Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A flexible architecture for job management in a grid environment[*]

LUAN Cui-ju[†1,2], SONG Guang-hua[2], ZHENG Yao[2], ZHANG Ji-fa[2]

([1]*College of Information Engineering, Shanghai Maritime University, Shanghai 200135, China*)

([2]*School of Computer Science and Center for Engineering and Scientific Computation, Zhejiang University, Hangzhou 310027, China*)

[†]E-mail: cjluan@cie.shmtu.edu.cn

Received Apr. 24, 2006; revision accepted Sept. 28, 2006

Abstract: Job management is a key issue in computational grids, and normally involves job definition, scheduling, executing and monitoring. However, job management in the existing grid middleware needs to be improved in terms of efficiency and flexibility. This paper addresses a flexible architecture for job management with detailed design and implementation. Frameworks for job scheduling and monitoring, as two important aspects, are also presented. The proposed job management has the advantages of reusability of job definition, flexible and automatic file operation, visual steering of file transfer and job execution, and adaptive application job scheduler. A job management wizard is designed to implement each step. Therefore, what the grid user needs to do is only to define the job by constructing necessary information at runtime. In addition, the job space is adopted to ensure the security of the job management. Experimental results showed that this approach is user-friendly and system efficient.

Key words: Grid, Job management, Job definition reuse, Steering of job transfer, Job space
doi:10.1631/jzus.2007.A0095 **Document code:** A **CLC number:** TP393

INTRODUCTION

The Grid is capable of coordinated resource sharing and problem solving in dynamic and multi-institutional virtual organizations (Foster *et al.*, 2001). As a kind of grid, the computational grid provides a new method to deal with engineering and scientific computation, and by using it we launch the Multidisciplinary ApplicationS-oriented SImulation and Visualization Environment (MASSIVE) project (Zheng *et al.*, 2004). The aims of the MASSIVE project are to use Grid technology to establish an enabling environment for distributed simulation and visualization of large-scale scientific and engineering research.

The MASSIVE provides a problem solving environment. It integrates the grid-enabled services, which hide the complexity of the computational grid so that scientists can concentrate on the problems to be solved, and need not be concerned with the details of the grid. As concerning the grid, what the scientists need to do is to define, submit, and monitor jobs, which shows that most of their work is dealing with the jobs in the grid. Hence a mechanism is required to help the user managing the jobs to make the grid-enabled services easy-to-use, and consequently to improve efficiency.

In this paper we present a flexible architecture of job management adopted in the MASSIVE project. The job management system is job-centric. It provides a mechanism of job definition reusability. Moreover, the system provides a set of flexible file operations, which can be described in the job definition and be performed automatically. Furthermore, the system provides frameworks for job monitoring and job scheduling. With this job management, the users can manage and perform their jobs conveniently and efficiently.

The rest of this paper is arranged as follows. Section 2 introduces the MASSIVE project briefly,

[*] Project supported by the National Natural Science Foundation of China (No. 90412014), the National Science Foundation of China for Distinguished Young Scholars (No. 60225009), and the China Next Generation Internet (CNGI) Project (No. CNGI-04-15-7A)

while Section 3 details the architecture of the job management. Section 4 discusses a few problems in the implementation of the job management in detail. Section 5 presents an experiment to illustrate the characteristics of the job management. Finally, we conclude with a summary of our work and an outline of our plans for continuing research in Section 6.

ANALYSIS OF THE JOB MANAGEMENT IN THE MASSIVE ENVIRONMENT

Outline of the MASSIVE project

The MASSIVE Environment is developed and deployed at the Center for Engineering and Scientific Computation (CESC), Zhejiang University. It provides a grid-based platform for geometrical modeling, discretization, scientific computing and visualization. Its advanced services can be accessed easily in a visual manner. The integration of services as a part of a workflow process enables the creation of services that can be easily reused by the community. Scientists are then in a position to concentrate on the science, while platform developers can focus on the delivery of services that can be assembled as building blocks to create more elaborate services.

The grid created for the MASSIVE project utilizes resources located at Zhejiang University. The grid currently uses the Globus Toolkit 2.4.3 (GT2) (Foster and Kesselman, 1997) as the middleware to enable the resources at each site to be accessed in a secure and uniform manner.

In the MASSIVE project, the grid enables the essential aspects of industrially relevant Computational Fluid Dynamics (CFD) and Computational Solid Mechanics (CSM) by using a services-oriented architecture based on emerging standards such as the OGSA. Services for geometry preprocessing and mesh generation, the migration and execution of application programs on remote platforms, collaborative visualization, and data analysis, form the basis of a globally distributed Virtual Organization (VO) (Foster *et al.*, 2001), in which security and performance are key concerns. A typical usage would be generation of meshes using the meshing service on an SGI IRIX platform, solution of a CFD/CSM problem on a PC Cluster with the meshes previously created, and collaborative visualization of the numerical results with equipment, such as a display wall and a BARCO stereo projector, at the CESC, Zhejiang University.

