JAMILA: A Usable Batch Job Management System to Coordinate Heterogeneous Clusters and Diverse Applications over Grid or Cloud Infrastructure

Juan Peng, Xiaoyi Lu, Boqun Cheng, and Li Zha

Institute of Computing Technology, Chinese Academy of Sciences,
100190 Beijing, China
{pengjuan,luxiaoyi,cbq}@software.ict.ac.cn,
char@ict.ac.cn

Abstract. Usability is an important feature of Grids or Clouds to end users, who may not be computer professionals but need to use massive machines to compute their jobs. For meeting various computing or management requirements, heterogeneous clusters with diverse Distributed Resource Management Systems (D-RMS) and applications are needed to supply computing services in Grids or Clouds. The heterogeneity of clusters and diversity of applications are easy to cause Grid or Cloud systems hard to use. This paper presents a usable batch job management system, called JAMILA, to coordinate heterogeneous clusters and diverse applications over Grid or Cloud infrastructure. JAMILA aims to implement a uniform job management mechanism to integrate different kinds of D-RMSes and it is designed to supply non-professional users from various fields with a professional but usable high performance computing environment. Some key technologies of JAMILA have been used in China National Grid and an experimental Cloud-oriented infrastructure.

Keywords: D-RMS, heterogeneous clusters, diverse applications, HCC, ASEM.

1 Introduction

As computing resources continue to move into Grids and Clouds, they let more and more users who have a variety of HPC (High Performance Computing) demand gather around Grids or Clouds. In the Grid environment, every site is managed autonomously, so different sites may be installed different D-RMSes and deployed diverse applications. According to our experience in developing Vega GOS [1, 2] for CNGrid (China National Grid), we draw the Fig. 1 to illustrate the typical scenario of heterogeneous computing resources in Grids. In the Fig. 1, Beijing site installs LSF, Shanghai site utilizes Torque, and Hong Kong site uses Condor. As different interfaces among D-RMSes, it causes the heterogeneities among clusters. If users directly use these clusters, they need spend a lot of time accepting unnecessary trainings. Meanwhile, a Grid, as a huge resource pool, should supply HPC services to users of various fields, who need to compute jobs by diverse applications. In the Fig. 1, most applications in all three sites are different, and even the same application in different sites may be installed with different versions, paths, or configurations. These cause the diversity among applications. In CNGrid, 42 kinds of applications are installed up to October in 2009, and many of them have different versions.

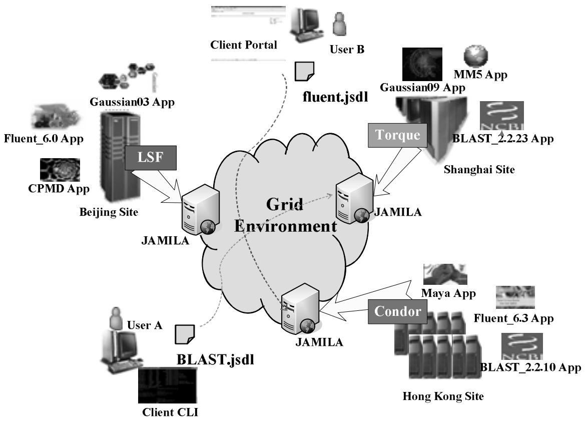


Fig. 1. Scenario of Heterogeneous Clusters and Diverse Applications in the Grid Infrastructure

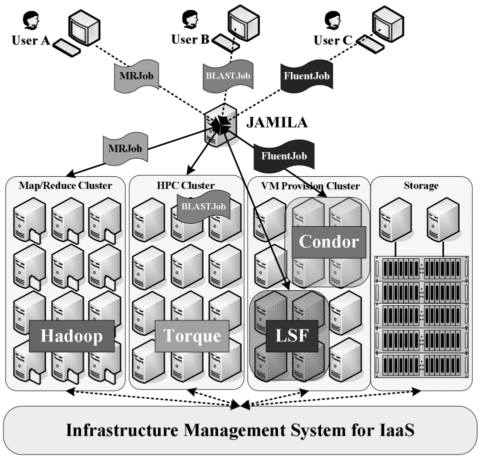


Fig. 2. Scenario of Heterogeneous Clusters and Diverse Applications in an Experimental Cloud Computing Oriented Infrastructure

