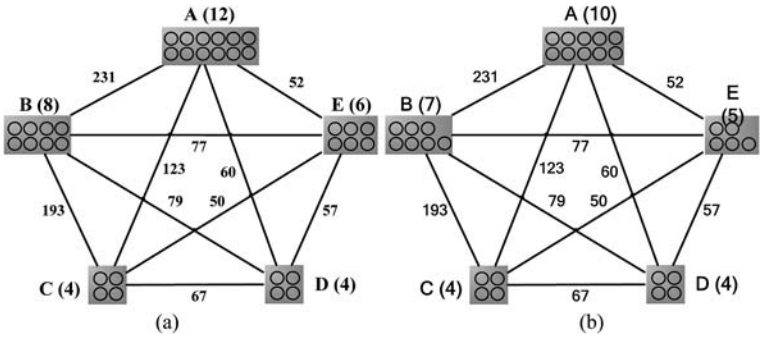

Fig. 10 (**a**) An example from our grid testbed; (**b**) after the resource broker eliminates heavily loaded nodes

An example of using this algorithm is shown in Fig. 10a. Suppose a grid consists of five domains and the resource broker is in domain A. "A(12)" means there are twelve working nodes (processors) in domain A. The number "231" represents the current communication bandwidth (Mbps) between domain A and domain B. The resource broker first queries information services to get the current status of all working nodes, then eliminates nodes with heavy loadings, as shown in Fig. 10b. A database is used to record the nodes with heavy loadings by using information services in grid resource broker.

Three possible cases can occur:

Case 1. If an incoming job needs 8 processors, the resource broker checks to see if 8 processors are available in the domain. If it finds 8 or more, the resource broker directly allocates 8 processors in order of speed to execute the job.

Case 2. If an incoming job needs 16 processors, and no one domain has that many available, the resource broker cannot directly allocate processors. It then must sort all edges between the domain in which it resides and the other domains to find the largest one ("edge" refers to current available bandwidth). When the largest (231) is found, the resource broker allocates 16 processors from domains A and B, in order of speed, to execute the job.

Case 3. If the job in Case 2 needs 20 processors and there are not enough in domains A and B, the resource broker sorts all edges connected to domains A and B until it finds domain C which, combined with domains A and B, has sufficient average bandwidth, and then allocates 20 processors from domains A, B, and C, in order of speed, to execute the job.

3.6 Flowchart

Figure 11 illustrates job execution from beginning to end in flowchart form. The numbers represent the execution sequence, the blue dotted lines interactions between user and portal, and the black lines system processes after the portal. Users need only know how to submit their jobs through the portal. This flowchart eliminates the need for users to understand how jobs are connected to resources and system intercommunications.

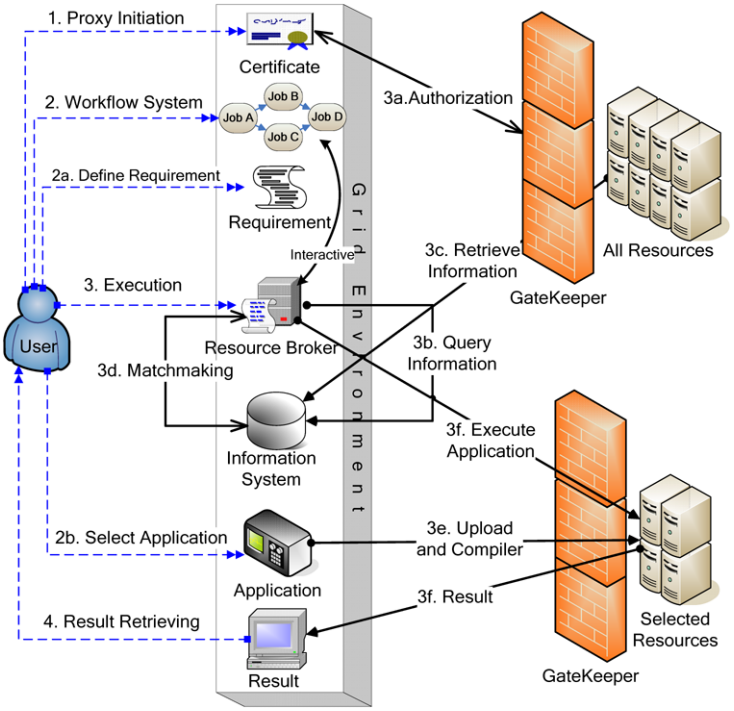

Fig. 11 Resource broker job flow

1. Proxy Initiation: Create a proxy for accessing grid resources
2. Workflow System: Resolve job dependencies
   a. Requirement Definition: User defines minimal requirements or preferences
   b. Application Selection: Select execution application
3. Execute: Execution
   a. Authorization: Use created proxy to get authorization
   b. Query Information: Query static and dynamic resource information
   c. Retrieve Information: Obtain information
   d. Matchmaking: Match user requirements to resources
   e. Upload and Compile: Prepare all applications and data on all resources
   f. Result: Generate the result and transfer it to the Portal
4. Result Retrieval: Get and display the result

4 Experimental results

4.1 Experimental environment

Our experiments were performed on a grid testbed consisting of 37 machines (50 processors) across the five schools shown in Fig. 12, Tunghai University, Providence

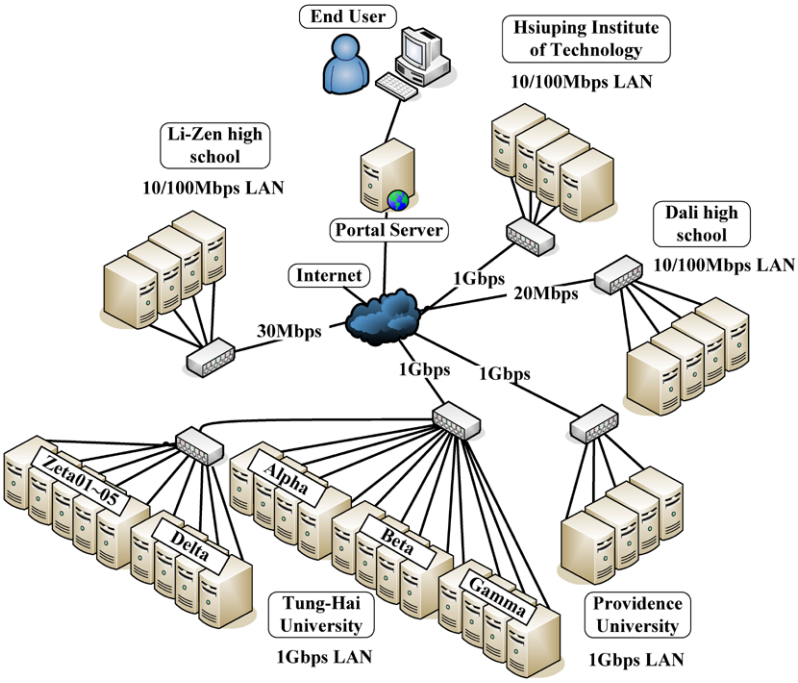

Fig. 12 The logical diagram of the test-bed experiment environment

University, Hsiuping Institute of Technology, Dali High School, and Li-Zen High School. The grid testbed specifications are shown in Table 1.

