MIS Review Vol. 20, No. 2, March (2015), pp. 77-104
DOI: 10.6131/MISR.2015.2002.04

Development of an In-House Grid Testbed Supporting Scheduling and Execution of Scientific Workflows

Harshadkumar B. Prajapati[1], Vipul A. Shah[2]
[1]*Department of Information Technology, Dharmsinh Desai University, India*
[2]*Department of Instrumentation and Control Engineering, Dharmsinh Desai University, India*

ABSTRACT: *Researchers working in Grid workflow scheduling need a real Grid environment to produce the results of experiments. However, many interested researchers of academic institutes may not be able to produce experimental results due to unavailability of a required testbed at their institutes. This article addresses an important challenge of developing an in-house Grid testbed that supports workflow scheduling and execution. This article proposes the architectural design of the in-house testbed and then concisely presents chosen software tools, their understanding, installation, configuration, and the testing related to the implementation of the testbed. Furthermore, the article also presents the methodology of performing experiments on the testbed. The in-house Grid testbed is implemented using open-source, freely available, and widely used software components. In addition, the testbed allows to produce a real Grid scenario of varying bandwidth values by emulating the network characteristics among the Grid-sites of the testbed. This article addresses testing of all the internal components of the testbed and their integrations for their proper working. This article also provides testing and demonstration of workflow scheduling and execution. We believe that this article can educate novice users about developing a Grid testbed. The presented Grid testbed can easily be replicated or adapted; furthermore, the presented deployment of the Grid testbed can guide to researchers for carrying out real experimentation for their research purposes.*

KEYWORDS: *Grid Testbed, Grid Deployment, Grid Software Integration, Workflow Scheduling, Workflow Execution.*

1. Introduction

Grid computing (Foster & Kesselman, 2003; Foster et al., 2001) enables to execute performance demanding scientific applications efficiently by exploiting distributed resources in a collaborative manner. Many research projects, a few examples include (Blaha et al., 2014; Exon, n.d.; LIGO, n.d.; Montage, n.d.), try to solve their computing problems by making computation demanding applications composed of reusable batch-executables. Various systems (Altintas et al., 2004; Deelman et al., 2005; Fahringer et al., 2005) have been used by such projects to execute computation demanding applications

efficiently. Moreover, the systems that are open-source, freely available, well documented, and actively updated attract attention of many researchers and users.

A Grid application having data dependencies among its jobs is called workflow application (Taylor et al., 2007), which can be represented as a Directed Acyclic Graph (DAG). A workflow scheduler respects the dependencies among the tasks of a workflow in the prepared output schedule and a workflow executor executes these tasks as per the arranged order on the chosen resources. The scheduling (Pinedo, 2008) aspect in Grid computing (Dong & Akl, 2006; Prajapati & Shah, 2014c) is involved in two diffierent entities: a local resource scheduler and an application scheduler. Workflow scheduling (Yu et al., 2008), an application scheduling, in Grid is complex and challenging (Deelman et al., 2005; Wieczorek et al., 2005), as the workflow scheduler has to decide about which resources will execute which tasks, what will be the order of the tasks, how to respect the data dependencies, and how to minimize the makespan of the workflow application.

A Grid computing architecture is complex to build and configure, as the Grid environment is generally implemented by using more than one software and with more than one physical computing resources. Readers may find how-to guide for installation and configuration of an individual software from software vendor. However, integrating various Grid related software is a big challenge for novice users. As a solution to this problem, in big organizations specialized system administrators are responsible for building the Grid testbed; however, in small organizations, it may not be the case. Many researchers working in academic institutes may not have budget to acquire specialized system administrators. Consequently, such researchers have no option other than deploying needed environment themselves. Because a Grid computing architecture involves use of various software in order to develop an environment with desired functionalities, the beginners need to spend a lot of time in understanding various topics related to Grid computing. Furthermore, many tools and software are available to solve the same purpose, which increases the complexity of building a Grid computing architecture.

Three major categories of Grid environment related software are (1) Local Resource Manager, (2) Grid middleware, and (3) higher level software and services. However, to perform workflow scheduling of scientific workflows requires integrations of appropriate software. Building and configuring of a required Grid environment involves the understanding of various software and their proper configuration. Therefore, the new researchers, specifically academic researchers, happen to stay away from Grid computing topic, or instead rely on simulation based approach. Moreover, if researchers want to embed new functionalities or new algorithms, then they need to take right decision about various software. Therefore, this article addresses the problem of design and implementation of the required Grid testbed supporting workflow scheduling and

execution with the minimal physical resources available. The article discusses the development in a logical way rather than showing the commands that are found in how-to guides.