Therefore, if we want to design a usable batch job management system for Grids, we should shield the heterogeneity of clusters. In addition, it also should make users focus on their job execution logic but the usages of Grid system's APIs or commands. The job execution logic is the process that users engage in computing activities: 1) **application selection**; 2) **parameter configuration of the application**; 3) **stage-in and stage-out files configurations**; 4) **resource quota configuration**. It means how many CPUs, memories and other else resources are required. This requires the batch job management system can provide a usable way for users to easily access all applications in the Grid.

For Cloud, in our opinions, we consider it has three major differences with the Grid. First and foremost, Cloud is more concerned about the management of resources within a site. There is always an infrastructure management system as the fundamental component of IaaS to support resource dynamic provisioning and partitioning. Each partition is likely to be used to support different computing modes and managed by different software. Besides, virtualization technology emerges as a new resource provisioning way. The most important feature of this way is multi-tenanted which means customers can configure their leased resources on demand within the lease rule. This easily causes the heterogeneous software deployed in virtual clusters. Last but not the least, new computing mode for large-scale data processing should be supported in a Cloud platform. Fig. 2 shows a resource view of our experimental Cloud environment. We anticipate tens of thousands of machines in a datacenter or a computing center will be divided into four big clusters: a MapReduce cluster, an HPC cluster, a VM Provisioning cluster for virtual cluster creation, and a big Storage pool. In this figure, we suppose there are two virtual clusters with Condor and LSF for users' mutative computing requirements. Similar with the Grid scenario, if we want to design a batch job management system for a Cloud, we also have to confront with the situation of heterogeneous clusters and diverse applications.

From all of above, we conclude that no matter in a Grid or a Cloud infrastructure, the heterogeneity of clusters and the diversity of applications are exist, and they are easy to cause the Grid or Cloud hard to use. This paper presents a usable batch job management system, called JAMILA, to coordinate heterogeneous clusters and diverse applications over Grid or Cloud infrastructure. JAMILA aims to implement a uniform job management mechanism to integrate different kinds of D-RMSes and it is designed to supply non-professional users a professional but usable HPC environment.

The rest of this paper is organized as follows: Section 2 presents the architecture of JAMILA. Section 3 shows some key technologies implementation of JAMILA. In the section 4, the evaluation of JAMILA will be demonstrated. Section 5 concludes the paper and gives the future work.

2 JAMILA Architecture

2.1 Design Goals and Basic Concepts

In our design, JAMILA should satisfy three major goals: 1) **Easy to use**. JAMILA should make end users just concentrate their attention on their jobs' execution logic but other things. To administrators, JAMILA should provide tools to help them integrate different clusters and manage other resources; 2) **Effectiveness**. It mainly embodies in two aspects: a) quick response to requests; b) job processing capacity: it can concurrently handle hundreds of jobs and requests well; 3) **Robustness.** The system should work well under heavy load, and the system should have recovery mechanism when system failure.

Before we introduce the architecture, four basic concepts in JAMILA are: 1) **User**. JAMILA has two kinds of users: global user and local user. A global user is managed by JAMILA while a local user is managed by local OS. User mapping exists between them; 2) **Application**. It is a software package providing some HPC functions to users; 3) **Cluster.** It is a computing resource. It is often installed D-RMS and applications to provide HPC services. We classify the clusters by the type of D-RMS installed on them; 4) **Job**. A job can be considered as a runtime instance of an application on a cluster by a global user. A job has a life cycle with different statuses.

2.2 Architecture Overview

From the scenarios in *Introduction* chapter, we find that end users mainly concern submitting jobs and quickly getting jobs' results. So, we consider designing a batch job management system should solve the following three most important problems: 1) providing a usable interface to make users easily submitting and managing jobs; 2) submitting and executing jobs on a proper cluster; 3) helping administrators integrate clusters. Based on these problems, we design JAMILA to three layers: *User Interface Layer*, *Job Execution Layer*, and *Cluster Connector Layer*, as shown in the Fig. 3.

