A Load Management System for Running HLA-based Distributed Simulations over the Grid

Wentong CAI   Stephen J. TURNER   Hanfeng ZHAO
Parallel and Distributed Processing Laboratory
School of Computer Engineering
Nanyang Technological University
Singapore 639798

Abstract

Running a large-scale distributed simulation may need a large amount of computing resources at geographically different locations. These resources may be from different organizations. The simulation may run for a long period of time and the availability and the amount of computing resources available may change during the course of the simulation execution. Therefore, coordinating and managing the resources for distributed simulation to complete the simulation efficiently and effectively is a critical issue. This paper describes a load management system for HLA-based distributed simulation. The system is constructed on top of a Grid Computing environment supported by Globus. The overall structure of the system is presented in the paper and how the system saves and restores a federate is also discussed in detail.

Keywords:*Distributed Simulation, HLA, Load Management System, Grid Computing, and Globus*

1 Introduction

The advances of the Internet and parallel commodity computers have made parallel and distributed simulation (PADS) a viable technology [4]. PADS involves the execution of a single simulation program on a collection of either tightly coupled processors (e.g., shared memory multiprocessors) or loosely coupled computers (e.g., PCs interconnected by a LAN or WAN). Applications of PADS include the analysis of complex systems such as the next generation Internet as well as computer-generated virtual worlds for training and entertainment.

The High Level Architecture (HLA) for Modeling and Simulation [6, 7] was developed as an IEEE standard to facilitate interoperability among simulations and promote reuse of simulation components. Using HLA, a large-scale distributed simulation can be constructed using a huge number of geographically distributed computing nodes.

In a large-scale simulation running over a distributed environment, there are a lot of simulation components running concurrently. The simulation may need computing resources from different organizations and the availability of these computing resources may change during the execution of the simulation. To meet the real-time requirement demanded by interactive simulations and the performance requirement demanded by analytical simulations, resource management mechanisms that balance the load in the simulation execution and provide fault-tolerance capability are desired [12]. In addition, to conduct simulation experiments easily over geographically distributed resources from different organizations, mechanisms that can deploy coordinated, secured simulation executions are required.

HLA is a software architecture for building a large-scale distributed simulation over the Internet. The benefits of the HLA are interoperability and reusability. However, the HLA does not provide any mechanism for managing the resources where the simulation is being executed. The objective of our research is to develop a Load Management System (LMS) to support the execution of HLA-based large-scale distributed simulations over geographically distributed computing resources efficiently and effectively. The LMS should provide mechanisms for load-balancing, fault-tolerance, coordination of simulation execution and security. The focus of this paper is on the overall infrastructure of the LMS and its support for load-balancing.

To take advantage of current developments in parallel and distributed computing and in PADS, our strategy is to combine an existing resource sharing system with the Run Time Infrastructure (RTI) of the HLA. The existing resource sharing system is employed to perform the task of resource monitoring, coordination of simulation execution and security. The RTI is used to perform simulation related tasks (e.g., synchronization and time management). Our LMS will match-make the resource requirements of an HLA-based simulation and the resources managed by the resource sharing system, schedule the simulation and provide mechanisms for load-balancing and fault-tolerance during the simulation. Our approach is significantly different from the system reported in [9] which was not built using any existing resource sharing system. That system requires more effort for development and may not be able to take full advatage of the Grid.

There are several software packages that can be used to manage resources for clusters, for example, Condor [1] and PBS [10]. In our case, we are more interested in resource management for running HLA-based large-scale distributed simulations over the Grid. The Grid is an emerging hardware and software infrastructure that provides dependable, consistent, pervasive, inexpensive and secure access to high-end computational capabilities [3]. It pools large-scale, heterogeneous resources from geographically diverse locations into an ensemble and presents them as an integrated, unified and single resource. Currently available toolkits that enable Grid Computing include Globus [2], Legion [5], MultiCluster [13] and Sun Grid Engine [11]. Globus is chosen in our project for its availability, completeness and popularity – it has become the de-facto standard for Grid Computing.