4.2 Performance of resource broker

We define the questions that we want to investigate in our grid resource broker (GRB) system:

1. What is the convenience of using our GRB?
2. What is the overhead of our GRB?
3. What is the performance of using GRB?

The first question is trivial because the time of opening web pages is much less than connection to the grid nodes, entering pass phase, looking for available resources, and submitting jobs. And it is easier to use a portal than to understand Linux instructions. So, developing a GUI based portal reduces the time and complexity of using or promoting our system. The second question is important. We first describe the experimental variables for our GGB actions.

- IST—Information search time
- BST—Broker scheduling time
- TMT—Machine file transfer time
- JRT—Job run time
- RRT—Result retrieval time

Table 1 Computing resource specifications

School	Resource	CPU type	Clock (MHz)	BogoMips	Mem	LAN	AVG LAN	Bandwidth	Linux kernel	Globus	Speed
THU	Alpha1	AMD Athlon MP	2000 × 2	3989.5 × 2	1 GB	1 G	194 M	1 G	2.6.9	3.2.1	2400 + ×2
THU	Alpha2	AMD Athlon MP	1667 × 2	3325.95 × 2	768 MB	1 G	194 M	1 G	2.6.9	3.2.1	2000 + ×2
THU	Alpha3	AMD Athlon MP	1667 × 2	3325.95 × 2	512 MB	1 G	194 M	1G	2.6.9	3.2.1	2000 + ×2
THU	Alpha4	AMD Athlon MP	1800 × 2	3325.95 × 2	512 MB	1 G	194 M	1 G	2.6.9	3.2.1	2000 + ×2
THU	Beta1	Intel Pentium D	2813 × 2	5636.4 × 2	1 GB	1 G	810 M	1 G	2.6.16	4.0.1	2.8 G × 2
THU	Beta2	Intel Pentium D	2813 × 2	5636.4 × 2	1 GB	1 G	810 M	1 G	2.6.16	4.0.1	2.8 G × 2
THU	Beta3	Intel Pentium D	2813 × 2	5636.4 × 2	1 GB	1 G	810 M	1 G	2.6.16	4.0.1	2.8 G × 2
THU	Beta4	Intel Pentium D	2813 × 2	5636.4 × 2	1 GB	1 G	810 M	1G	2.6.16	4.0.1	2.8 G × 2
THU	Gamma1	Intel Pentium 4	2806	5603.32	1 GB	1 G	770 M	1G	2.6.12	4.0.1	2.8 G
THU	Gamma2	Intel Pentium 4	2806	5603.32	1 GB	1 G	770 M	1 G	2.6.12	4.0.1	2.8 G
THU	Gamma3	Intel Pentium 4	2806	5603.32	1 GB	1 G	770 M	1 G	2.6.12	4.0.1	2.8 G
THU	Gamma4	Intel Pentium 4	2806	5603.32	1 GB	1 G	770 M	1 G	2.6.12	4.0.1	2.8 G
THU	Delta1	Intel Pentium 4	3001	5980.16	1 GB	1 G	805 M	1 G	2.6.12	4.0.1	3.0 G
THU	Delta2	Intel Pentium 4	3001	5980.16	1 GB	1 G	805 M	1 G	2.6.12	4.0.1	3.0 G
THU	Delta3	Intel Pentium 4	3001	5980.16	1 GB	1 G	805 M	1 G	2.6.12	4.0.1	3.0 G
THU	Delta4	Intel Pentium 4	3001	5980.16	1 GB	1 G	805 M	1 G	2.6.12	4.0.1	3.0 G
THU	Zeta01	AMD Athlon MP	1667	3325.95 × 2	1 GB	1 G	720 M	1 G	2.6.12	3.2.1	2000 + ×2
THU	Zeta02	AMD Athlon MP	1667	3325.95 × 2	1 GB	1 G	720 M	1 G	2.6.12	3.2.1	2000 + ×2
THU	Zeta03	AMD Athlon MP	1667	3325.95 × 2	1 GB	1 G	720 M	1 G	2.6.12	3.2.1	2000 + ×2
THU	Zeta04	AMD Athlon MP	1667	3325.95 × 2	1 GB	1 G	720 M	1 G	2.6.12	3.2.1	2000 + ×2
THU	Zeta05	AMD Athlon MP	1667	3325.95 × 2	1 GB	1 G	720 M	1 G	2.6.12	3.2.1	2000 + ×2

Table 1 (*Continued*)

School	Resource	CPU type	Clock (MHz)	BogoMips	Mem	LAN	AVG LAN	Bandwidth	Linux kernel	Globus	Speed
LZ	Iz01	Intel Celeron	898	1789.13	256 MB	10/100	85 M	30 M	2.4.20	3.2.1	900
LZ	Iz02	Intel Celeron	898	1789.13	256 MB	10/100	85 M	30 M	2.4.20	3.2.1	900
LZ	Iz03	Intel Celeron	898	1789.13	384 MB	10/100	85 M	30 M	2.4.20	3.2.1	900
LZ	Iz04	Intel Celeron	898	1789.13	256 MB	10/100	85 M	30 M	2.4.20	3.2.1	900
HIT	Gridhit0	Intel Pentium 4	2800	5586.94	512 MB	10/100	93 M	1 G	2.6.12	3.2.1	2.8 G
HIT	Gridhit1	Intel Pentium 4	2800	5586.94	512 MB	10/100	93 M	1 G	2.6.12	3.2.1	2.8 G
HIT	Gridhit2	Intel Pentium 4	2800	5586.94	512 MB	10/100	93 M	1 G	2.6.12	3.2.1	2.8 G
HIT	Gridhit3	Intel Pentium 4	2800	5586.94	512 MB	10/100	93 M	1 G	2.6.12	3.2.1	2.8 G
PU	hpc09	AMD Athlon XP	1991	3971.48	1 GB	1 G	280 M	1 G	2.4.22	3.2.1	2400+
PU	hpd10	AMD Athlon XP	1991	3971.48	1 GB	1 G	280 M	1 G	2.4.22	3.2.1	2400+
PU	hpc11	AMD Athlon XP	1991	3971.48	1 GB	1 G	280 M	1 G	2.4.22	3.2.1	2400+
PU	hpc12	AMD Athlon XP	1991	3971.48	1 GB	1 G	280 M	1 G	2.4.22	3.2.1	2400+
Dali	tc01	Intel Pentium 4	1816	3596.28	128 MB	10/100	91 M	20 M	2.6.5	3.2.1	1.8 G
Dali	tc02	Intel Pentium 4	1816	3596.28	128 MB	10/100	91 M	20 M	2.6.5	3.2.1	1.8 G
Dali	tc03	Intel Pentium 4	1816	3596.28	128 MB	10/100	91 M	20 M	2.6.5	3.2.1	1.8 G
Dali	tc04	Intel Pentium 4	1816	3596.28	128 MB	10/100	91 M	20 M	2.6.5	3.2.1	1.8 G