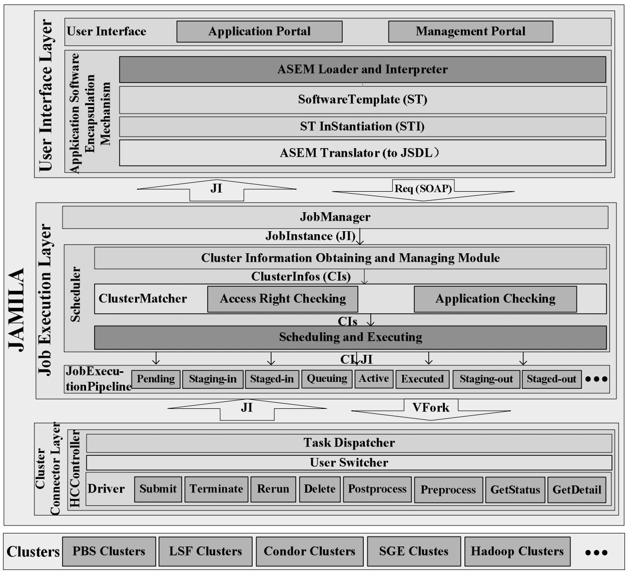
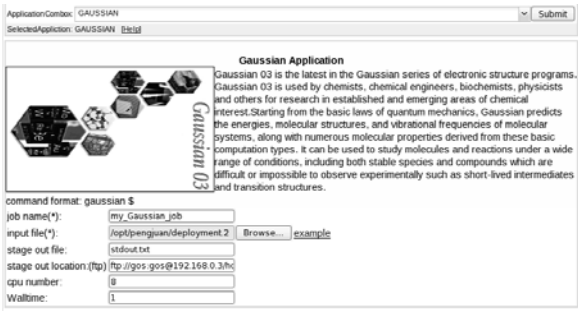


Fig. 3. Architecture of JAMILA

1) User Interface Layer: it is to make end users directly face to all kinds applications and to shield all things that are not directly related with job execution logic. In this layer, JAMILA provides two web 2.0 technology based portals: one is *Management Portal*, and the other is *Application Portal*. Web-based computing services can be provided anytime, anywhere to users from various fields. Administrators can use *Management Portal* to configure user mapping, to integrate heterogeneous clusters and to manage diverse applications. End users can use *Application Portal* to submit or to manage jobs. When doing submission, users first select an application, and then define the jobs execution logic on a web page related to the selected application. Finally, they click the submit button, and wait for jobs' results. Fig. 4 shows the submission page of Gaussian.



ApplicationCombox: GAUSSIAN Submit
SelectedAppliction: GAUSSIAN [Help]

Gaussian Application

Gaussian 03 is the latest in the Gaussian series of electronic structure programs. Gaussian 03 is used by chemists, chemical engineers, biochemists, physicists and others for research in established and emerging areas of chemical interest.Starting from the basic laws of quantum mechanics, Gaussian predicts the energies, molecular structures, and vibrational frequencies of molecular systems, along with numerous molecular properties derived from these basic computation types. It can be used to study molecules and reactions under a wide range of conditions, including both stable species and compounds which are difficult or impossible to observe experimentally such as short-lived intermediates and transition structures.

command format: gaussian $

job name(*):	my_Gaussian_job
input file(*):	/opt/pengjuan/deployment.2 Browse... example
stage out file:	stdout.txt
stage out location:(ftp)	ftp://gos:gos@192.168.0.3/hc
cpu number:	8
Walltime:	1