A key issue in the MASSIVE project is the capability for effectively managing jobs, which involves file operation, job definition, job scheduling, job execution, job monitoring, and so on. The goal of the job management in the MASSIVE is to make the grid environment easy-to-use and efficient.

Requirements for the job management

Generally speaking, in the area of engineering and scientific computing, one problem needs to be solved many times in order to get a reasonable result, while the difference is usually just a few parameters. To meet this kind of needs, there should be some mechanism to realize job definition reuse, which means that once a job is defined it can be reused later on, and what the user needs to do is only to change some parameters. Furthermore, the job definition reuse must be flexible and easy-to-use.

Large-scale CFD and CSM simulations are computationally intensive, and may involve access to resources that are intrinsically distributed. For example, in the case of an organization in which multiple partners from industry and academia are cooperating to design and build a complex system that requires CFD/CSM simulations, the geometry of the component may be created at one location, a mesh conforming to this geometry may be generated at a second location, and a CFD/CSM simulation based on the mesh may be performed at a third location. Finally, the output from the simulation may be analyzed and visualized at one or more other locations.

There are many data or files to be transferred among the resources. It will be very fussy, if all the file operations are carried out manually. And it will take much time during the interactive operation. Moreover, the output cannot be transferred to the destination locations in time with this mode. So, there must be a mechanism to describe the file operations prior to and following job execution when defining the job. And they can be carried out automatically without the user intervention.

In addition to the file transfer, various types of file operations are involved in the grid environment, such as creating directory, deleting file, checking whether a file exists, determining the file property, and so on. Sometimes the object is folder, so all the corresponding folder operations must be provided. In order to ensure the integrity of the job definition, the file/folder operations should be included in it.

The input and output data of large-scale CFD and CSM simulations are very large. So the quantity of the data transferred among the resources is very large. Therefore, the file transferring performance will affect the whole performance of the job obviously. Because the grid is dynamic, so is the network and the performance of the same transfer mode. Therefore, the file transfer performance needs to be monitored, so that the file transfer mode can be modified dynamically to improve the efficiency.

The applications in the computational grid are not the same type. Different applications have different characteristics, and their demands on the resources may differ greatly. In order to be adaptive to the applications, the scheduling algorithm must take the target applications into account and determine the scheduling policy accordingly.

The grid middleware used in the MASSIVE environment is GT2. Globus (Foster and Kesselman, 1998) uses the Resource Specification Language (RSL) to describe the resources when submitting a job. The RSL is a powerful and flexible language, but it needs the user to understand the complex details about the grid. Although the RSL files can be reused, it is inconvenient to manage them orderly. For example, the user cannot use the identifier (job ID or job name) to search a job with the GUI. Moreover, some complex files operations, such as creating directory, third-party or parallel transfer, cannot be described in the RSL and have to be done manually, and the folder operations are not supported.

Other grid middlewares, such as Gridsolve (YarKhan *et al.*, 2006), Condor (Tannenbaum *et al.*, 2002), SAMGrid (Baranovski *et al.*, 2004) and CREAM (Andreetto *et al.*, 2006), etc., seem to be paid more attention to job reusability and flexibility in file operations.

Characteristics of the job management

Based on the requirements analyzed above, the job management model we have devised has the following characteristics:

(1) All the operations provided by the job management system can be accessed easily in a visual manner.

(2) A flexible, easy-to-use, and effective job definition reuse mechanism is put forward.

(3) The system provides various types of file/folder operations.

(4) The file/folder operations can be described in the job definition, and can be executed automatically.

(5) The file/folder operations can be monitored. The file transfer performance can be visualized, so the user can steer the file transfer visually.

(6) The scheduling policy is application adaptive.

(7) Jobs are saved in queues, from which the scheduler access to the waiting jobs.

(8) Our model is able to distribute a given job on multiple resources available in the Grid.

JOB MANAGEMENT ARCHITECTURE

In this paper, the job is defined as an object running on the grid in order to solve a certain problem. This kind of object has many properties, such as ID, name, type, etc. Furthermore, they designate the resources to be used, the executable file or command to be executed on the remote resources, the input and output data or files, and the manner in which to deal with the results, and so on. The methods operating on them include defining, scheduling, submitting, executing, monitoring, steering, etc.

The job management components include job creator, job scheduler, job executor, job monitor and file controller. Its architecture is shown in Fig.1.