This paper is organized as follows: Section 2 will briefly introduce HLA/RTI and Globus. Section 3 will describe the architecture of our load management system and the communication and interaction between components in our system. Section 4 will describe the interface for suspending and resuming federates and highlight the features in our prototype implementation. Finally, Section 5 will conclude the paper with the directions of future work.

2 Background

The HLA baseline definition comprises three main components: *Rules*, *Interface Specification* and *Object Model Template*. The HLA Rules are a set of ten basic rules that define the responsibilities and relationships among the HLA components of an HLA federation. The HLA Interface Specification provides a specification of the functional interfaces between HLA federates and the HLA Run-Time Infrastructure (RTI). The HLA Object Model Template provides a common presentation format for HLA object models.

In HLA, a federate can be viewed as a component simulation model that is taking part in a large-scale simulation. A federation (that is, a large-scale simulation) is made up of a set of federates. The RTI is the software needed to execute a federation and it is basically an implementation of the HLA Interface Specification. Figure 1 shows a typical

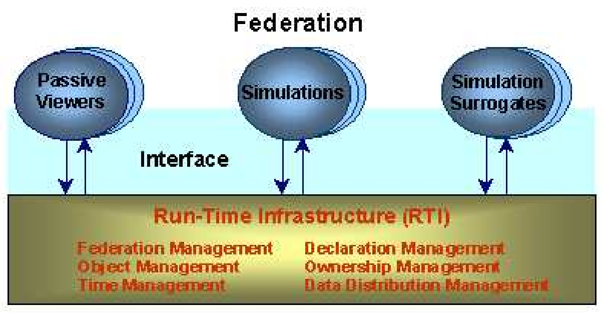


Figure 1. An HLA Simulation Federation

configuration of an HLA federation. A federate can be a passive viewer, a surrogate for human participants in a simulation, or a simulation of some number of simulation entities. A federate is defined as having a single contact point with the RTI. There are many federates in a federation, but only one RTI. In a federation execution, federates communicate indirectly through the RTI which provides services, e.g., object management and time management, for coordinating the execution of federates at geographically different locations to form a large-scale simulation.

Globus is a Grid toolkit that originated from Argonne National Laboratory and University of Southern California, USA. It takes a "library of libraries" approach and is structured as a set of largely orthogonal components. Globus provides libraries, written in C, for addressing individual concerns in Grid Computing like security, resource management and resource discovery. In the Globus framework, applications can take advantage of only the libraries they need without adopting the entire architecture to create a distributed application. The following are the major modules in Globus that are used in our LMS:

- *Grid Security Infrastructure* (GSI) provides sevices for authentication, communication protection and authorization.

- *Grid Resource Allocation Manager* (GRAM) provides services for resource allocation, monitoring and control.

- *Grid Information Service* (GIS) provides access to static and dynamic information regarding heterogeneous resources.

Figure 2 shows our Grid testbed where clusters and individual machines are "glued" together using Globus. In our testbed, different scheduling packages are employed to manage resources in different clusters.