Pegasus WMS is a widely used workflow management system for scheduling of scientific workflows; it is open-source and freely available. Therefore, we choose it as a WMS and integrate it in the testbed for scheduling and execution of scientific workflows. However, Pegasus WMS cannot run standalone; Pegasus WMS requires that Grid infrastructure is available and the infrastructure is exposed to itself in a specific way. Our testbed is implemented using Globus 5.2.3 (Globus Toolkit, n.d.), Condor 7.8.7 (HTCondor, n.d.), Pegasus WMS 4.1.0 (Pegasus WMS, n.d.), Network Time Protocol, and Dummynet (Dummynet, n.d.). The Globus software is used as a Grid middleware and Condor is used to implement Local Resource Manager. Moreover, Condor-g component of Condor is used for submitting jobs to the Grid-sites, and DAGMan of Condor is used to execute the concrete workflow that is prepared by Pegasus WMS. Pegasus WMS is used for workflow planning and workflow monitoring. Dummynet is used to control bandwidth among the Grid-sites in order to emulate a real network of Grid-sites, say the Grid-sites present in different countries. NTP is used for synchronizing clocks of Grid-sites.

Our Contributions: To carry out research on workflow scheduling aspect, we needed a Grid testbed supporting desired functionalities and administrative control. However, such Grid testbed was not available at our institute. Therefore, we develop a Grid testbed made of open-source and freely available software components. We attempt an important research challenge: many researchers interested in workflow scheduling can not perform experiments on a real Grid due to either unavailability or access of Grid testbed at their institutes or organizations.

Our major contributions in this article are as follows:

- Design a Grid testbed that uses minimal number of resources and still allows the execution of workflow scheduling.
- Discuss constituent software components and their roles in the development of the Grid testbed.
- Present preparation of a real Grid network scenario through network emulation.
- Discuss installation, configuration, integration, and testing of various software in a concise and logical way, rather than showing commands found in how-to guides, to allow beginners to reproduce a similar testbed.
- Demonstrate the applicability of the prepared testbed by scheduling and executing black-diamond workflow of Pegasus WMS; show the effect of data

communication size on makespan by scheduling and executing two workflows having the structure of Molecular dynamics code.

The interested researchers can use or replicate the ready-made design architecture presented in this article for their research purposes. Moreover, all the important details pertaining to the development of the Grid testbed are concisely discussed to enable its replication or customization needed by the interested researchers. To summarize, the article can help to researchers in two significant ways. First, it can help in understanding overall Grid architecture with roles and responsibilities of various involved software. Second, it guides to researchers in implementing a real Grid testbed with minimal hardware resources available with them.

2. Related work and motivations for this work

Some large scale Grid testbeds are available in certain countries, a few examples include Grid'5000 for users of France, EUROGRID for users of Europe region, and Open Science Grid for users of United States and few other countries. The work in (Lai & Yang, 2003) demonstrates building a Grid computing environment on Linux clusters, an in-house Grid testbed. The mentioned work (Lai & Yang, 2003) uses Globus toolkit (Foster & Kesselman, 1997) as a Grid middleware and Sun Grid Engine (Gentzsch, 2001) as a clustering software. Their work demonstrates performance achievement using Grid infrastructure for parallel applications; however, their work does not focus on workflow applications and the workflow scheduling aspect. Similarly, the work in *Introduction to Grid Computing with Globus* (Ferreira et al., 2003) provides the installation steps of building a Grid infrastructure using GT 4; however, it does not focus on workflow scheduling aspect. A recent work in (Sajat et al., 2012) focuses on the implementation steps of achieving security in Grid through Grid Security Infrastructure (GSI). Specifically, their work focuses on installation and testing of host certificates and client certificates and their testing. As compared to (Sajat et al., 2012), our work has wider scope, not just the security in Grid.