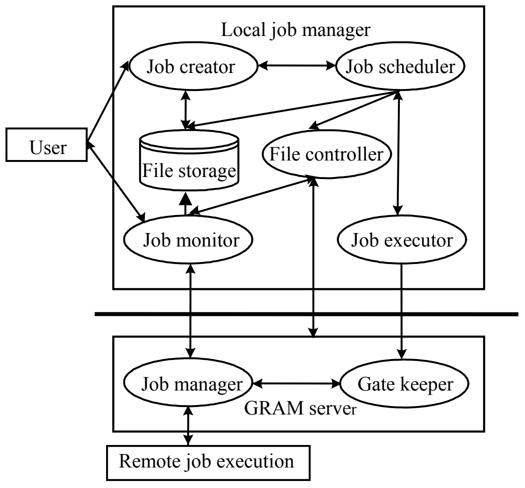


Fig.1 Architecture of the job management

Job creator

Every job has to be described before being executed on the grid. The job creator is just used to collect the information on the jobs from the user with GUI. The job creator provides two modes of definition, which can meet the different needs of the users. One is the simplified definition, which can collect the basic information of the job, such as the job name, job type, process count, executable file, running parameters, working directory, etc. It can be used if there are no files/folders to be transferred automatically. The other is the advanced definition, which can collect the information about resource filtering, complicated file/folder operations, and other advanced information besides the basic one. If there are files/folders to be transferred automatically or if the job has special resource requirement, it can be used to define the jobs. To reuse the job definition, all the job information will be described in an XML file. The job creator will assign a job ID to every job, which uniquely identifies a job at the client side.

Job scheduler

The function of the job scheduler is to allocate physical resources to the jobs. The available resources come from the VOs, to which the user belongs. Resources are discovered and published by the Extensible Monitory and Discovery Service for the MASSIVE Grid (MEMDS) (Wang *et al*., 2005), filtered by the user's resources requirement, and then allocated to the job.

The physical resources are those that locate in the individual Grid sites and can execute the Grid jobs. While the logical resources are the virtual resources that meet the requirement of resource filtering conditions. The logical resources are allocated by the job creator, and should be converted to physical resources that exist in VOs by the job scheduler. This conversion procedure comprises two phases: filtering phase and allocating phase (Liu *et al*., 2002). In the filtering phase, the system will remove any resource not meeting the filtering conditions, and the remainder constitute the waiting scheduled resources pool. In the allocating phase, resources in the pool can be allocated to the jobs.

The job scheduler provides two methods to allocate the physical resources: manual and automatic. That is, to allocate the resources by the user manually or by the system automatically. According to the resources information provided by the MEMDS, the users can designate the appropriate resources to execute their jobs. The system can also schedule the resources automatically by the scheduling algorithms based on the resource information. At present, the system provides two automatic resource scheduling algorithms, the heuristic-based stochastic scheduling algorithm (HSSA) and the heuristic-based greedy scheduling algorithm (HGSA) (Luan *et al*., 2006), to allocate the resources automatically.

The merit of the two-phase resources allocation is, when the VOs or physical resources changed, the only thing that needs to be done is to reconstruct the mapping from the logical resources to the physical ones.

Job executor

The job executor's function is to accept the jobs and execute them. It adopts the first come first served (FCFS) algorithm to determine the executing order of the jobs.

The job executor constantly receives messages sent by the job scheduler. When receiving an executing request, it will create a new thread to send the job designated in the message to the gatekeepers (Czajkowski *et al*., 1998) on the remote resources. After getting a job identifier (i.e. the handle to the job at the server side) as the response, the job executor will send it to the corresponding thread of the job scheduler. The job handle will be used to monitor the job status and control the job.

Job monitor

Because of the differences between the monitoring information of the architecture and the job, there are two components to be used to monitor them respectively. MEMDS is responsible for discovering and monitoring the resources. The job monitor is used to monitor the jobs.

Except the whole job status, the job monitor can provide the detailed information on job processes, at the same time the job can be steered at the process level. Moreover, it can monitor and steer the file operations. Besides steering the applications, the user can use the job monitor to change the mode of the file transfer, such as changing the parallelism to improve the performance.

File controller

In the grid environment, except transferring and deleting files, checking whether the files exist or whether their properties meet the needs is useful for the fault tolerance. Therefore, the job management should provide comprehensive file operations. Moreover, in the MASSIVE project, there are potentially several related input or output files in a job. To ease the work, folder operations are required in the job management.

The file controller provides all kinds of file/folder operations needed in the grid environment, which are implemented on top of the GridFTP and GASS in the Globus toolkit. The file controller provides capabilities for the job management to transfer files/folders securely, conveniently, efficiently and flexibly. The files/folders can be transferred by multi-channels or in third-party style. Partial file transfer has also been implemented.

Whenever a job deals with file/folder operations, the file controller will be invoked. The job creator provides an interface to define file/folder operations before and after job execution. With the functions provided by the job management, it is easy to share the data, software, storage and computing resources located at different sites.

IMPLEMENTATION OF THE JOB MANAGEMENT

Job reusability

From the analysis in Section 2, the job definition reuse is important in the engineering and scientific computing. In order to reuse the jobs, the system constructs a history job list, which contains all the defined jobs, including the running jobs, the finished jobs, and the to-be-scheduled jobs.

Thus, there may be many jobs in the memory, but only a few jobs to be reused. To save the memory, the job management adopts a two-level storage structure. The first level stores the necessary and index information on the jobs in random file, while the second level stores the detailed information in XML format, which is shown in Fig.2. The history job list in the memory contains the information stored in the first level storage. Only when a job in the list is called, its detailed information is read in the memory. If a job needs to be reused, what the user should do is to select the job in the history job list by the job ID or the job name, and submit it after editing it or not.