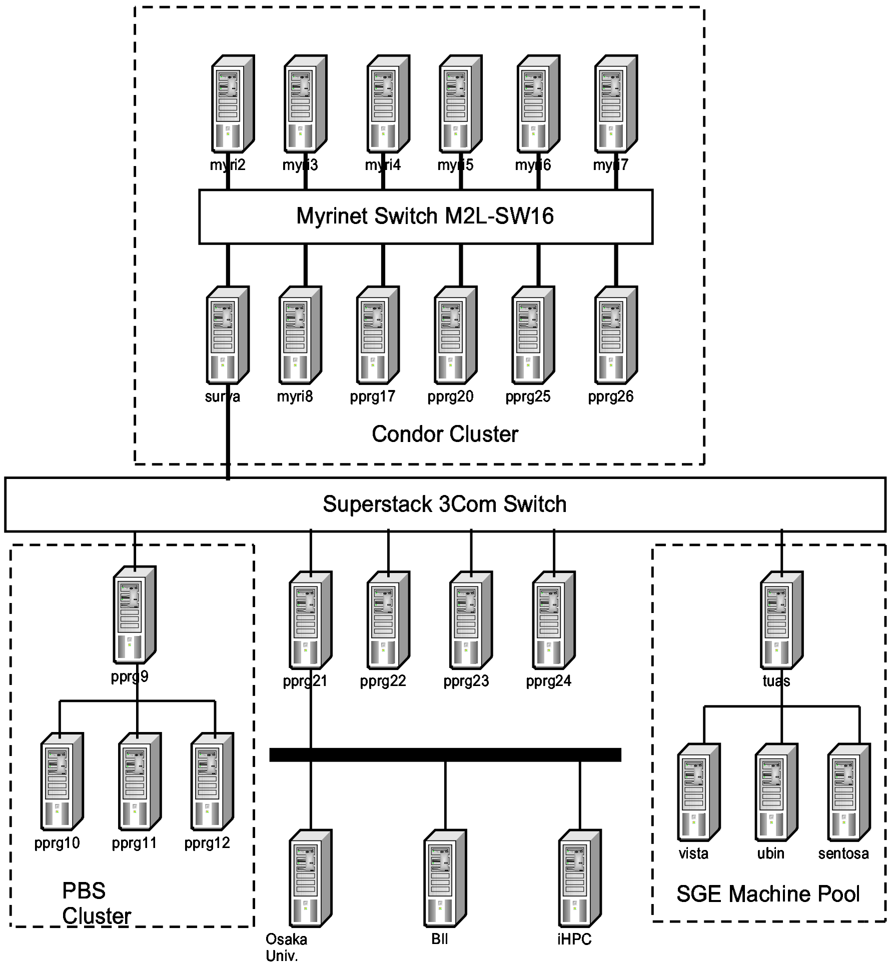


Figure 2. Grid Testbed

3 Design of Load Management System

Our objective is to develop a Load Management System (LMS) to manage resources over the Grid for distributed simulations. With the introduction of the LMS, conceptually, Figure 1 should be modified as shown in Figure 3. RTI is the infrastructure for supporting the execution of the simulation federation and LMS is the system for managing the resources where the simulation federation is executing. Federates communicate with the RTI for coordinating simulation execution and they communicate with the LMS for performing the tasks of invocation of federate execution, federate state saving and federate migration. Note that the communication between the LMS and federates does not violate HLA rules, since there is no simulation specific information exchanged between the federate and the LMS.