Workflow concept allows reusing data analysis operations to solve higher-level analysis problems in various domains; for example, the work in (Turner & Lambert, 2014) addresses use of workflows in social sciences. A Workflow Management System (WMS) can integrate various operations related to workflow modeling, scheduling, execution, and result gathering. Various WMSs support either Directed Acyclic Graph based workflow or Control Flow Graph based workflow or both. Triana (Taylor et al., 2004), GridAnt (Amin et al., 2004) or Karajan (von Laszewski and Hategan, 2005), UNICORE (Erwin & Snelling, 2001), Askalon (Fahringer et al., 2005), and ICENI (Furmento et al., 2002) are

Control Flow Graph based WMSs, and DAGMan (Frey, 2002), Taverna (Hull et al., 2006), GrADS (Berman et al., 2001), GridFlow (Cao et al., 2003), Gridbus (Buyya & Venugopal, 2004) are DAG based WMSs. Most of the mentioned systems are Globus based except Taverna, GridFlow, and UNICORE. Furthermore, a few systems also support web-services. However, only a few systems are active in further development of WMSs and in providing help or support to their users, and Pegasus WMS is one of them. Moreover, Pegasus WMS is open-source and freely-available. Therefore, we choose Pegasus WMS, a widely used workflow management system, for performing scheduling of scientific workflows.

Performing workflow scheduling and execution on a real Grid computing environment requires that the needed infrastructure is available. Moreover, performing experiments on a real system takes time and efforts and requires sound understanding of the system. If researchers require the results of workflow scheduling quickly and there is no need to develop a real Grid environment, then simulation based workflow scheduling and execution can become useful. The work in (Pop et al., 2008) provides MONARC based simulation solution for performing decentralized dynamic resource assignment for large scale workflow applications. Their work provides fault tolerant dynamic scheduling, which allows re-scheduling of remaining workflow when some allocated resources fail. The work in (Simion et al., 2007) proposes ICPDP (Improved Critical Path using Descendant Prediction) workflow scheduling algorithm. Moreover, their work adds facility of dependent tasks scheduling in DIOGENES, which was not available in DIOGENES. Their work implements the proposed algorithm in DIOGENES and compares its performance with HLFET, ETF, and MCP algorithms based on total scheduling time, schedule length, and normalized schedule length. SimGrid (Casanova et al., 2008) framework, which is a versatile framework supporting generic functionalities needed for simulating parallel and distributed applications, has been used by many researchers for performing workflow scheduling. Another popular simulation framework is GridSim (Buyya & Murshed, 2002). A workflow scheduling simulator called WorkflowSim (Chen & Deelman, 2012) is available for performing simulation of workflow overhead analysis, job clustering, and job failure analysis in addition to support of optimization of workflow execution. WorkflowSim mainly focuses on job clustering based workflow scheduling.

The following reasons motivated us to work on the development work presented in this article. First, we have worked on a simulation based evaluation of workflow scheduling algorithm (Prajapati & Shah, 2013). However, to validate the practical applicability and to evaluate various workflow scheduling algorithms, a real test environment is needed. As we did not have a real Grid testbed at our institute, we needed to develop a small, representative testbed to experiment with various workflow scheduling algorithms. Second, it is possible to get deployed the needed Grid testbed by

acquiring specialized system administrators; however, for an academic institute it is not a viable solution. Third, we felt that researchers who do not have access to a Grid testbed should not refrain themselves from research in Grid computing. Fourth, certain research projects (European Grid Infrastructure, n.d.; FutureSystems, n.d.; Pordes et al., 2007; SHIWA, n.d.) provide access to their Grid testbeds freely; however, the access is country specific due to their own limitations and policies. Finally, the most important one, though somehow researchers get access to an external Grid testbed, they do not get any freedom of changing any part of any component of the testbed, e.g., changing workflow scheduling algorithm. To evaluate a new algorithm, researchers require to do changes in the system, which is possible only if they own or get control of the testbed.

3. Architecture of the testbed and constituent components

Figure 1 shows a generalized architecture of Grid computing environment with focus on LRMs and connecting them using Grid computing mechanism. A Grid computing environment involves various Grid-sites, generally one organization is considered as one Grid-site. Each organization generally contains a batch-queue controlled cluster, which is exposed to external organizations through Grid computing services and protocols. Each Grid-site contains a Head node, which is accessible through network, generally Internet.

We attempt to implement this Grid computing architecture, shown in Figure 1, consisting of four Grid-sites using four machines, in which each machine represents one Grid-site. The architectural diagram of a Grid testbed supporting workflow scheduling is presented in Figure 2. In a real Grid system, each Grid-site generally involves many computing nodes, as shown in Figure 1. However, we use only one computing node under each Grid-site in our testbed to build the required computing infrastructure with minimal resources. Moreover, a real Grid computing architecture can involve Grid-sites of various countries. Therefore, to have such real network scenario in the prepared Grid testbed, we emulate the bandwidth and latency characteristics of network using dummynet (Carbone & Rizzo, 2010).