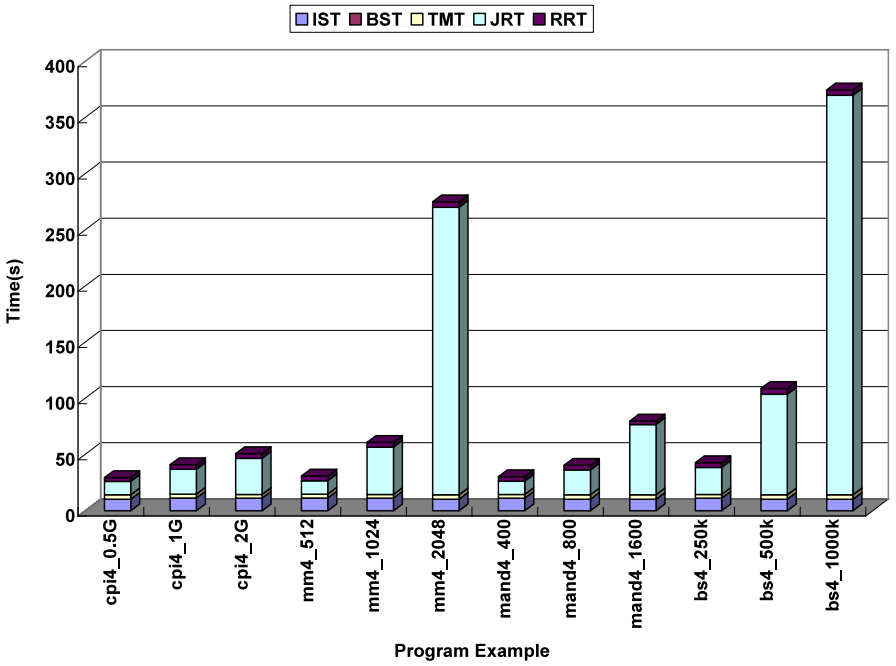

Fig. 13 Execution time distribution for 4 CPUs using a resource broker

IST is the time needed to gather system information, such as CPU speeds, loading, and free memory sizes. The broker used the information to select the best resource set according to the application model (BST), automatically transferred the selected machine files to the first resource (TMT), and then started job execution on all machines simultaneously (JRT). Upon completion, the broker transferred the execution results to the client side (RRT) and displayed them on our portal.

We can treat the time except JRT as the overhead of our system. Figures 13 and 14 show the time distribution of execution jobs with GRB. There are four program examples: PI, Matrix Multiplication, Mandelbrot Set, and Bucket Sort used for experimentation. We executed four applications with 4 CPUs and 8 CPUs. "m1_512" means matrix multiplication of a 512×512 matrix (problem size) using 1 CPU and so on. The result shows that when the broker schedules with a small problem size, the overhead seems large relative to JRT, but when the problem size is increased, the overhead decreases relative to JRT. Because grid computing is suitable for large-scale problems, our broker overhead was constant and did not increase linearly with problem size. This proves that the overhead can be ignored for large problem size. It just fits the design goal of the grid system.

Figures 15 and 16 show a comparison of total turnaround times using the resource broker and a Linux console to execute jobs. We can see that a constant but acceptable overhead exists with programs of different sizes. This result also emphasize that the overhead remains constant with different program size.