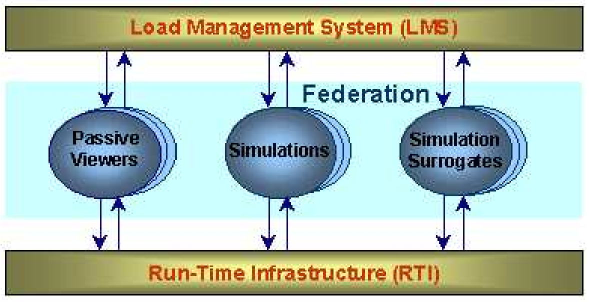


Figure 3. Load Management System for HLA-based Simulation

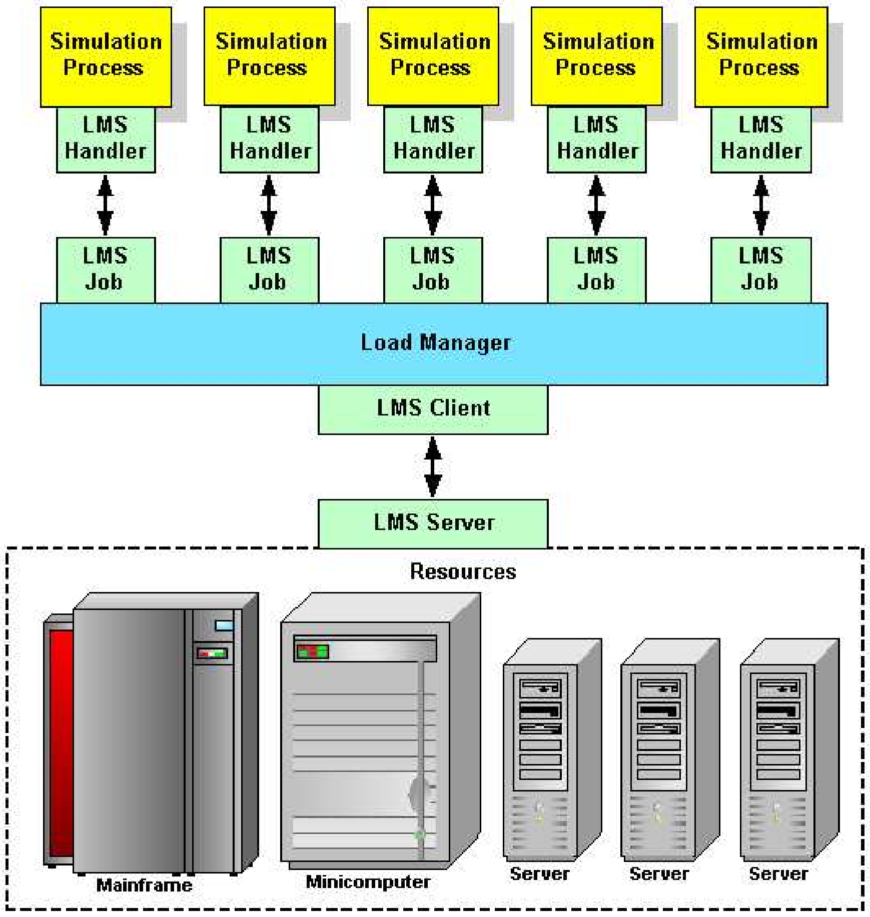


Figure 4. Overall Structure of Load Management System

3.1 Overall Structure

Figure 4 shows the overall structure of the LMS. Basically, it consists of two sub-systems: *job management* sub-system (LMSJobs, LMSHandlers and the load manager) and *resource management* sub-system (LMSClient and LMSServer). The job management sub-system is responsible for monitoring the execution of federates and performing load-balancing activities. The resource sub-system is realized using the services provided by the Globus Toolkit and is in charge of resource discovery and monitoring.

For each simulation federation, LMS creates a *Load Manager* (LM) which contains several instances of LMSJobs and a LMSClient. The LMSJobs correspond to the different simulation federates. The LMSClient gets a request from LM and passes it to the LMSServer. The requests include:

- request for resources (the LMSClient will connect to an LMSServer to discover available resources, there is one LMSServer for each group of Globus enabled resources.).
- start or restart a job at a resource (a job is a simulation federate with a corresponding LMSHandler).
- get information about a resource (resource information is used by LM for making load-balancing decisions).

The LMSServer uses Globus GSI services to authenticate the client connection, Globus GIS services to obtain information about available resources and Globus GRAM services to start execution of jobs at allocated resources.

HLA federates are not independent. Since they communicate with each other through the RTI, the RTI has to know the information on the locations where the federates are being executed. Simply stopping the execution of a federate without informing the RTI will cause an error to the entire simulation. Therefore, a LMSHandler has to be integrated with each federate, which communicates with a LMSJob in the LM and carries out the task of stopping a federate execution properly.

To increase concurrency, a multithreading architecture is adopted in the development of the LM. A LMSJob is created for each federate in the federation. It communicates with the corresponding LMSHandler and represents a job in the Load Management System. LMSJob stores the following information about a job: job status, state file information, host, start time and end time.

Section 4.2 describes the load-balancing algorithm used by LMS. The mechanism that support load-balancing is briefly described as follows: When the system needs to migrate the job to another host, the LMSJob will send a message to the corresponding LMSHandler to suspend the job. Using the interface provided by the LMSHandler, a federate can detect the condition and save the states of simulation objects into a file. The LMSHandler then sends the state information (e.g., state file name) to the corresponding LMSJob. How a federate is suspended and resumed will be discussed in detail in the following sections.

The LM uses some load-balancing strategies to decide the host where the job should be executed. It sends a *start job* request to the LMSServer via LMSClient and the LMSServer will start the job at the selected host using Globus GRAM services.