Next, we describe each component that we use in our testbed in brief.

3.1 Dummynet as a network emulator

Dummynet (Carbone & Rizzo, 2010) is a link emulator, which supports emulating configurable network environments. It is a kernel level bandwidth shaper, which can work without modifying an existing OS. It is easy to use, as once it is installed into OS, it can be configured using `ipfw` commands. Dummynet can be used to emulate network topologies also, including the emulation of a router device. Dummynet also supports

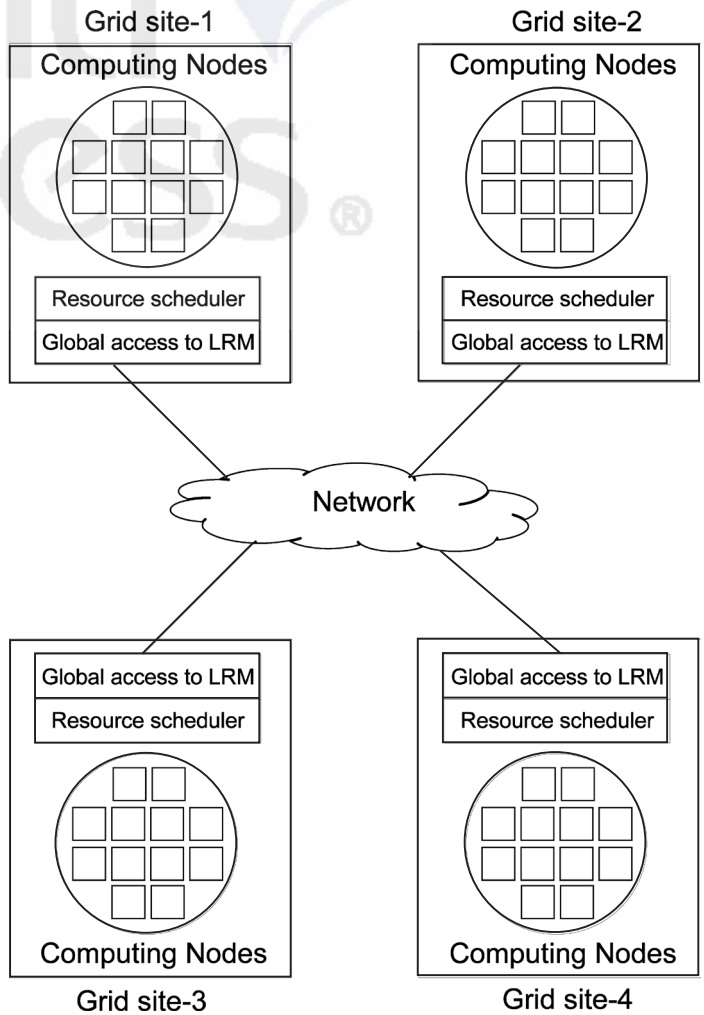

Figure 1 An Architectural Diagram Showing Four Grid-Sites, Each Having Its Own LRM of Computing Nodes under Its Control

packet classification, various queue management policies, and loss generation. Dummynet has two components: emulation engine, which works at kernel level, and packet classifier command (`ipfw`), which instructs the emulation engine. Readers are directed to Section 4.2 for further details.

3.2 HTCondor as an LRM

HTCondor (HTCondor, n.d.), which is formerly known as Condor (Litzkow et al., 1988), is a set of daemons and commands that enable to implement concept of Cluster computing (Buyya, 1999). In this article, the words Condor and HTCondor are used to refer to the same thing. In Condor terminology, the batch queue controlled cluster prepared using Condor is referred by the word Condor Pool. The interaction with Condor system