To ease the work, the operations of job management are organized as a workflow. The job information is carried on from the first operation to

```
<?xml version="1.0"?>
<vjob>
  <basic>
    <id></id>
    <name></name>
    <type></type>
    <process></process>
    <status></status>
    <deftime></deftime>
    <starttime></starttime>
    <finishtime></finishtime>
    …
  </basic>
 <rsl>
  <clause>
   <resourceManagerContact qmark="1">
   </resourceManagerContact>
   <count></count>
   <jobType qmark="1"></jobType>
   <lable qmark="1"></lable>
   <environment></environment>
   <arguments></arguments>
   <directory qmark="1"></directory>
   <executable qmark="1"></executable>
   …
  </clause>
  …
 </rsl>
 <filter>
   <ostype></ostype>
   <cpucount></cpucount>
   <cpuspeed></cpuspeed>
   <cpufreerate></cpufreerate>
   <memtotal></memtotal>
   <memfree></memfree>
   <filesystemtotal></filesystemtotal>
   <filesystemfree></filesystemfree>
   …
 </filter>
 <files>
  <send>
   <transfer>
    <srcRM></srcRM>
    <desRM></desRM>
    <srcpath></srcpath>
    <despath></despath>
   </transfer>
  </send>
  …
 </files>
 <folders>
  …
 </folders>
 <resource>
  <item>
   <vo></vo>
   <id></id>
   <processcount></processcount>
   …
  </item>
  …
 </resource>
 …
</vjob>
```

Fig.2 An example XML file for a job

the last one. Within the workflow the job definition can be saved, closed or deleted at any time. Hence the job can be halted and restarted later.

In the workflow, jobs are saved in one of the three job queues, the waiting, scheduled and finished queues, according to their status. The waiting queue keeps the un-submitted jobs. The scheduled queue saves the running jobs. And the finished queue contains the jobs that have been executed successfully or failed. The workflow diagram of the job management is illustrated in Fig.3.

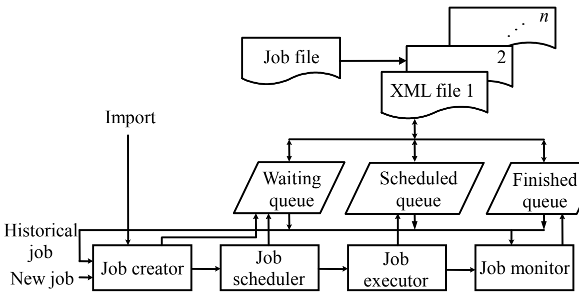


Fig.3 Workflow of the job management

After getting adequate information from the job creator, the job will be put in the Waiting Queue. As long as it is allocated with physical resources and sent to the job executor, the status of the job becomes running and it will be put in the Scheduled Queue. When the job monitor finds it finished, it will be moved to the Finished Queue. At this point of time the entire process is completed, and the renewed information about the job will be saved. There are three approaches that a job can enter this workflow. The first is to create a new job completely, the second is to reuse a historical job listed in the history job list, and the last is to import a job from outside of the system. The second one is to just adapt to the job reuse.

At the client side, multiple jobs can be defined at the same time, so there may exist more than one job flow in the system. They are independent of each other and identified by the job ID, hence the jobs are manipulated in their own flows. Using the workflow to conduct the whole process makes the complicated operations easy to perform, and the creating and reusing of jobs are unified in the same flow.

Adaptive job scheduler for various types of applications

In the present environment, an adaptive job scheduler is utilized. The corresponding scheduling framework of the job management is in a distributed fashion, as illustrated in Fig.4.

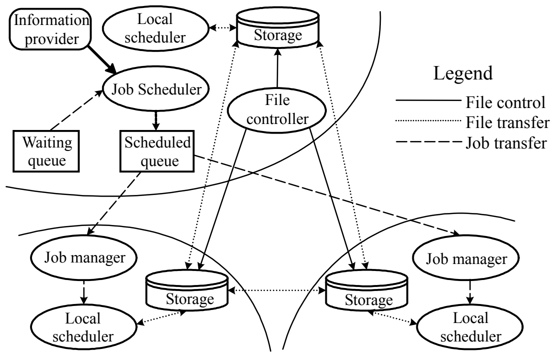


Fig.4 Framework of the job scheduling

The Job Scheduler (JS) is distributed. It schedules all the available resources in the computational grid and selects the best resources for the job by consulting the resources information. The JS chooses the proper resources by utilizing the scheduling algorithms. The Local Scheduler (LS) is a job scheduler provided by the local host system, such as OpenPBS, condor or LSF, etc. It decides how to schedule the allocated jobs according to its local resources. Once a job is submitted to a particular site, which is determined by the JS, it will be managed by the LS.

The heuristic-based greedy scheduling algorithm is one of the algorithms used to schedule the resources. Selecting the optimized resources and adapting to different applications is the goal of the HGSA. Multiple available resource metrics are considered to rank the resources, but they have different effects on different applications. The algorithm uses the weights and the impact factors of workload to reflect the special resource requirements of the applications, in terms of the metrics given. The metric weight is used to distinguish the effect of the metric on the application, while workload impact factor is used to identify the impact of the assigned workload of the job on the metrics. Their values can be customized.