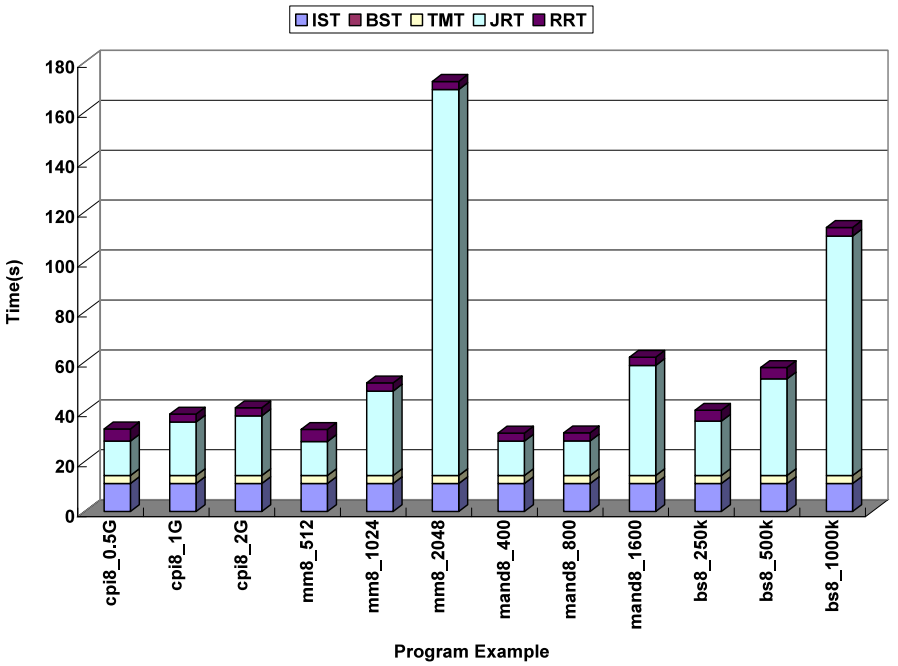

Fig. 14 Execution time distribution for 8 CPUs using a resource broker

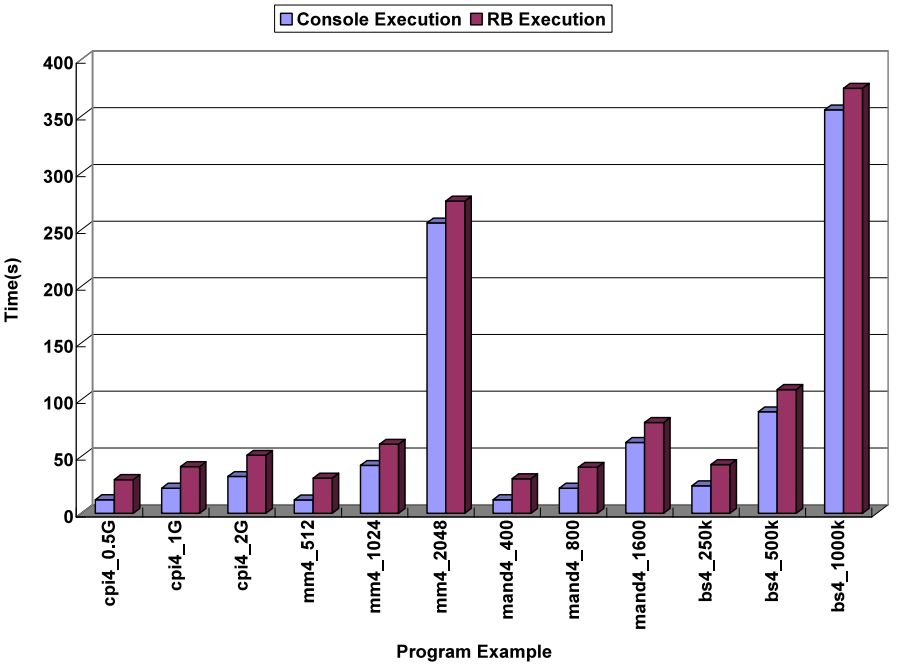

Fig. 15 Console execution vs. RB execution with 4 CPUs

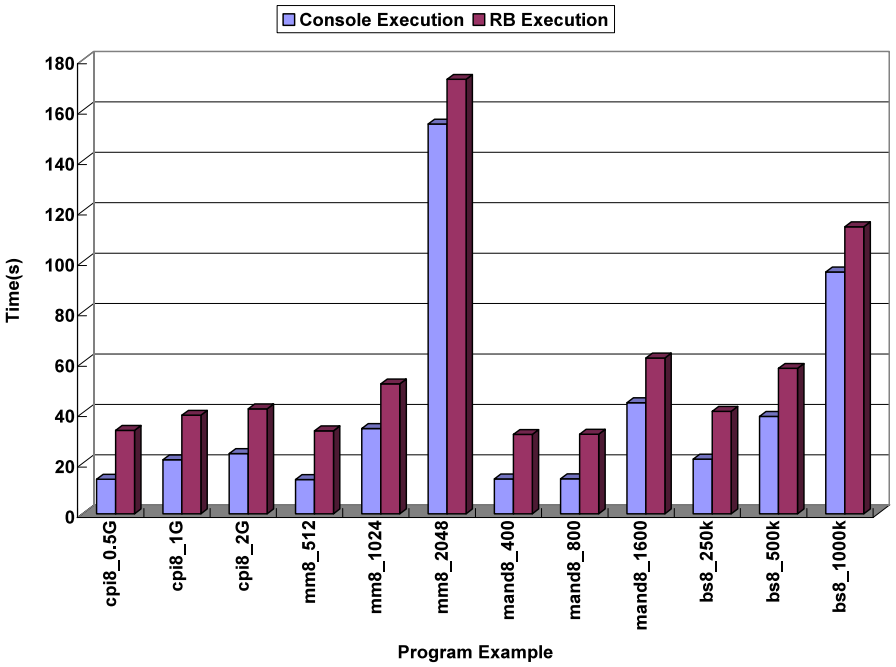

Fig. 16 Console execution vs. RB execution with 8 CPUs

In Figs. 17 and 18, RB_CPU indicates use of a CPU-intensive broking algorithm that allocates available resources according to processor speed. RB_Net indicates use of a communication-intensive brokering algorithm (addressed in Sect. 3.5) that allocates available resources according to network speed. Random means selecting resources without knowing resource information. Figure 17 shows a comparison of brokering strategies on square matrix multiplication. "mm_8_2048" means compute a 2048 by 2048 square matrix with 8 processors. The results in Fig. 17 indicate the RB_CPU strategy will obtain better performance compared with the other two because the matrix multiplication application requires a high amount of CPU power to perform multiplication operations. Therefore, it is suitable for users to select RB_CPU for GRB to select resources. Besides that, the results also show that using GRB is better than randomly selecting resources.

The comparison result of the prime number problem is shown in Fig. 18. Given a range of numbers in which we want to find a list of prime numbers, for instance, between 1 and 20,000,000, the process writes code that initially runs on a master node and sends the task of testing 101–200 range value to node 1, the task of testing 201–300 range value to node 2, and so on. The RB_CPU strategy performed better than the other two because the matrix multiplication application and prime number application both require considerable CPU power. Clearly, RB_CPU is the optimal brokering strategy.

For the third question, we compare three different kinds of broker strategies as shown from Figs. 19 to 22. Random means selecting resources randomly without