3.2 Interaction among Components

As illustrated in the following diagrams, there are communications between LMSHandler and LMSJob, LMSJob and the load manager, the load manager and LMSServer. These communications are currently implemented using sockets.

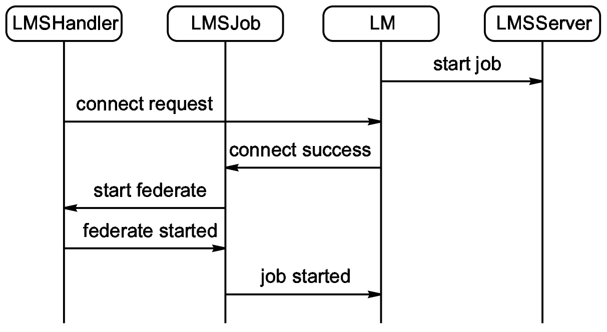


Figure 5. Start a Job

Figure 5 shows how a job is started. First, the LM will start a job (federate) by asking LMSServer to allocate a resource. Once the federate is executed, a LMSHandler will be created. It makes connection to the LM by sending a *connect request* message. On receiving the message, the LM will create a corresponding LMSJob and pass it the connection via *connect success* message. The created LMSJob will then inform the LMSHandler to start the execution of the federate.

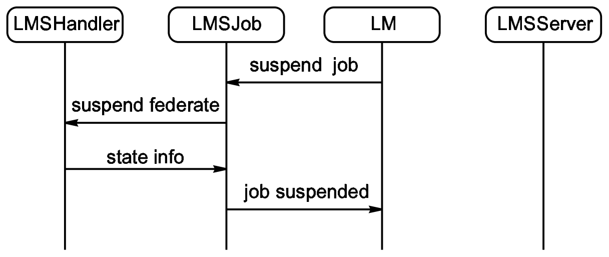


Figure 6. Suspend a Job

Figure 6 shows how a job is suspended. To suspend a job, the LM sends a *suspend job* message to the LMSJob which in turn sends a *suspend federate* message to the corresponding LMSHandler. On receiving the message, the LMSHandler will save the federate state in a file, resign from the federation execution and send a *state info* message to the LMSJob.

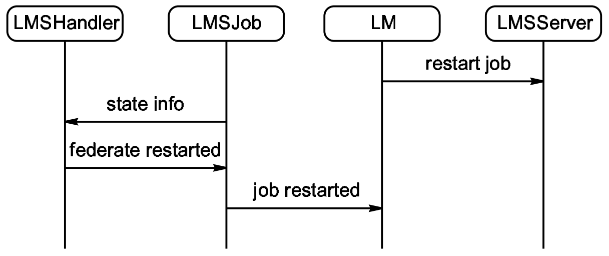


Figure 7. Restart a Job

Figure 7 shows how a suspended job is restarted. To restart a job, the LM sends a *restart job* request to the LMSServer (via LMSClient). The LMSServer will restart the job execution at a selected host. But, in this case, the federate execution will not be started until the LMSHandler receives a *state info* message from the corresponding LMSJob. Note that there is no need for the LMSHandler to make a connection to the LM and for the LM to create the LMSJob. Since the job is a suspended job, the LMSJob

Table 1. LMSHandler **Interface**

```
// constructor
public LMSHandler(String host, int port, String jobName)
// methods
public boolean isJobStopped( )
public String getAttribute(String attr)
public void setAttribute(String attr, String value)
```

must have already been created. The connection information is passed to the LMSHandler when the LMSServer restarts the job.

4 Prototype Implementation

Commodity Grid Toolkit (CoG Kit) [8] is used in the implementation of the prototype. It provides structured Java packages for accessing Globus services.

4.1 LMSHandler **Interface**

LMSHandler is defined by extending the Java Thread class. So, it runs as a separate thread with the user federate. Table 1 shows the interface of the LMSHander.

Table 2 shows how the methods defined in LMSHander are used in an example user federation – HelloWorld. First, the federate needs to create an instance of LMSHandler and to start the thread execution. In the statement, LMHost is the host where the load manager is running. LMPort is the port on which the load manager listens for connections, and FederateName is the federate's name. These are the arguments passed to the federate when it is started. Then, in the main simulation loop, the federate needs to check whether or not the execution is suspended by the load manager by executing isJobStopped( ). Recall that the load manager suspends a federate by asking LMSJob to send a message to the corresponding LMSHandler. Once receiving this message, LMSHandler will set a boolean variable so that isJobStopped( ) will return true.