The idea of the HGSA is simple, that is, by customizing the metric weights and workload impact factors to rank the filtered resources and select the resource set with the best performance, where the best performance may be the lowest resource utilization, the lowest resource cost, the fastest speed, or the best synthetical performance, etc.

Assume that *ResList* is the filtered available resources list; *PROCESSCOUNT* is the process count of the job, which is defined by the user; *SelectedList* is the selected resources list, i.e., the result. The data type of *ResList* is the list of the resource classes. There are two important properties in the resource class: *processcount* and *cpucount*. *processcount* is the allocated process count of the job to the resource, and its initial value is zero. Therefore, if the *processcount* of a resource is bigger than zero at the end of the HGSA, the resource is selected. *cpucount* is the CPU count of the resource. If the *PROCESSCOUNT* value is more than one, the job is parallel. *ResList* and *PROCESSCOUNT* are the input, while *SelectedList* is the output of the algorithm. The HGSA is described in Fig.5.

```
maxvalue=0;
SelectedList=NULL;
for (i=0; i<PROCESSCOUNT; ++i) {
  for (resit=ResList->begin(); resit!=ResList->end();
      ++resit) {
    if ((*resit).processcount<(*resit).cpucount) {
      curvalue=Rank(resit);
      if (maxvalue<curvalue) {
        maxvalue=curvalue;
        selectedit=resit;
      }
    }
  }
  if (maxvalue==0)
      return NULL;
  (*selectedit).processcount=(*selectedit).processcount+1;
  maxvalue=0;
}
for (resit=ResList->begin(); resit!=ResList->end();
      ++resit) {
  if ((*resit).processcount>0)
    SelectedList.append(*resit);
}
return SelectedList;
```

Fig.5 The heuristic-based greedy scheduling algorithm

A ranking mechanism sorts filtered resources based on the formula

$$Rank=\sum_{i=0}^{k-1}(a_i+nf_i)w_i,\quad \sum_{i=0}^{k-1}w_i=1,\quad 0\le w_i\le 1,$$

where k is the count of metrics, n is the allocated process count of the current job, a_i is a metric value, f_i is workload impact factor and w_i is metric weight. In this formula, a_i is normalized. If the effect of metric i on selecting resources is negative, a_i should be $-a_i$, such as the metric of network latency.

The MASSIVE monitoring system

In order to monitor the running jobs, the MASSIVE Monitoring System (MMS) (Luan *et al*., 2005) was developed. Its architecture is based on the Grid Monitoring Architecture (Tierney *et al*., 2002) proposed by the Global Grid Forum. Due to the large scale and distributed feature of the grid application, the MMS adopts the distributed and hierarchy structure as shown in Fig.6. The main components of the MMS are Local Monitors (LM), Site Monitors (SM), Node Monitors (NM), and Job Register Tables (JRT).

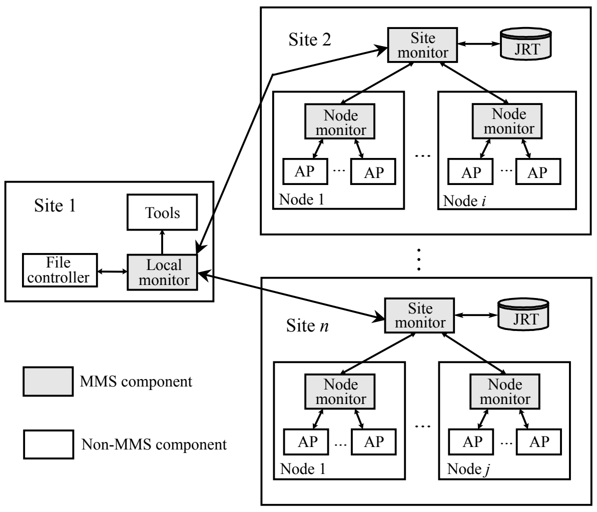


Fig.6 Architecture of the MASSIVE monitoring system

An NM runs on each node where there are application processes to be monitored. It collects information from processes running on the node and transfers them back to the SM on the same site. And it can steer the processes according to the commands obtained from the SM. In this context, the SM is the consumer and the NM is the producer.

There is an SM on each site to which the job is submitted. The SM accepts the requests from the LMs and distributes them to appropriate NMs on the same site. After receiving the information from the NMs, it forwards them to the LMs. In this context, the LM is the consumer while the SM is the producer.

The LM resides on the client side to monitor the running jobs. It can query or subscribe to the SMs for monitoring information about the jobs, and it can send the commands to the SMs to steer the jobs. In addition, it can communicate with the file controller to get information about file operations and create the performance views, according to which the user can change the mode of file transfer, such as changing the parallelism. The monitor described in Section 3 is just the LM.