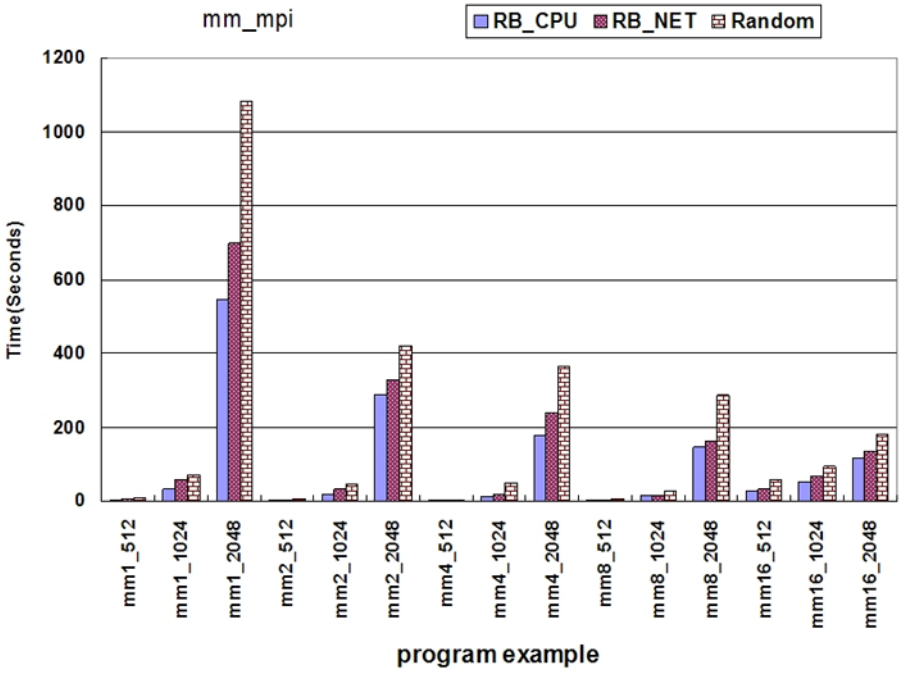

Fig. 17 Performance of broker strategy in matrix multiplication

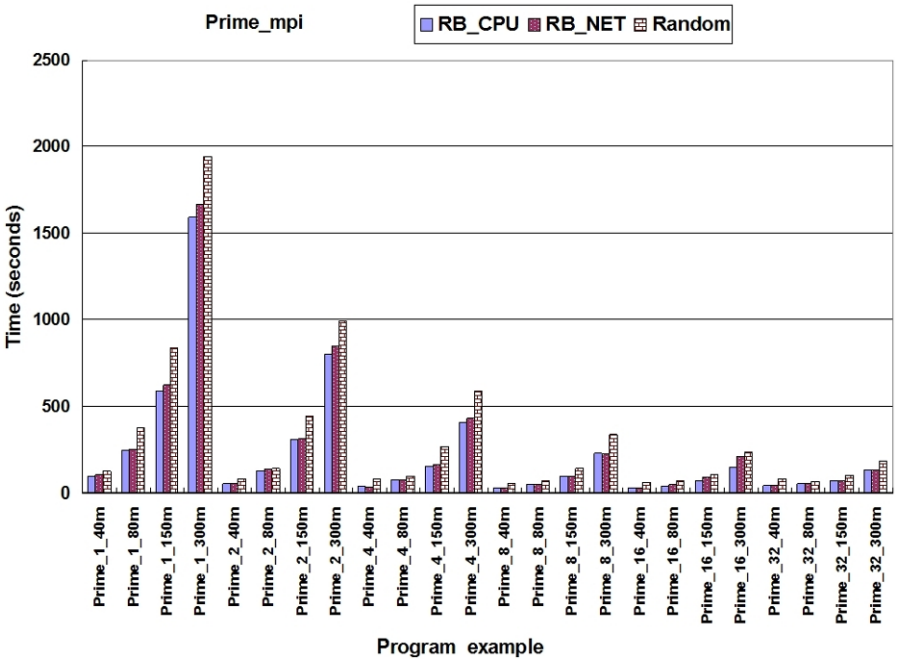

Fig. 18 Performance of broker strategy in prime numbers

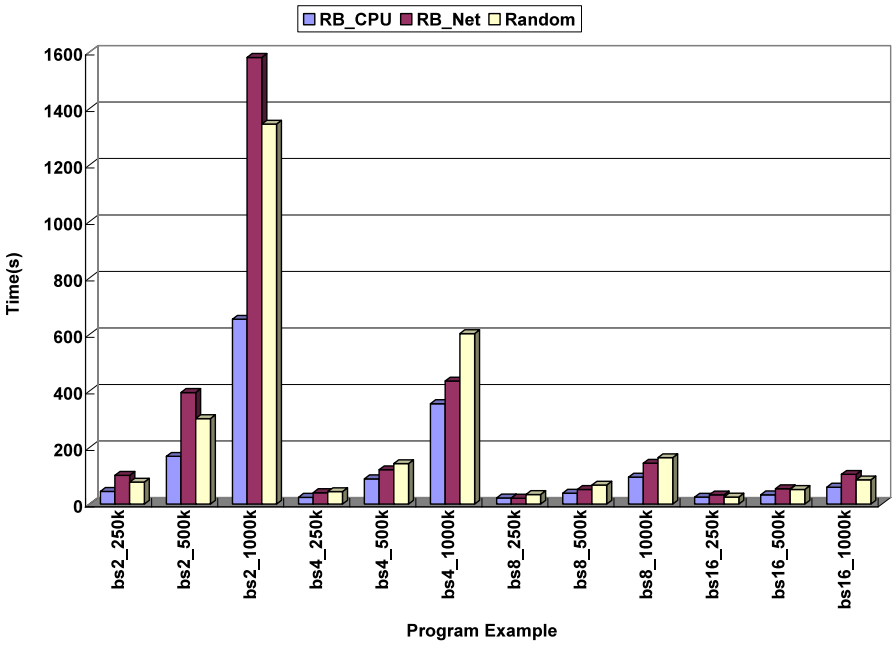

Fig. 19 Performance of broker strategy in Bucket sort

knowing resource information. Figure 19 performs the same experiment described above but for the bucket sort application. The results show the RB_CPU is still the best broker strategy. However, something unexpectedly occurred that resulted in random selection performing better than RB_Net for some cases such as bs2_1000k, bs2_500k, and bs16_1000k. We believe two reasons caused this situation. First, this situation only happened when selecting two resources at least (one is not allowed) or a maximum of 16 resources in our environment. While selecting 2 resources, we find out that RB_Net will choose resources from alpha1 to alpha4 because the average inner-domain bandwidth is the largest. But the CPU frequency is just in the middle of all resources. Therefore, a random selection will increase the chances of selecting a fast CPU. While selecting 16 resources, 16/22 resources are selected. Random still has a chance of selecting faster CPUs then RB_Net meaning that the RB_Net strategy is not appropriate for this application.

Figure 20 shows an example in which RB_Net is better than the other two. But while selecting many resources, the performance of RB_CPU and RB_Net are closed. That is because when selecting almost all resources, the function of the broker becomes useless. No broker is needed while selecting all resources. Because the degree of parallelism is limited, using resources more than the boundaries will not obtain better performance. That is the reason we need GRB to select resources. We summarize these experiments as showing that our GRB is convenient for general users to use grid environments without generating too much overhead.

Some applications require frequent communication during execution, and the resulting communication speeds may influence total execution time. Figures 21 and 22

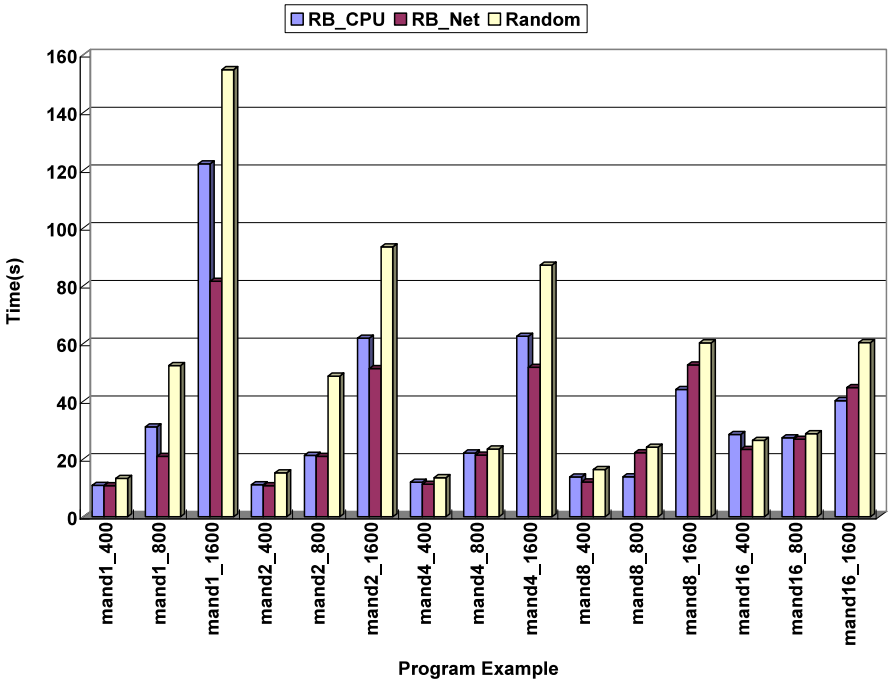

Fig. 20 Performance of broker strategy in Mandelbrot set

present an example showing that RB_Net performed better than RB_CPU or Random on Eatbw, a communication-intensive program; Eatbw_32_1024 means each of 32 processors transfers 1024 kB files. The performance of RB_CPU and RB_Net are close while selecting more and more resources. RB_CPU and RB_NET both perform better than Random in most cases. Because the degree of parallelism was limited, using more resources than the boundaries did not perform better performance. Clearly, this indicates that our resource broker can handle this type of application.

4.3 Performance of history-based model

As we mentioned before, a history-based model is used to find previous execution time records for similar applications based on certain parameters. Later, the average execution time of previous execution results is computed within a tolerable error rate, and these data are used to estimate the execution time of current parallel applications. Using empirical data analysis, it is hard to understand the behavior of performance generated by these applications. Therefore, we only consider the embarrassingly parallel jobs, which do not communicate with each other during execution. Our focus is to estimate the total execution time (TET) executed on different sets of resources with different job sizes. In order to identify jobs that are suitable for specific resources, an execution time estimation model is required.

The purpose is to describe a history-based execution time estimation model, in order to predict the execution time of parallel applications, according to previous