Fig. 4. Job Submission Page of Gaussian

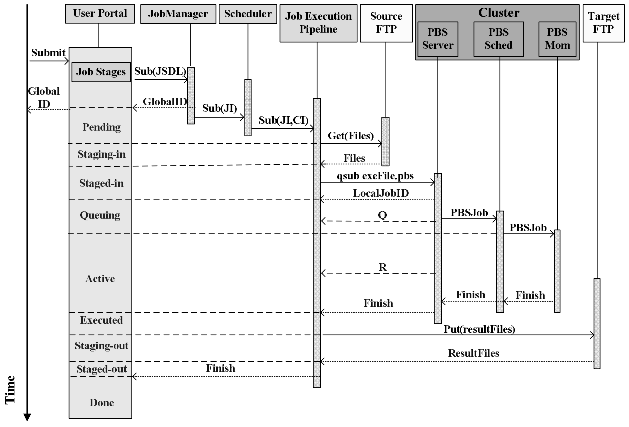


Fig. 5. The Submission Processing Sequences and Job Stages Conversion in JAMILA

2) Job Execution Layer: it addresses all requests from end users. It contains three parts: *Job Manager*, *Scheduler*, and *Job Execution Pipeline*. *Job Manager* likes a task dispatcher which pushes submission, rerunning, terminating requests to *Scheduler* and handles other simple requests by itself. *Scheduler* is to find out a target cluster and put the job to *Job Execution Pipeline* to be executed (If the request is to terminate a job, the *Scheduler* does nothing but directly pushes this request to *Job Execution Pipeline*.). *Job Execution Pipeline* is the real place to manage the execution of a job on a cluster. In this layer, JAMILA adopts GridSAM's pipeline technology [3], which utilizes SEDA (Staged Event Driven Architecture) [4] as the implementation principle, and builds on Quartz [5] framework to schedule stages and allocate threads. By pipeline technology, a job's life cycle is divided into several stages. Each stage does a specific task. Fig. 5 shows the process of submission operation and job stages. If a job gets errors at any stages, the job will exit from pipeline directly after changing its status to fail. *Job execution pipeline* technology highly improves the system's throughput.

3) Cluster Connector Layer: it aims to address how to connect heterogeneous clusters in a uniform way. The main component of this layer is *HCC* (*Heterogeneous Cluster Controller*), which contains three parts: *Task Dispatcher*, *User Switcher*, and *Driver*. *Task Dispatcher* delivers jobs to their target clusters to run. Then the *User Switcher* on the target cluster receives these jobs, maps every global user to a local user on the cluster, and redirects the task on behalf of the local user to the *Driver*. Finally, the *Driver* invokes D-RMS on the target cluster to do this task and returns D-RMS's response to *Job Execution Pipeline*.

3 Key Technologies Implementation in JAMILA

As mentioned in *Introduction*, two difficulties of implementing a batch job management system over a Grid or Cloud are how to make users easily access all kinds of applications, and how to coordinate heterogeneous clusters. JAMILA invents two technologies to solve these two problems, one is *ASEM* (*Application Software Encapsulation Mechanism*) and the other is *HCC* (*Heterogeneous Cluster Controller*). In addition, an optimization technology is also shared.

3.1 Application Software Encapsulation Mechanism (ASEM)

ASEM, which is similar with the Rappture [6, 7] technology in nanoHUB.org [7], provides a uniform solution of how to encapsulate diverse applications with good sharability and usability. In order to better understand the implementation detail of *ASEM*, we first introduce the concept of *ST*.

ST is a method for encapsulating an application software usage and it contains a vocabulary, variables, functions, and normative XML Schema that facilitate the expression of application software usage.

Definition 3.1. *ST = < JobSubmitTemplate, JobManageTemplate>*
JobSubmitTemplate = < SubAppearance, SubOperation>: defining job submission logic.
JobManageTemplate = <ManAppearance, ManOperation>: defining job management logic.

Definition 3.2. *SubAppearance|ManAppearance = < HTMLContet, {Image}>*
HTMLContet: containing an HTML string to define the application's web page.
Image = < CodingType、ImageFilePath、ImageCode>: storing the images of web pages.

Definition 3.3. *SubOperation = {Name, SoftwareName, SoftwareVersion, JobProject, Description, Executable, Argument, Output, Error, Input, {StageIn}, {StageOut}, Resource, {GenerateFile}, JobWorkDirectory}:* it is to store information of job execution.