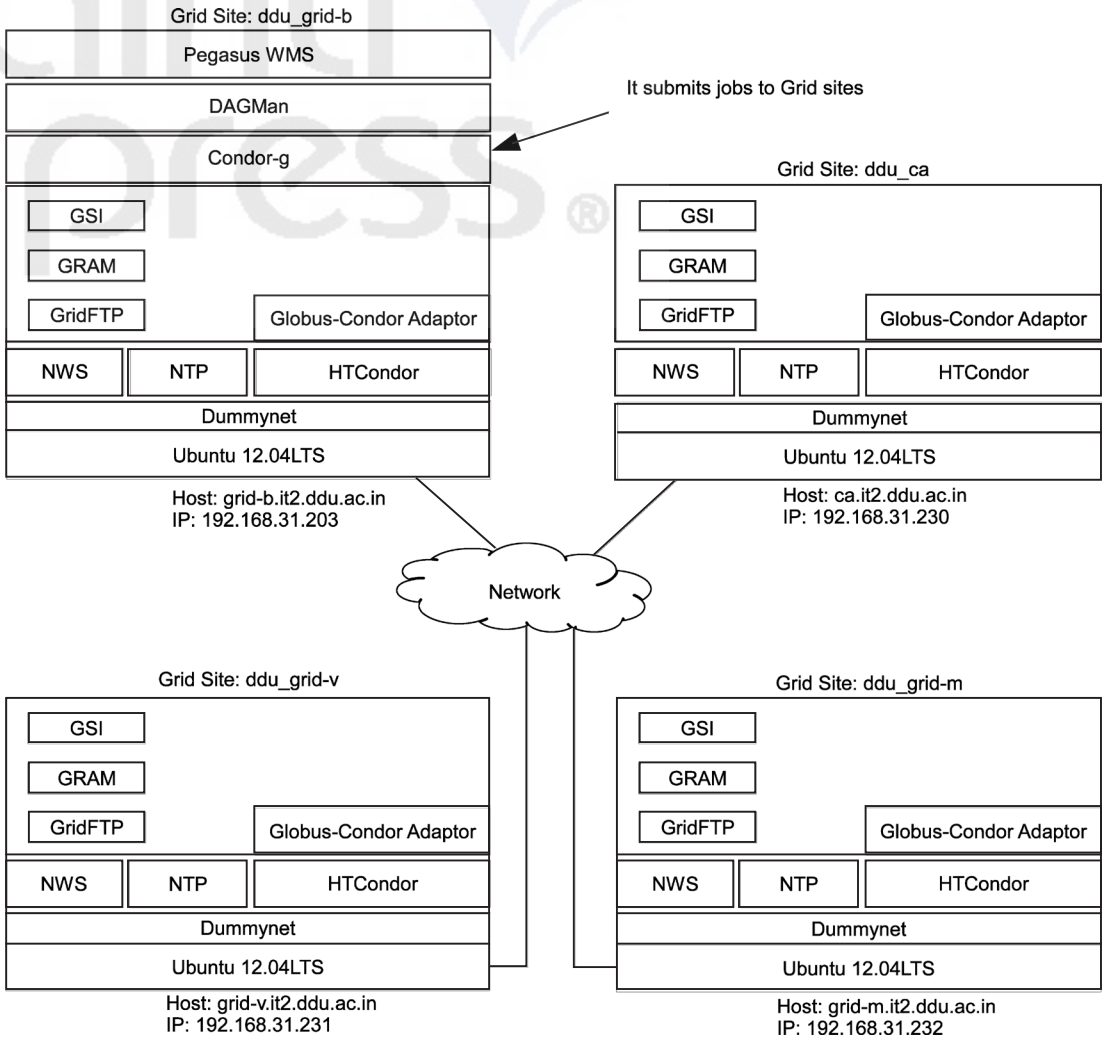

Figure 2 Proposed Architectural Diagram of an In-House Grid Testbed Supporting Workflow Scheduling

for various activities is done through command interface; the internal working of Condor system involves various daemons: Condor master, Condor startd, Condor collector, Condor schedd, Condor negotiator, Condor shadow, and Condor starter (Litzkow et al., 1988). Condor allows job submission, job execution, job monitoring, and input-output data transfer for batch jobs. In our testbed, we expose HTCondor LRMs to grid users through Globus middleware.

3.3 Globus toolkit as a grid middleware

To implement Grid computing (Foster & Kesselman, 2003), a Grid middleware software, which glues different local resources of organization, is needed. We use

Globus toolkit (Foster & Kesselman, 1997), which is a de-facto standard software for implementing Grid computing, as a Grid middleware. The testbed uses Globus for following activities.

- Grid Security (Certificate based authentication, authorization, and single signon) and Simple CA certificate authority for signing host and user certificates.
- GSI (Certificate) based GridFTP (Allcock et al., 2003) (client and server) for transferring data (files) on Grid-sites (machines)
- GRAM client and GRAM server to allow remote job submission using standard protocol -- GRAM.