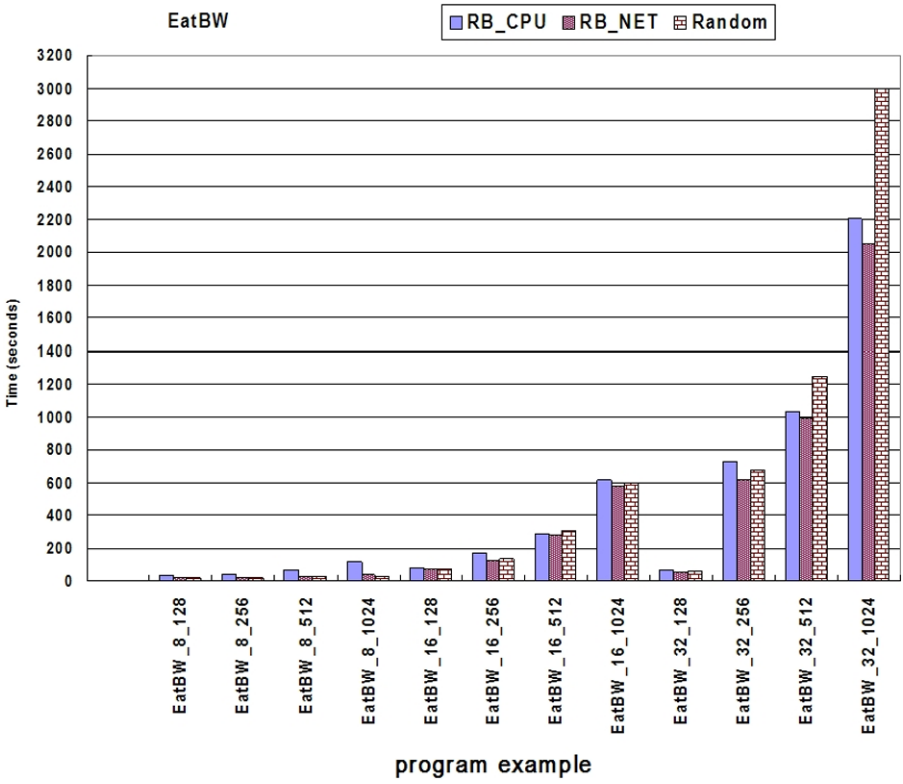

Fig. 21 Performance of broker strategy in EatBW (fixed processor number)

execution results. We focus on estimating execution time of embarrassingly parallel jobs. There are several factors which might influence the overall performance of an application in the underlying heterogeneous grid environment, such as processor power, network bandwidths, or memory sizes. A set of applications were ran on the TIGER environment we built using the standard grid middleware Globus Toolkit, and those experimental results show that our model can accurately predict the execution time of parallel applications. The average network bandwidth is about 30 Mbps over different sites. Table 2 shows the MPI and Globus overhead versus number of processors, that is, the real-world value of T_mpi_{nop}. This variable is obtained by running the hello world MPI program, which is almost the smallest program, with no CPU power needed.

During the test case, we chose three embarrassingly MPI parallel programs, which are CPI, prime number, and Mandelbrot set. These programs communicate at the start of execution, when the master node "tells" slave nodes what part of the job to handle and send results back to the master node when finishing the execution. We will describe the characteristics of each program and show the estimation results in the following section.

CPI is a program that calculates the number π accurately. It computes the value of π by numerical integration. Table 3 shows the calculated α value and estimated time

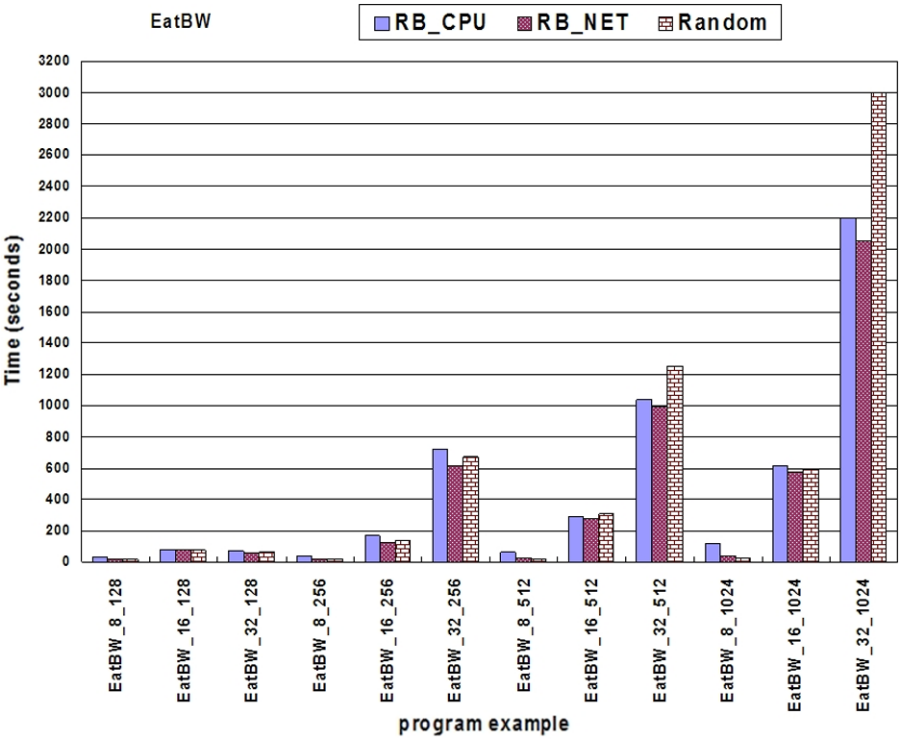

Fig. 22 Performance of broker strategy in EatBW (fixed transmission data size)

Table 2 MPI and Globus overhead vs. number of processors

N.O.P.	1	2	3	4	5	6	7	8	9	10
Time	1.10	12.56	12.40	14.47	14.15	15.73	16.23	16.93	16.97	16.86
N.O.P.	11	12	13	14	15	16	17	18	19	20
Time	17.31	17.64	18.88	20.86	23.59	26.62	26.51	26.93	27.01	27.84

Table 3 Estimation results of CPI example

α	1.17683	1.18001	1.15143	1.16017	1.16946	1.16713	1.16509
Estimate		12.932	15.067	57.099	76.283	114.379	229.138
Time	30.174	12.897	15.420	57.555	76.130	114.423	229.248
Size	100000	100000	100000	100000	100000	100000	100000
N.O.P.	3	7	6	4	3	2	1
α	1.22250	1.21813	1.21436	1.16991	1.14709	1.14499	1.14963
Estimate	14.086	18.555	27.561	45.607	18.579	22.362	28.320
Time	13.397	17.926	26.973	46.680	19.404	23.327	29.041
Size	100000	100000	100000	100000	100000	100000	100000
N.O.P.	8	6	4	2	12	10	8

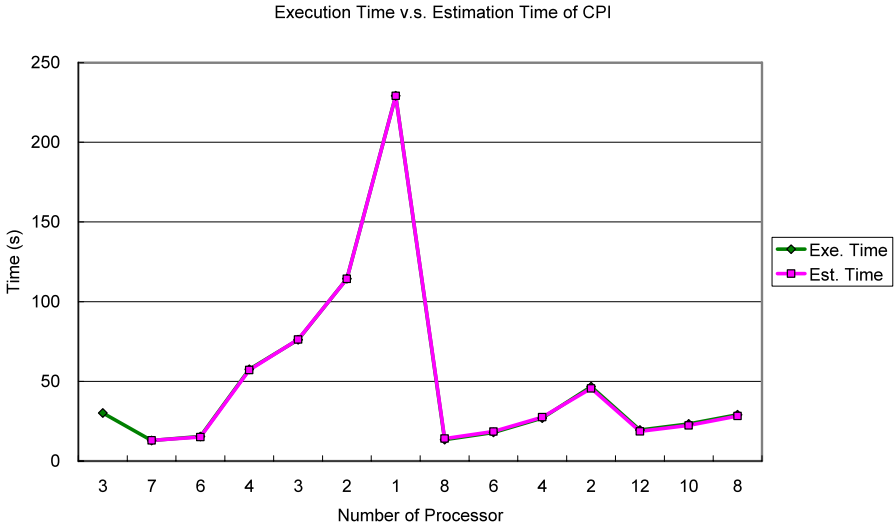

Fig. 23 Execution time vs. estimation time of CPI example

in our model. We randomly execute the CPI program on various numbers of processors with different CPU power. Top-left is the first execution result with 3 processors (N.O.P.). The second estimation is based on the first result; the third estimation is based on the first and second results, and so on. This process continues until we have more than 5 newest previous results. The size in Table 4 is irrelevant because we cannot change any parameter of this program. The value 100,000 is just a proper value that makes α more readable. Figure 23 shows that our model can precisely estimate future values of execution time. In our experiments, the average error rate is 2.12% and the maximum error rate is 5.14%. The error rate is calculated by parentage of $|(\text{Estimate time} - \text{Actual time})/\text{Actual time}|$.