Definition 3.4. *ManOperation = < Invoker, OperationSequence >*
Invoker: component of invoking the management operation
OperationSequence = < {GenerateFile},{UploadFile} >: defining management operation.

Definition 3.5. *StageIn | StageOut | UploadFile = < Source, Target >*
Source: file address referenced to the client host.
Target: file address referenced to the server host

Image element is to store coding data of the picture with the coding type indicated in *CodingType* element. *ASEM loader and Interpreter* has coding and decoding functions. So, when an administrator creates an *ST*, he or she configures other text parts of the *ST* and uploads pictures, then the *ASEM loader and Interpreter* codes the pictures and adds several *Image* elements to the *ST*. Next time, when he or she exports this *ST*, pictures' code is in it and it works well in other environments. This makes *STs* easy and convenient to be shared.

All *STs* in JAMILA are a kind of resource. *ASEM Loader* finds out this resource and shows them in *Application Portal* in the form of a list. When the end user chooses an application from the list, the *ASEM Interpreter* paints the web page according to the *HTMLContent* element of the selected application's *ST,* like the page shown in Fig. 4. So, by *ST* and *ASEM Loader and Interpreter*, end users can submit jobs on a web page instead of developing a complicate JSDL [8] file or using D-RMS commands directly. As users are quite clear about their job execution logic, filling such a web page form is not difficult.

Since applications installed on different clusters may have different versions, paths, and other properties which are closely dependent on specific environments. To better share the developed *STs* across Grids or Clouds, we must remove application's variability from *ST*. So we introduce *STI*, which is used to store the applications' variable information such as version, command paths, and so on. This design makes *STs* be independent with environment, so *STs* can be used to other environment besides the developed environment. And the *Management Portal* contains exporting *ST* function.

The last component is *ASEM Translator* which is to transfer end users' inputs, *ST*, and relative *STI* together to a JSDL. Then it pushes the generated JSDL to *Job Execution Layer*. For most batch job management systems supporting JSDL, so in order to be easy to interoperate with other systems, *Job Execution Layer* uses the JSDL as its input.

3.2 Heterogeneous Cluster Controllers (HCC)

HCCs are bridges between JAMILA and clusters. It contains three parts: *Task Dispatcher*, *User Switcher*, and *Driver*. All these three parts are implemented by Linux Shell Scripts. The *Driver* provides six standard interfaces for all kinds of D-RMSes as shown in the Table 1. Operations related with jobs in JAMILA are implemented by these six basic interfaces. All kinds of *Drivers* for specific D-RMSes need to be implemented these six standard interfaces. Therefore, the above layer can use a uniform way to invoke *HCCs*.

HCC separates implementation from logic of the system which makes upper layer not require caring about the underlying resources and makes JAMILA easily integrate different kinds of clusters. Furthermore, by using *HCC*, heterogeneous clusters can dynamically join to or leave from JAMILA.

Table 1. Interfaces of *Driver*

Parameter Format	Function
-b	Submit a job to a cluster's D-RMS
-s	Get a job's latest status
-d	Get a job's execution detail, including execution queue, consumed CPU time and so on
-c	Cancel a specific job
-r	Rerunning a specific job
-p	Doing pre-process or post-process of a specific job.

3.3 Job Submission Rate Controlling

In the *Job Execution Layer* of JAMILA, we design a simple but efficient mechanism to improve the system's capacity. In this layer, *job execution pipeline* uses Quartz to schedule jobs stages. When lots of jobs are submitted into the pipeline, it very easily causes congestion in Quartz, which makes many jobs hard to be scheduled. If more and more jobs continually rush into the system, it will make all threads in the system look like locked. Therefore, we design and implement a simple job submission rate controlling mechanism to make the job's arrival rate to be nearly stable. If jobs cannot be served at current, they will be saved into database. Through our experiments in the next section, we can see the processing capacity and throughput of JAMILA are highly improved by it.