3.4 Globus to LRM adapter

Each Local Resource Manager, e.g., Condor (Litzkow et al., 1988), SGE (Gentzsch, 2001), or PBS (Henderson, 1995), has its own interface of performing various job management related activities. GRAM (Foster & Kesselman, 2003) is a standard way of accessing a Globus Grid resource. A GRAM-LRM adapter enables usage of any LRM using GRAM. The GRAM-LRM adapter translates GRAM messages into LRM specific messages. Therefore, to access any LRM in a Globus based Grid, the installation of GRAM-LRM adapter is required. We use Globus to Condor adapter for accessing the Grid-sites of the testbed, which are running Condor LRMs.

3.5 NTP for time synchronization

Network Time Protocol (NTP) (NTP, n.d.) is used to synchronize the time of a computer (client or server) to another reliable computer or a reference computer. NTP is a networking protocol for clock synchronization. NTP uses a hierarchy of clock sources. We use NTP in our testbed to have the clocks of all the computers in sync. NTP is needed in the testbed because due to the clock mismatch it is quite possible to send a certificate from one computer to the another on which the start validity period of the certificate has not yet come.

3.6 Condor-g and DAGMan as pre-requisites for Pegasus WMS

Condor-g (Frey et al., 2002) provides an ability to access Grid resources in the Condor way. Therefore, using Condor-g it is possible to use non-Condor Grid resources, such as PBS, SGE, for execution of the jobs that are submitted to a Condor queue. Moreover, Condor-g also enables exploiting job management related features of Condor for the jobs submitted to non-Condor remote Grid resources. Pegasus WMS uses Condor-g for submitting jobs to remote Grid resources. Condor-g communicates with resources and transfers files from and to these Grid resources. Condor-g uses GRAM protocol for job

submission to Grid resources and a local *Global Access to Secondary Storage* (GASS) (Bester et al., 1999) server for file transfers.

DAGMan (Frey, 2002) is HTCondor technology supporting execution of the jobs of a DAG on a Condor pool. It can utilize non-Condor Grid resources such as SGE, PBS, and LSF using Condor-g and can utilize the facility of flocking to get more resources in the pool. DAGMan submits a DAG job as a Condor job to a Condor scheduler. However, a data movement from one Grid-site to another Grid-site is not automatically handled by DAGMan, for which pre-script and post-script need to be associated with the jobs. DAGMan provides fault tolerance through generating a rescue DAG, which can be restarted from the failure point without redoing the earlier computed work.

3.7 Pegasus WMS as a workflow management system

Pegasus WMS (Deelman et al., 2005) is a workflow execution and management software for workflow jobs, which are represented as Directed Acyclic Graph in DAX format. It manages the dependencies of the jobs of a workflow. Pegasus WMS can allow use of one or more Grid resources for execution of the jobs of a particular workflow application. Pegasus WMS uses DAGMan (Frey, 2002) for execution of dependent jobs and Condor-g for job submission. A separate section is devoted to Pegasus WMS, see Section 6 for further details.

4. Deployment of the grid testbed: network configuration and network emulation

The Grid middleware software and other related services/software are available for Linux OSes. Therefore, we use Ubuntu OS for the machines of the Grid testbed, though the details presented on the deployment are applicable to other Linux variants with minor differences in the installation steps or in the OS specific configuration files.

4.1 Network configuration

The testbed includes four personal dual-core computers having Ubuntu 12.04LTS operating system and a networking switch to make a LAN environment. The configuration of the host names and the IP addresses is shown in Table 1. For each machine, the configuration of IP address is done using GUI based Network settings utility available in Ubuntu. The host name of a machine is configured in the `/etc/hostname` file and the mapping of host name to IP address is configured in the `/etc/hosts` file.

Table 1 The Host Names and IP Addresses of Four Computers Used for Deployment of the Grid Testbed

Host Name	IP Address	Fully Qualified Hostname
ca	192.168.31.230	ca.it2.ddu.ac.in
grid-b	192.168.31.203	grid-b.it2.ddu.ac.in
grid-v	192.168.31.231	grid-v.it2.ddu.ac.in
grid-m	192.168.31.232	grid-m.it2.ddu.ac.in

4.2 Network emulation

We use dummynet (Carbone & Rizzo, 2010) to emulate network links. The dummynet emulator is available in source code form (Dummynet, n.d.); therefore it needs to be compiled into binary before installation. We used `20120812-ipfw3.tgz` file, which is available on its web-site. We uncompress this file using the tar command, and we build binary files by running the `make` command. The installation of dummynet includes placing the `ipfw` executable in the `/usr/local/sbin` directory and the `ipfw_mod` kernel module in the directory: `/lib/modules/'uname-r'`. We install dummynet on each computer of the Grid testbed, but we configure each machine with different bandwidth values. Figure 3 shows the additional steps of configuring dummynet in the Grid testbed.