The JRT stores the information of mappings between the JRC and the JVT, where JRC is the job register code provided by the client to register the jobs, and the JVT is the job vector table storing information about the jobs running on which nodes and what their processes IDs are. It is used for the consumers to find the producers. The JRC can be used to judge whether the user is the owner of the job. When the user uses the JRC, of which the format is userID@hostname:jobID, to query the monitoring information or send the control commands, the SM will check whether the connection comes from the host of the hostname and his ID is userID. Only the passed user can access the information or steer the job.

There are two types of sensors in the MMS, which are responsible for collecting the monitoring information and steering the jobs, respectively. According to the monitoring information received from the former kind of sensors, the job can be controlled by the latter.

The MMS adopted top-down start-up to start the monitoring components, which can ensure that the components are at their place whenever needed. That is, even if the SMs or NMs abort abnormally the monitoring can be carried out for the subsequent jobs.

In the MASSIVE environment, there may be many files to be moved in a typical job. In addition, there are a few applications that involve plenty of data to be transferred. Therefore, the file transfer can affect the efficiency of the job enormously. In most systems, the mode of file transfer is fixed. But because the grid is dynamic, no file transfer mode can maintain the high efficiency all the time. Therefore, it is reasonable to adjust the file transfer mode during the process. The file controller provides the function of adjusting file operations dynamically.

Job security

The MASSIVE project uses the Grid security infrastructure (GSI) (Welch *et al*., 2003) to deal with the security problems. The GSI provides the system with the advantages of single sign-on for all resources, no need for user to keep track of accounts and passwords at multiple sites, and no plaintext passwords.

The job management system is a multi-user system. To prevent malicious users from manipulating other users' jobs, and to ensure the integrity of the jobs, we adopt the Job Space (JS). Every user can have an exclusive JS, to which other users do not have access to. All the information and data related to the jobs are saved in the JS, and all the operations are carried out in the JS. On the other hand, there may be some jobs that are shared by the users. We provide the mechanism of job importing, by which the jobs outside can be imported into the JS. So that the JS can ensure the security and integrity of the jobs, at the same time the jobs can be shared among the users.

EXAMPLE AND EVALUATION

A prototype of the job management system has been implemented to validate the job management architecture and its requirements. The system has been deployed on the CESC grid and used to manage engineering jobs.

Example

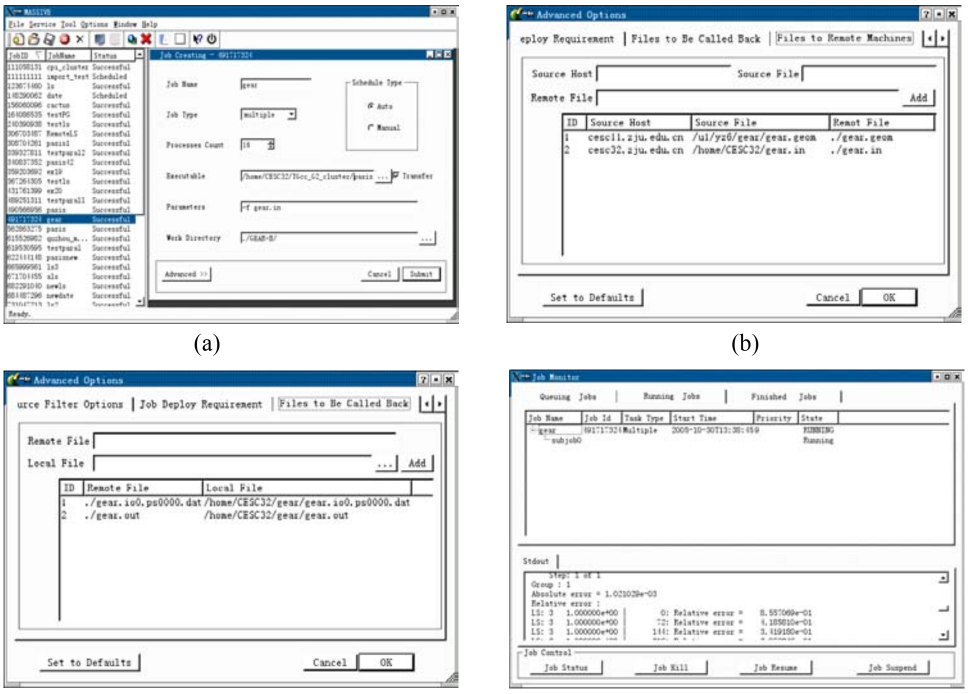
In this subsection, we use the job management system to deal with a job, which was defined before and is reused again in this example. Its description is illustrated in Fig.7.

The application is a structural analysis of a gear. As presented in Fig.7, the job name is gear, job type is multiple, schedule type is automatic, processes count is 16, executable file is "/home/CESC32/TGcc/pasis", the job parameter is "-f gear.in", the work directory is "./GEAR-B", and so on. The resource filtering conditions include: the OS type is Linux, CPU count is 16, etc. The file operations mainly relate to the file moving. Before the job execution, the parameter file "gear.in" edited at the local machine and the data file "gear.geom" generated by the supercomputer should all be transferred to the executed host. After the job execution, the resulting data file "gear.io0.ps0000.dat" and "gear.out" will be transferred from the executed host to local machine.