4 Evaluations

4.1 Evaluations of Design Goals

Easy to use: For an administrator, he or she can integrate new clusters or old clusters with new configurations to JAMILA by *HCC*. He or she just needs to do three simple steps: copy *User Switcher* and a fit *Driver* to the cluster, change the commands path of D-RMS in the *Driver*, and configure *sudo & SSH* of the cluster to enable JAMILA server executing the *Driver* with any local users' identities without password. Now, JAMILA has *Driver*s for most popular D-RMSes including PBS, Torque, Condor, LSF, SGE and Hadoop. What is more, developing a new *Driver* is not difficult by our *Driver* Template.

JAMILA aims to supply non-professional users a professional but friendly HPC environment. In JAMILA, users do not need to learn extra computer knowledge. What they need to do when submitting a job is to define their jobs execution logic mentioned in *Introduction* chapter, and then push the jobs to JAMILA. In a word, JAMILA makes job submission and management operations like surfing internet, which can be finished just by a few mouse clicks and filling a web page form.

Effectiveness: We do some experiments to evaluate JAMILA's performance mainly on its max processing capacity and response time. All experiments are done on the server with following configuration: x86_64, two Intel(R) Xeon(R) with each four cores, 2.0GHz, 8GB memory, Gigabytes Ethernet, CentOS 5.3 (kernel 2.6.18), MySQL 5.0.45, JDK 1.6.0, Apache Tomcat 5.0.28, Torque 2.1.9.

Firstly, we evaluate the response time for job submission operation from 10 jobs/min to 400 jobs/min. As shown in the Fig. 6, the response time keeps stable nearly before 100 jobs/min, and the average response time is 316ms from 10 jobs/min to 400 jobs/min. Then, we evaluate the average turnaround time evaluated by the number of finished jobs per minute, and the result is shown in the Fig. 7. From this result, we can see that when the job submission rate reaches 300 jobs/min, if no submission rate controlling mechanism, the finished job number per minute will reduce dramatically; if we adopt job submission rate controlling, the finished job number keep stable basically.

Robustness: From the Fig. 7, we can see that Job Submission Rate Controlling makes JAMILA work well on a big submission rate. Moreover, as JAMILA having functions like job information persistence and recovery mechanism by Quartz, when the system fails, those unprocessed or unfinished jobs can continue to run.

4.2 Evaluations of Key Technologies

JAMILA's two major technologies *ASEM* and *HCC* are widely used in CNGrid for about two years. Up to October in 2009, there are 42 kinds of applications encapsulated by *ST* in CNGrid. These applications cover many fields, such as Biology, Meteorology, Chemistry, and so on. Closely analyzing these *STs*, we find that 62.07% *STs* can be developed from a simple *ST*, like Gaussian *ST* shown in Fig. 4. According to our training experience of developing *STs*, a simple *ST* like above can be developed in 10 minutes. Even the most complicated *STs* in CNGrid can be developed less than 2 hours. What's more, *ST*s can be shared by different clusters since it is independent with environments. So, developing a *ST* is not a difficult thing.

HCC technology is used to shield the heterogeneities of clusters in Grid or Cloud infrastructures. Developers of GridSAM also introduced this technology into GridSAM by our collaboration. Firstly, we give a simple statistic result of drivers' code lines in *HCC* as shown in Table 2. From this table, we can see all six *Drivers* of *HCC* are not very complex and the average code of all *Drivers* is 323 lines. Here, JAMILA can also easily integrate Hadoop cluster for large scale data processing in our experimental cloud environment through *ASEM* and *HCC* technologies.

Besides, we do experiments to show the cost of driver execution in the job submission operation and the cost of driver invocation by VFork. Both of two costs have little change as the number of continually submitted jobs increasing and the average driver invocation cost is only 6.7% to that of driver execution. We adopt JNI to invoke more light-weighted VFork mechanism to fast create a child process, instead of Fork. We find VFork can dramatically reduce the memory usage.