To emulate different bandwidth values among the four machines of the testbed, we configure each machine with different bandwidth value. The `grid-b` has 1,024 kbit/s, `grid-v` has 512 kbit/s, `grid-m` has 256 kbit/s, and ca has 128 kbit/s as bandwidth values. Figure 4 shows how to configure bandwidth control on the `grid-b` machine through the `install-bandwidth-limiter.sh` file, which we create and is not an available configuration file. The bandwidth control uses two pipes: one for the upload bandwidth and the second for the download bandwidth. For other machines, the `install-bandwidth-limiter.sh` file is similar, except for the value of bandwidth and the source and the destination IP addresses. The `pipe 101` is used to control the bandwidth on the traffic that travels from the machine itself to the other three machines of the testbed. Similarly, the `pipe 102` is used to control the bandwidth on the traffic that arrives from

Step 1 : Create two executable files:
/ipfw3/install-bandwidth-limiter.sh file to establish bandwidth control and
/ipfw3/remove-bandwidth-limiter.sh file to remove bandwidth Control
Step 2 : Append following line in /etc/environment file to enable bandwidth control when the system starts.
/ipfw3/install-bandwidth-limiter.sh

Figure 3 Additional Steps of Configuration Dummynet in the Grid Testbed

```
ipfw –q flush
ipfw –q pipe flush
echo “setting delay 100ms and bandwidth 1024kbit/s ”
ipfw pipe 101 config delay 100ms bw 1024kbit/s
ipfw add pipe 101 ip from 192.168.31.203 to 192.168.31.230,192.168.31.231,192.168.31.232
ipfw pipe 102 config delay 100ms bw 1024kbit/s
ipfw add pipe 102 ip from 192.168.31.230,192.168.31.231,192.168.31.232 to 192.168.31.203
ipfw pipe show
```

Figure 4 Content of the Additional File: `install-bandwidth-limiter.sh` on the `grid-b` Machine for Controlling Bandwidth in the Grid Testbed

any other machine of the testbed to the machine itself. When bandwidth control is not needed, the network emulation can be disabled by running the `ipfw -q` flush and `ipfw -q pipe` flush commands.

5. Deployment of the grid testbed: LRM and grid middleware

5.1 Our common procedure for installation of software on Ubuntu

In Ubuntu OS, software that are maintained by various Ubuntu repositories are installed on a computer by connecting the computer to Internet and then running sudo `apt-get install` command. We first installed, without doing any configuration, all the required software and their dependencies on one computer using sudo `apt-get install`. As part of any software installation on a Ubuntu machine using apt tool, all the needed .deb files with dependencies are downloaded into the `/var/cache/apt/archives` directory. Using these downloaded .deb files, for each software we create the required package repository and package indexing, i.e., `Packages.gz` file, using the `dpkg-scanpackages` command. Then, we modify the `/etc/apt/sources.list` file to reflect the locations of various local repositories created in the earlier step. Next, we update the package repository using the `sudo apt-get update` command. Finally, we install a particular software using the `sudo apt-get install <package-name>` command, where `<package-name>` is the name of the chosen software, as if you are connected to Internet.

Installation of non-Ubuntu maintained software on Ubuntu OS involves one additional step of configuring vendor’s repository on the computer. Installation using either the `sudo apt-get install <package-name>` command or through searching in *Ubuntu Software Center* will not succeed, as such software are not maintained by Ubuntu repositories. For non-Ubuntu maintained software, the required repository files (*.deb) for various supported platforms are provided by its software vendor. For configuring the local repository of a computer, first we need to choose appropriate repository file depending

upon the target platform of the computer, characterized by operating system and hardware architecture (32 bit or 64 bit); next, we need to download the chosen repository file to the computer; then, we need to install the repository file using the `sudo dpkg -i <repository-file.deb>` command to update the local repository configuration of the machine.

5.2 Globus installation and configuration

Globus requires Java and Ant as prerequisites. Therefore, we install Java from `jdk-7u9-linux-i586.gz` file and Apache Ant from `apache-ant-1.8.4-bin.tar.gz` file on all the four machines of the testbed. We present the installation and configuration of globus components next.