If the federate execution is suspended, the federate will have to perform state saving by using the methods provided by the LMSHandler. In the case of HelloWorld, the value of ticknum and the value of current population need to be saved. LMSHandler saves these in a file and passes the location of the file to the corresponding LMSJob. The load manager will then use the information and GSI FTP services provided by Globus to transfer the file to a pre-specified ftp server.

To resume the federate execution, the load manager will stage the state file to the assigned execution node from the ftp server (again by using GSI FTP services). The federate then can use the methods provided by the LMSHandler to retrieve the values and initialize simulation state before starting the main simulation loop.

Table 2. Execution Loop in a Federate

```
import lms.LMSHandler;
...
// create an LMSHandler and join the federation
LMSHandler lmsHandler =
  new LMSHandler(LMHost, LMPort, FederateName);
lmsHandler.start();
...
// main simulation loop
for (int ticknum = init_tick; ticknum <= total_ticks; ticknum++) {
 ...
 if (lmsHandler.isJobStopped()) {
  System.out.println("Stopping job...");
  // save states into a file
  lmsHandler.setAttribute("TickNum",
    String.valueOf(ticknum));
  lmsHandler.setAttribute("Population",
    String.valueOf(myCountry.getPopulation()));
  // break from the main simulation loop
  break;
 }
 ...
}
// resign for the federation
...
```

4.2 Load Balancing Algorithm

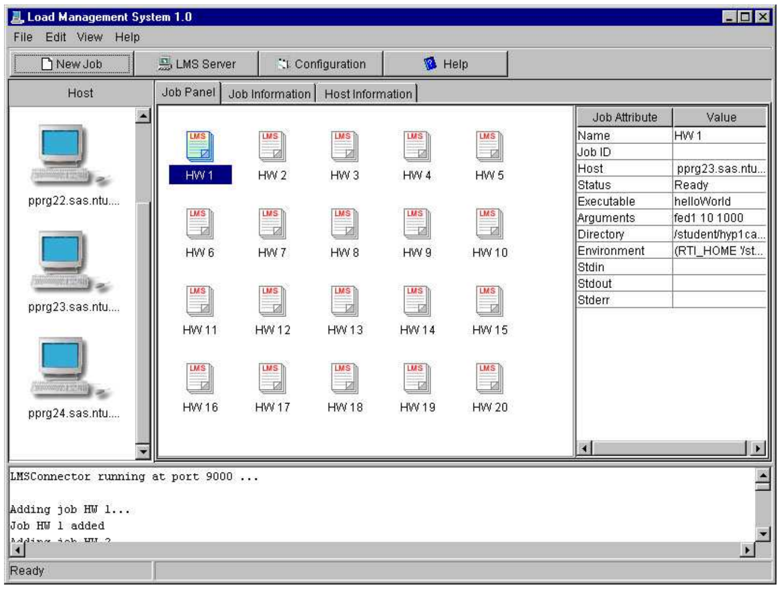
GIS services are used to collect resource information which includes *CPU speed*, *physical memory size* and *CPU load* etc. The loads are classified into levels. The objective of the algorithm is to keep the load of all the nodes at the same level. The load manager will always schedule a federate at the node with the lowest load level.

The load manager checks the resource information every five minutes. If a node is overloaded, the load manager will suspend some federates and migrate them to other lightly loaded nodes.

An HLA federation often suffers from high communication overhead. As future work, the load management system should also consider the optimization of communication while balancing the load of resources.

4.3 Graphical User Interface

The Load Management System is written entirely in Java. Figure 8(a) shows the interface of our LMS.