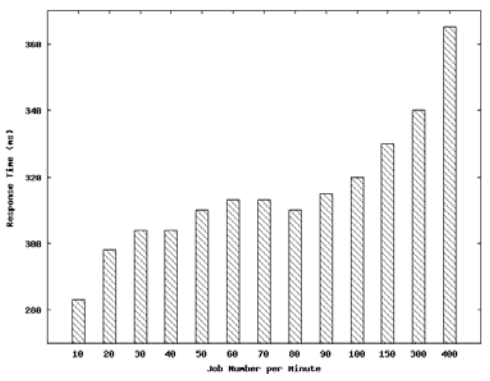


Fig. 6. Average Response Time under Different Job Submission Rate

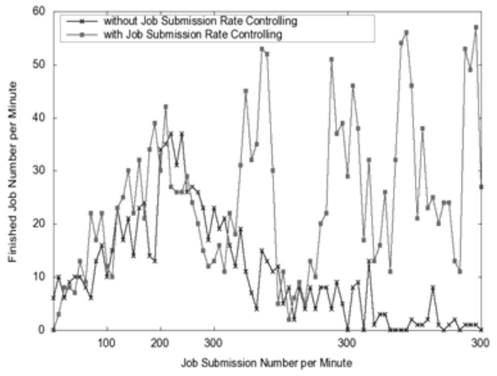


Fig. 7. Average Finished Jobs/Min under Different Job Submission Rate

Table 2. Statistic Results of Drivers' Code Lines in HCC

Driver Type	Code Lines of Driver
PBS	405
LSF	323
Condor	378
Fork	138
SGE	412
Hadoop	341

5 Conclusion and Future Work

This paper presents a batch job management system JAMILA, to coordinate diverse applications and heterogeneous clusters over Grids or Clouds. In JAMILA, *ASEM* is to encapsulate applications with good sharability and usability. With it, users just need to concern jobs execution logic. ASEM has been used in CNGrid practically for two years over 12 sites across China and up to October in 2009, 42 kinds of applications are encapsulated by *ST*. *HCC* is used to integrate heterogeneous clusters in a uniform way. In order to integrate a new kind of cluster, administrators just need to develop a fit *Driver* for the new cluster. Now, JAMILA has *Driver*s for Fork, PBS, LSF, Condor, SGE, and Hadoop.

Besides these two important technologies, we do some work on improving JAMI-LA's effectiveness and robustness, such as job submission rate controlling, memory leak checking and so on. These efforts largely improve the system's processing capacity and reliability. JAMILA with 1GB memory usage configuration can handle about 230 million jobs' whole life cycle running at the same time.

There is a lot of future work to do. The most important is JAMILA needs a good scheduling algorithm to balance loads among clusters. Currently, scheduler component in JAMILA uses random scheduling algorithm to get a target cluster.

Acknowledgment

This research is supported in part by the National Basic Research (973) Program of China (Grant No. 2005CB321807) and the Hi-Tech Research and Development (863) Program of China (Grant No. 2006AA01A106, 2006AA01Z121, and 2009 AA01A130).

References

1. Xu, Z., Li, W., Zha, L., Yu, H., Liu, D.: Vega: A Computer Systems Approach to Grid Computing. Journal of Grid Computing 2(2), 109–120 (2004)
2. Zha, L., Li, W., Yu, H., Xie, X., Xiao, N., Xu, Z.: System Software for China National Grid. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 14–21. Springer, Heidelberg (2005)
3. Lee, W., McGough, A.S., Darlington, J.: Performance Evaluation of the GridSAM Job Submission and Monitoring System. UK eScience Program All Hands Meeting (2005)
4. Welsh, M., Culler, D., Brewer, E.: SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In: Proceedings of the 18th ACM Symposium on Operating System Principles, pp. 230–243. ACM Press, Banff (2001)
5. Cavaness, C.: Quartz Job Scheduling Framework: Building Open Source Enterprise Applications. Prentice Hall PTR, Upper Saddle River (2006)
6. Rappture, `https://nanohub.org/infrastructure/rappture/`
7. Klimeck, G., McLennan, M., Brophy, S.P., Adams III, G.B., Lundstrom, M.S.: NanoHUB.org: Advancing Education and Research in Nanotechnology. Computing in Science and Engineering 10, 17–23 (2008)
8. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) Specification V1.0. Open Grid Forum, GFD.136 (2008)