In this case, we still randomly execute the prime number search on various numbers of processors with various CPU power. Table 4 shows partial results of our estimation model. The size unit is one million. Figure 24 shows that our model can still estimate the future execution time with an acceptable error. The average error rate is 28.82% and the maximum error rate is 57.84%.

The Mandelbrot set is a set of points in a complex plane that are quasi-stable (will increase and decrease, but not exceed some limit) when computed by iterating a function, $Z_{K+1} = Z_k^2 + c$. This program is particularly convenient to be parallelized for message-passing systems, since each pixel can be computed without any information about surrounding pixels. Table 5 shows the error rate that occurs on this program, and there are two size parameters that affect the TET. Figure 25 show the estimation results. The average error rate is 42.95% and the maximum error rate is 136.42%.

From the experiments, we can observe that firstly the estimation result of CPI is precisely because there is no size variable, and the workload is equally distributed to each processor. Although the CPU power is different, our estimation model still can handle it. Secondly, the prime number program possesses almost the same characteristics as CPI, except its size. The size here represents the total workload of the pro-

Table 4 Estimation results for prime number search

α	0.022972	0.018626	0.019046	0.017385	0.014597	0.012958
Estimate	26.184	47.537	39.309	59.605	107.777	176.815
Time	48.445	95.600	68.793	94.205	168.299	252.777
Size	50	80	80	100	150	200
N.O.P.	12	12	8	8	8	8
α	0.011755	0.020827	0.015487	0.013007	0.007871	0.005814
Estimate	247.810	220.066	430.018	661.217	80.443	172.756
Time	348.315	160.061	430.503	768.894	151.321	409.746
Size	250	100	200	300	100	200
N.O.P.	8	3	3	3	11	11

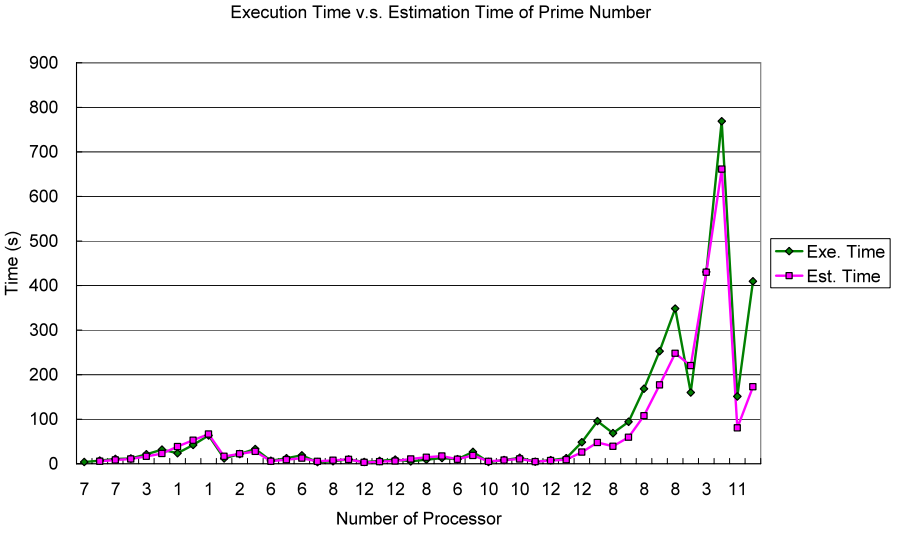

Fig. 24 Execution time vs. estimation time for prime number search

gram, which is linearly increased. Although we cannot make sure that each processor will have an equal workload (total number of prime numbers is not the same on each interval), the number of mathematical operations is about the same. Our model can estimate the execution time with an acceptable error rate.

Finally, the execution results of the Mandelbrot set program show that our prediction model is not sufficiently accurate. This happens because the two size parameters and the behavior of this program are not under control. The first parameter determines the number of iterations that the Mandelbrot set program needs to generate its graph. The second parameter defines the pixel height and width of the generated graph. The more iterations, the heavier the workload. This is also true for pixel size. However, the load is not equally distributed to each processor, since some powerful processors may be allocated a light workload and vice-versa. This is why we cannot accurately estimate execution time.

Table 5 Estimation results of Mandelbrot set

α	56191.7	59102	38593.7	54619.1	35908.2	25656.4	41945.4
Estimate		25.182	16.364	31.841	20.535	45.913	87.223
Time	33.576	23.942	24.443	23.988	23.352	56.174	68.719
Size 1	8000	6000	4000	8000	8000	10000	20000
Size 2	1200	1200	1200	1000	800	1100	1100
N.O.P.	8	8	8	8	8	11	11
α	38085.6	35261.6	39006.1	40177.7	80657.6	110259.7	133173.7
Estimate	141.703	186.279	230.371	320.351	75.536	125.993	144.321
Time	113.525	163.489	184.743	286.97	36.426	53.293	66.185
Size 1	30000	40000	50000	80000	10000	20000	30000
Size 2	1100	1100	1100	1100	1100	1100	1100
N.O.P.	11	11	11	11	11	11	11

Fig. 25 Execution time vs. estimation time of Mandelbrot set

5 Conclusions and future studies

In this paper, we design and implement a grid resource broker whose main function is to match the available resources to the user's needs. The resource broker provided a uniform interface for accessing available and appropriate computing resources. This paper presents a workflow-based computational resource broker whose main function is to match available resources with user requests and solve job dependence problems.

To identify and schedule jobs that are suitable for determined resources, an execution time estimation model is required. A history-based execution time estimation model to predict current execution time, according to previous execution results is proposed. We propose a scheduling algorithm for allocating appropriate resources

to communication-intensive applications. We also implemented a user-friendly grid portal for general users to submit jobs and monitor detailed resource statuses.

We constructed a grid platform using Globus Toolkit that integrated the resources of five schools in Taichung that have integrated grid environment resources called the TIGER project. The resource broker runs on top of TIGER. Therefore, it provides security and current information about available resources and serves as a link to the diverse systems available in the grid. The experimental results also show that users can obtain good performance with this broker to submit their applications. Experiments show that our broker is a viable contender.

In the future, a prediction model for finish times of communication-intensive jobs is needed to enhance resource utilization rates, and the scheduling algorithm of the broker must be improved. Furthermore, a database is needed to save job status and resource reliability rates in order to deal with a wider variety of jobs. Then the resource broker could query this "behavior database" for help in making decisions and to increase its efficiency in matching tasks and resources. In future studies, we plan to enhance our resource broker to support scalability and more different application characteristics to obtain better performance of different applications.

Acknowledgements The authors would like to acknowledge the National Center for High-Performance Computing for providing resources under the national project, "Taiwan Knowledge Innovation National Grid."