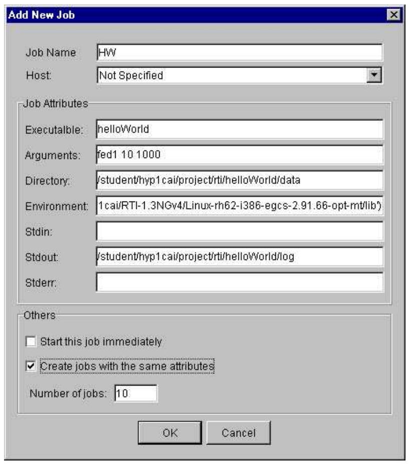


Job Attribute	Value
Name	HW 1
Job ID	
Host	pprg23.sas.ntu...
Status	Ready
Executable	helloWorld
Arguments	fed1 10 1000
Directory	/student/hyp1ca...
Environment	(RTI_HOME '/st...
Stdin	
Stdout	
Stderr	

```
LMSConnector running at port 9000 ...

Adding job HW 1...
Job HW 1 added
```

(a)



Job Name: HW
Host: Not Specified
Executalble: helloWorld
Arguments: fed1 10 1000
Directory: /student/hyp1cai/project/rti/helloWorld/data
Environment: 1cai/RTI-1.3NGv4/Linux-rh62-i386-egcs-2.91.66-opt-mt/lib')
Stdin:
Stdout: /student/hyp1cai/project/rti/helloWorld/log
Stderr:

☐ Start this job immediately
☑ Create jobs with the same attributes
Number of jobs: 10

(b)

Figure 8. Job Panel & Job Creation

The *LMS Server* button is for the LM making connection with a LMSServer. Once the connection is established, the hosts managed by the LMSServer will be added to the *Host* panel. The *New Job* button is for users to create new jobs. On clicking the button, a *New Job Dialog* window will be shown (see Figure 8(b)). After a job is created, it will be added to the *Job Panel*. The *Configuration* button is for setting system configuration (e.g., remove the LMSServer or set the ftp server for file transfer).

The user can choose to control the loads manually (drag the jobs and drop to the hosts) or let the system control the loads automatically. After jobs are started, dynamic job information (e.g., job status and host) can be obtained by clicking on a job icon or *Job Information* panel (see Figure 9).

5 Conclusions and Future Work

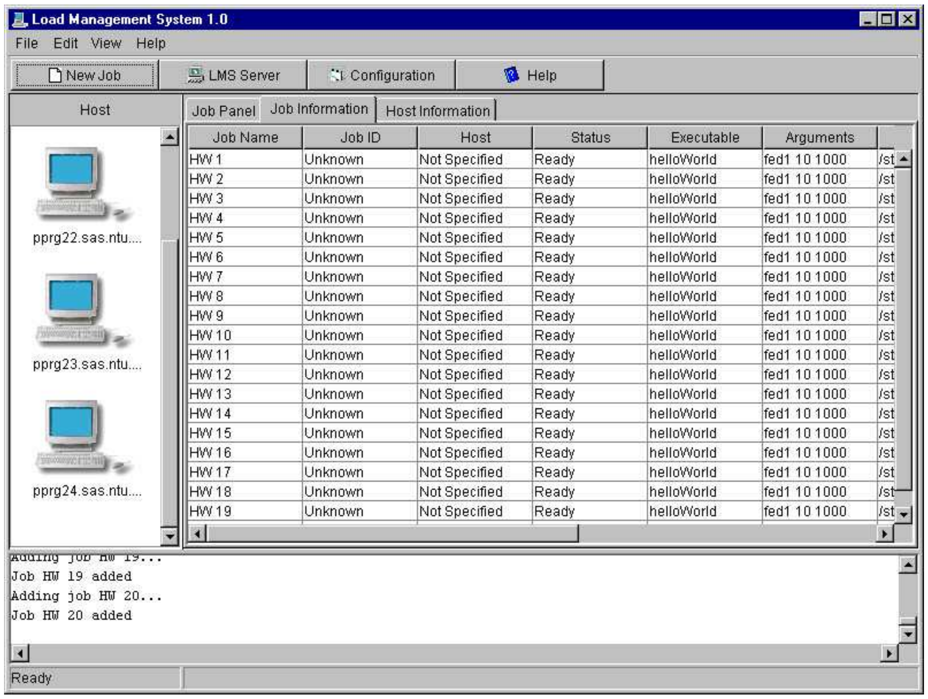
This paper describes the architecture of our Load Management System (LMS) for HLA-based large-scale distributed simulations over the Grid and the mechanisms that support load-balancing. To take advantage of current developments in parallel and distributed computing and in PADS, our strategy is to combine an existing resource sharing system with the Run Time Infrastructure (RTI) of the HLA. Globus has been chosen for performing the tasks of authentication of connection, discovery of resources and invocation of jobs. However, it is not a straightforward application of Globus since HLA federates are not independent processes. The communication between federates must be through the RTI and thus it must be aware of the migration of federates.