Fig.8 presents the job management operations. Fig.8a shows the GUI environment, in which the historical job list is presented left and the highlighted item is the job of gear. The workspace form is used to get the job description information. When the item is double clicked in the job list, the detailed job information will be present in the workspace form, by which the current job information can be edited. Figs.8b and 8c show the information on the file transferring related to the job. Because the schedule type is automatic, the job can be submitted once the corresponding information is edited. The system will use the job scheduler to allocate the resources to the job automatically by using the HGSA, and after that the job will be sent to the job executor, to which the job will be submitted to the allocated resources. The running jobs can be monitored and steered by the job monitor as shown in Fig.8d.

```
<?xml version="1.0"?>
<vjob>
 <basic>
  <id>491717324</id>
  <name>gear</name>
  <process>16</process>
  <deftime>2005-9-15T14:24:46</deftime>
  …
 </basic>
 <rsl>
  <clause>
   <resourceManagerContact qmark="1">
    cesc12.zju.edu.cn</resourceManagerContact>
   <count>16</count>
   <arguments>-f gear.in</arguments>
   <directory qmark="1">./GEAR-B/</directory>
   <directory qmark="1">
    home/CESC32/TGcc/pasis</directory>
   …
  </clause>
 </rsl>
 <filter>
  <ostype>Linux</ostype>
  <cpucount>16</cpucount>
  …
 </filter>
 <files>
  <send>
   <transfer>
    <srcRM>cesc11.zju.edu.cn</srcRM>
    <desRM>cesc12.zju.edu.cn</desRM>
    <srcpath>/u1/yz6/gear/gear.geom</srcpath>
    <despath>gear.geom</despath>
   </transfer>
   …
  </send>
  <receive>
   <transfer>
    <srcRM>cesc12.zju.edu.cn</srcRM>
    <desRM>cesc32.zju.edu.cn</desRM>
    <srcpath>gear.io0.ps0000.dat</srcpath>
    <despath>/home/CESC32/gear.io0.
      ps0000.dat</despath>
   </transfer>
   …
  </receive>
 </files>
 …
</vjob>
```

Fig.7 Job description of the 'gear'



Job Name: gear
Job Type: multiple
Processes Count: 16
Executable: /home/CESC32/TGcc_G2_cluster/pasis ✓ Transfer
Parameters: -f gear.in
Work Directory: ./GEAR-B/
Schedule Type: [✓ Auto] Manual

ID	Source Host	Source File	Remot File
1	cesc11.zju.edu.cn	/u1/yz6/gear/gear.geom	./gear.geom
2	cesc32.zju.edu.cn	/home/CESC32/gear.in	./gear.in

ID	Remote File	Local File
1	./gear.io0.ps0000.dat	/home/CESC32/gear/gear.io0.ps0000.dat
2	./gear.out	/home/CESC32/gear/gear.out

Job Name	Job Id	Task Type	Start Time	Priority	State
gear	491717324	Multiple	2005-10-30T13:38:459		RUNNING
subjob0					Running

```
    Step: 1 of 1
Group : 1
Absolute error = 1.021029e-03
Relative error :
LS: 3   1.000000e+00 |      0: Relative error =   8.557069e-01
LS: 3   1.000000e+00 |     72: Relative error =   4.185810e-01
LS: 3   1.000000e+00 |    144: Relative error =   3.419180e-01
```

(a) (b) (c) (d)

Fig.8 Job management operations. (a) Getting the basic information of a job; (b) Specifying files transferred before job execution; (c) Specifying files transferred after job execution; (d) Monitoring the jobs

With the job management system, if the computing result is unsatisfactory, the user needs only to modify the parameter file "gear.in", and then reuses the defined job gear once more.

Evaluation

The job management system used in the MASSIVE environment offers several advantages, as discussed below.

Job definition reuse. The job description can be reused easily and orderly in a visual manner, just as the above example. It is especially applicable to applications that need to be executed many times to get a reasonable result.

File operations are described in the job definition. When defining the job, the involved file operations can be described in the job file. Therefore, once a job is defined, the user can leave free, all the operations can be carried out automatically by the system. Moreover, the file controller can check the validation related to the file operations to improve the robustness of the system.

Application-Adaptive Resource Scheduling. At present, the resources used in the MASSIVE environment consist of an SGI Onyx 3900 supercomputer, a Dawning PC cluster, 6 SGI workstations and nearly 40 PCs. According to the requirements of the job, only the cluster has the waiting scheduled resources pool at the end of the filtering stage. Therefore, only the cluster can be allocated to the job in the example.

Restricted by the environment, we have crafted a simulation with the SimGrid (Casanova, 2001) simulator to evaluate the HGSA. The experimental results showed that the HGSA is efficient and adaptive to different types of application. The detailed contents on the HGSA are discussed in another paper, which is under reviewing.