The work is supported in part by the National Science Council, Taiwan R.O.C., under grants no. NSC96-2221-E-029-019-MY3 and NSC96-2218-E-007-007.

References

1. Allcock B, Bester J, Bresnahan J, Chervenak AL, Foster I, Kesselman C, Meder S, Nefedova V, Quesnal D, Tuecke S (2002) Data management and transfer in high performance computational grid environments. Parallel Comput 28(5):749–771
2. Allcock B, Bester J, Bresnahan J, Chervenak AL, Liming L, Meder S, Tuecke S (2002) GridFTP protocol specification. GGF GridFTP working group document
3. Allcock B, Tuecke S, Foster I, Chervenak A, Kesselman C (2000) Protocols and services for distributed data-intensive science. In: ACAT2000 proceedings, 2000, pp 161–163
4. Allcock W, Bresnahan J, Foster I, Liming L, Link J, Plaszczac P (2002) GridFTP update January 2002, Technical Report, 2002. Available at http://www-fp.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf
5. Buyya R (1999) Deploying a high throughput computing cluster. In: High performance cluster computing, vol 1. Prentice Hall PTR, Englewood Cliffs
6. 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
7. Czajkowski K, Fitzgerald S, Foster I, Kesselman C (2001) Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE international symposium on high-performance distributed computing (HPDC-10'01), August 2001, pp 181–194
8. Foster I, Kesselman C (2003) The grid 2: blueprint for a new computing infrastructure, 2nd edn. Morgan Kaufmann, San Mateo, ISBN: 1558609334
9. Foster I (2002) The grid: a new infrastructure for 21st century science. Phys Today 55(2):42–47
10. Foster I, Karonis NT (1998) A Grid-Enabled MPI: message passing in heterogeneous distributed computing systems. In: Proceedings of 1998 supercomputing conference, 1998
11. Foster I, Kesselman C (1997) Globus: a metacomputing infrastructure toolkit. Int J Supercomput Appl 11(2):115–128

12. Ferreira L, Berstis V, Armstrong J, Kendzierski M, Neukoetter A, Takagi M, Bing-Wo R, Amir A, Murakawa R, Hernandez O, Magowan J, Bieberstein N (2003) Introduction to grid computing with globus. IBM Redbooks Press, Raleigh. Available at http://www.ibm.com/redbooks
13. Java CoG. http://www-unix.globus.org/cog/
14. Yang CT, Lai CL, Li KC, Hsu CH, Chu WC (2005) On utilization of the grid computing technology for video conversion and 3D rendering. In: Parallel and distributed processing and applications: third international symposium, ISPA 2005. Lecture notes in computer science, vol 3758. Springer, Berlin, pp 442–453
15. Laszewski V, Foster I, Gawor J, Lane P (2001) A Java commodity grid kit. Concurr Comput Pract Exp 13:645–662
16. Le H, Coddington P, Wendelborn AL (2004) A data-aware resource broker for data grids. In: IFIP international conference on network and parallel computing (NPC'2004). LNCS, vol 3222. Springer, Berlin
17. Yang CT, Shih PC, Li KC (2005) A high-performance computational resource broker for grid computing environments. In: Proceedings of the international conference on AINA'05, Taipei, Taiwan, March 2005, vol 2, pp 333–336
18. Yang CT, Li KC, Chiang WC, Shih PC (2005) Design and implementation of TIGER grid: an integrated metropolitan-scale grid environment. In: Proceedings of the 6th IEEE international conference on PDCAT'05, Dec 2005, pp 518–520
19. Nabrzyski J, Schopf JM, Weglarz J (2005) Grid resource management. Kluwer Academic, Dordrecht
20. Park SM, Kim JH (2003) Chameleon: a resource scheduler in a data grid environment. In: Proceedings of the 3rd IEEE/ACM international symposium on cluster computing and the grid, May 2003, pp 258–265
21. Yang CT, Lai CL, Shih PC, Li KC (2004) A resource broker for computing nodes selection in grid environments. In: Grid and cooperative computing—GCC 2004: 3rd international conference. Lecture notes in computer science, vol 3251. Springer, Berlin, pp 931–934
22. Yang CT, Shih PC, Chen SY, Shih WC (2005) An efficient network information modeling using NWS for grid computing environments. In: Grid and cooperative computing—GCC 2005: 4th international conference. Lecture notes in computer science, vol 3795. Springer, Berlin, pp 287–299
23. Network Weather Service. http://nws.cs.ucsb.edu/ewiki/
24. Ganglia. http://ganglia.sourceforge.net/
25. TIGER. http://gamma2.hpc.csie.thu.edu.tw/ganglia/
26. Aloisio G, Cafaro M (2002) Web-based access to the grid using the grid resource broker portal. Concurr Comput Pract Exp 14:1145–1160
27. Krauter K, Buyya R, Maheswaran M (2002) A taxonomy and survey of grid resource management systems for distributed computing. Softw Pract Exp 32:135–164
28. Rodero I, Corbalán J, Badia RM, Labarta J (2005) In: eNANOS grid resource broker. LNCS, vol 3470. Springer, Berlin, pp 111–121
29. Venugopal S, Buyya R, Winton L (2006) A grid service broker for scheduling e-science applications on global data grids. Concurr Comput Pract Exp 18:685–699
30. Aloisio G, Cafaro M, Carteni G, Epicoco I, Fiore S, Lezzi D, Mirto M, Mocavero S (2007) The grid resource broker portal. Concurr Comput Pract Exp 19(12):1663–1670
31. Cafaro M, Epicoco I, Mirto M, Lezzi D, Aloisio G (2007) The grid resource broker workflow engine. In: Proceedings of the sixth international conference on grid and cooperative computing (GCC 2007), 2007
32. Deelman E, Singh G, Su M, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, Laity A, Jacob JC, Katz DS (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Sci Program J 13(3):219–237
33. Shah SP, He DYM, Sawkins JN, Druce JC, Quon G, Lett D, Zheng GXY, Xu T, Quellette BFF (2004) Pegasys: software for executing and integrating analyses of biological sequences. BMC Bioinform 5:40

Chao-Tung Yang is a professor of computer science and information engineering at Tunghai University in Taiwan. He received a B.S. degree in computer science and information engineering from Tunghai University, Taichung, Taiwan in 1990, and the M.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan in 1992. He received the Ph.D. degree in computer and information science from National Chiao Tung University in July 1996. He won the 1996 Acer Dragon Award for 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. 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 high-performance computing, and internet-based applications. He is both member of the IEEE Computer Society and ACM.

Kuan-Chou Lai received his M.S. degree in computer science and information engineering from the National Cheng Kung University in 1991, and the Ph.D. degree in computer science and information engineering from the National Chiao Tung University in 1996. Currently, he is an assistant professor in the Department of Computer and Information Science at the National Taichung University. His research interests include parallel processing, heterogeneous computing, system architecture, grid computing, P2P systems, and multimedia systems. He is a member of the IEEE Computer Society.

Po-Chi Shih received the B.S. and M.S. degrees in computer science and information engineering from Tunghai University in 2003 and 2005, respectively. He is now studying Ph.D. degree at computer science in National Tsing Hua University. His research interests include parallel processing, heterogeneous computing, grid computing, and P2P systems.

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.