How our LMS utilizes the Globus services and performs federate migration is discussed in the paper. By utilizing services of Globus, users of our LMS will be able to use vast resources with a "single log-in" and the system will be able to obtain the resource information and schedule the jobs in a straightforward manner.

As for future work, dynamic load-balancing strategies suitable for running HLA-based large-scale distributed simulations over the Grid will be further investigated. Mechanisms that support fault-tolerance will also be developed. In addition, the current federate migration mechanism may also need to be enhanced by considering simulation object ownershipment management in order to handle more complicated scenarios. For more complicated applications, there are time management issues that also need to be considered.

References

[1] Condor. `http://www.cs.wisc.edu/condor`, Computer Science Dept., Univ. of Wisconsin-Madison.



Job Name	Job ID	Host	Status	Executable	Arguments	
HW 1	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 2	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 3	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 4	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 5	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 6	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 7	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 8	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 9	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 10	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 11	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 12	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 13	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 14	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 15	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 16	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 17	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 18	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st
HW 19	Unknown	Not Specified	Ready	helloWorld	fed1 10 1000	/st

```
Adding job HW 19...
Job HW 19 added
Adding job HW 20...
Job HW 20 added
```

Ready

Figure 9. Job Information Panel

[2] Ian Foster and C. Kesselman. “Globus Project: A Status Report”. in Procs of Heterogeneous Computing Workshop, pp.4-18, 1998 (also see `http://www.globus.org`).

[3] Ian Foster, C. Kesselman and Steve Tuecke. “The Anatomy of the Grid”. *International Journal of High Performance Computing Applications*, 15(3):200-222, 2001.

[4] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*, Wiley-Interscience, 2000.

[5] Andrew S. Grimshaw, W. A. Wulf. “The Legion Vision of a Worldwide Virtual Computer”. *Communications of the ACM*, 40(1):39-45, January 1997.

[6] IEEE Standard 1516 (HLA Rules), 1516.1 (Interface Specification) and 1516.2 (OMT), Sept. 2000. (also see `http://www.dmso.mil/public/transition/hla/`)

[7] Frederick Kuhl, Judith Dahmann and Richard Weatherly. *Creating Computer Simulation Systems: An Introducion to the High Level Architecture*, PTR, Sept. 1999.

[8] Gregor von Laszewski, Ian Foster, Jarek Gawor and Peter Lane. “A Java Commodity Grid Kit”. *Concurrency and Computation: Practice and Experience*, Volume 13, Issue 8-9, pp.643-662, 2001.

[9] Johannes Lüthi and Steffen Großmann. “The Resource Sharing System: Dynamic Federate Mapping for HLA-based Distributed Simulation”. in Procs. of 15th Workshop on Parallel and Distributed Simulation, pp.91-98, Lake Arrowhead, California, May 2001.

[10] Portable Batch System. Veridian Systems. `http://www.openpbs.org`

[11] Sun Grid Engine. Sun Microsystems. `http://www.sun.com/software/gridware`

[12] Eric White and Michael Myjak. “A Conceptual Model for Simulation Load Balancing”, 98S-SIW-010, 1998 Spring Simulation Interoperability Workshop, Orland, Florida, USA, March 1998.

[13] Ming Q. Xu. “Effective Metacomputing Using LSF MultiCluster”, in Procs. of 2001 International Symposium on Cluster Computing and the Grid, Brisbane Australia, May 2001.