Job monitor steers the file transferring before and after job execution. It is impossible to presume a stable trend in the file transferring. The job monitor in the system can monitor the file operations whose performance can be presented visually. According to the performance view, users can steer the process of the file transferring. Fig.9 presents the performance figure when changing the parallelism during the file transferring. Changing the parallelism in Fig.9 can improve the performance.

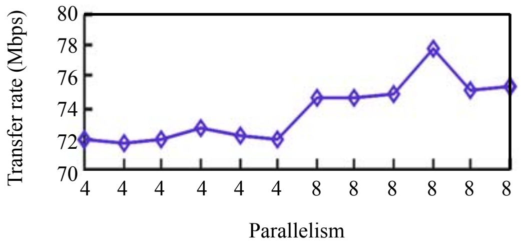


Fig.9 Changing the parallelism from 4 to 8 during the files transfer

CONCLUSION AND FUTURE WORK

In computational grid, job management is an important issue and efforts can be made to improve its efficiency, flexibility, convenience and security. In this paper, an applicable mechanism for job management is proposed, and the detailed aspects on the design and implementation are described afterwards.

The main issues solved include: the job definition reuse, which is easy-to-use and efficient; the flexible and comprehensive file/folder operations, which can meet various demands on the file operations; monitoring and steering of the job including the file operations, which provide users with the detailed job information and flexibility of controlling the jobs; the adaptive job scheduler, which can fit different types of applications; the job space, which ensures the security of the jobs, and so on. Example, evaluation revealed that this job management system is job-centric and user-friendly. It hides the details about the grid, which makes the scientists concentrate on the scientific problems.

In the future, we plan to study how to manage the data well for the jobs. In order to analyze the monitoring data effectively, we will adopt more advanced job monitoring instruments to the job management.

ACKNOWLEDGEMENTS

We appreciate the helpful discussions among the members of the Grid Computing Group at the Center for Engineering and Scientific Computation, Zhejiang University, and would like to thank Guiyi Wei, Chaoyan Zhu, Wei Wang and Xuqing Zhu for their input in the project.

References

Andreetto, P., Borgia, S.A., Dorigo, A., *et al*., 2006. CREAM: A Simple, GRID-Accessible, Job Management System for Local Computational Resources. Proceedings of Computing in High Energy and Nuclear Physics (CHEP 2006). Mumbai, India.

Baranovski, A., Garzoglio, G., Terekhov, I., Roy, A., Tannenbaum, T., 2004. Management of Grid Jobs and Data within SAMGrid. Proceedings of the 2004 IEEE International Conference on Cluster Computing. IEEE Computer Society, Washington DC, USA, p.353-359.

Casanova, H., 2001. Simgrid: A Toolkit for the Simulation of Application Scheduling. Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01). IEEE Computer Society, p.430-437.

Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S., 1998. A Resource Management Architecture for Metacomputing Systems. Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing. Springer-Verlag, p.62-82.

Foster, I., Kesselman, C., 1997. Globus: a metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, **11**(2):115-128.

Foster, I., Kesselman, C., 1998. The Globus Project: A Status Report. Proceedings of IPPS/SPDP'98 Heterogeneous Computing Workshop. IEEE Press, p.4-18.

Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, **15**(3):200-222.

Liu, C., Yang, L.Y., Foster, I., Angulo, D., 2002. Design and Evaluation of a Resource Selection Framework for Grid Applications. Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11). IEEE CS Press, p.63-72.

Luan, C.J., Song, G.H., Zheng, Y., 2005. An Infrastructure for Grid Job Monitoring. Proceedings of the International Workshop on Grid and Cooperative Computing (GCC'05). Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, **3795**:443-448.

Luan, C.J., Song, G.H., Zheng, Y., 2006. Application-adaptive resource scheduling in a computational grid. *Journal of Zhejiang University SCIENCE A*, **7**(10):1634-1641. [doi:10.1631/jzus.2006.A1634]

Tannenbaum, T., Wright, D., Miller, K., Livny, M., 2002. Condor—A Distributed Job Scheduler. Beowulf Cluster Computing with Linux. The MIT Press, Cambridge, MA, USA, p.307-350.

Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V., Wolski, R., 2002. A Grid Monitoring Architecture. Http://www.ggf.org/documents/GFD/GFD-I.7.pdf

Wang, W., Zheng, Y., Song, G.H., 2005. The Design and Implementation of Scalable Information Services in a Grid Environment. Proceedings of the 2005 IEEE International Conference on Services Computing. IEEE Computer Society, Los Alamitos, California, **2**:265-267.

Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L., Tuecke, S., 2003. Security for Grid Services. Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC-12). IEEE Press, p.48-57.

YarKhan, A., Seymour, K., Sagi, K., Shi, Z., Dongarra, J., 2006. Recent developments in gridsolve. *International Journal of High Performance Computing Applications*, **20**(1):131-141. [doi:10.1177/1094342006061893]

Zheng, Y., Song, G.H., Zhang, J.F., Chen, J.J., 2004. An Enabling Environment for Distributed Simulation and Visualization. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004). IEEE Computer Society, Los Alamitos, California, p.26-33.