5.2.1 Roles and corresponding machines

In our testbed, the ca machine, which works as a Certificate Authority using SimpleCA, issues host certificates and user certificates to other machines. The other three machines: `grid-b`, `grid-v`, and `grid-m`, and `ca` play roles of Grid nodes. The Grid node role indicates that a particular machine is a compute node, which allows the execution of a remotely submitted job.

5.2.2 Installation of globus components on the ca machine

We install the `globus-gram5` and `globus-gridftp` packages as per the procedure discussed in Section 5.1. Similarly, on CA, i.e., the `ca.it2.ddu.ac.in` machine, we install `globus-gsi` and `globus-simple-ca` Globus packages. As part of the above installation on CA, the following steps are done automatically by the installer: (1) install Grid Security Infrastructure and Simple CA, (2) create the `simpleca` user automatically, (3) create the self-signed host-certificate, and (4) the `simpleca` user gets `globus`, default one, as the pass-phrase to sign requests of signing host-certificates and user-certificates. We create a user with the name `gtuser` on all the machines, including ca as it also plays role of a Grid node.

5.2.3 Configuration of grid user on the ca machine

On the `ca` machine, as the `gtuser` user we send a request for user certificate using the `grid-cert-request` command. The command prompts the requesting user to enter its name and choose PEM pass-phrase, and generates the following three files: `usercert_request.pem`, `userkey.pem`, and an empty `usercert.pem`. Then, we send the user certificate request (`usercert_request.pem`) file to the `simpleca` user for signing, which the `simpleca` user signs using the `grid-ca-sign` command and sends the generated file back to the `gtuser` user. We as the `gtuser` user store the received signed certificate file under its `/home/gtuser/.globus` directory with the name `usercert.pem`. The `gtuser`

user can verify the installation and configuration using the `grid-proxy-init -debug -verify` command. Next, we do a mapping from a user certificate to a local Linux user account by running the `grid-mapfile-add-entry` command as the `root` user. Finally, we verify working of all components by running `globus-job-run ca.it2.ddu.ac.in/jobmanager-fork/bin/hostname -f` command as the `gtuser` user.

5.2.4 Preparing GSI configuration file on CA for distribution

On the `ca` machine, we as the `simpleca` user create a .deb file containing GSI configuration using the `grid-ca-package` command. The generated .deb file is used to setup GSI on other (Non-CA) machines which will work as Grid nodes. Other Grid nodes install GSI configuration using the `dpkg` command and the generated .deb file.

5.2.5 Installation and configuration on other grid nodes

In this paragraph, we discuss about how we install and configure Globus components on the remaining Grid nodes, i.e., `grid-b`, `grid-v`, and `grid-m`. We install GRAM and GridFTP Globus components using the procedure discussed in Section 5.2.2. Next, as the `root` user, we install CA certificates and signing policy, using the distribution file generated by `simpleca` user on the ca machine in Section 5.2.4, on the `grid-b`, `grid-v`, and `grid-m` machines. Next, on each machine we generate a request for a host certificate and get it signed from SimpleCA. This step is similar to getting a signed user certificate, except that the host certificate request is generated as the user being `root` user. Next, we do the configuration of the Grid user on each machine, for which the used procedure is similar to as discussed in Section 5.2.3.

5.3 Installation and configuration of NTP

We use the `grid-b` machine as a local NTP server and other machines as NTP clients to `grid-b`. After installing NTP on each machine of the testbed, we configure the `/etc/ntp.conf` file. We put following configuration line: `server 127.127.1.0`, on the `grid-b` machine in its `/etc/ntp.conf` file. On the other machines, we put `server 192.168.31.203` as the configuration line in their `/etc/ntp.conf` files, where `192.168.31.203` is the IP address of the `grid-b` machine.

5.4 Installation and configuration of condor components

We use `Condor 7.8.7` as a LRM. In the testbed, as each machine is to work as a Grid-site, we install personal Condor on each machine. We install condor using sudo `apt-get install condor` command by following the procedure discussed in Section 5.1. Next, we verify the installation of Condor by submitting a Condor submit file containing a Vanilla universe job using `condor_submit` command.

To allow submission and execution of remotely submitted jobs, we install GRAM-Condor adapter on each Grid-site of the Grid testbed. We install the adapter using `sudo apt-get install globus-gram-job-manager-condor` command. For each Grid-site, we do testing of GRAM-Condor adapter by submitting a GRAM job from any other machine of the testbed. For example, to test working of GRAM-Condor adapter on the `grid-v` machine, we perform following steps: (1) login on `grid-b` as the `gtuser` user, (2) create proxy credentials using `grid-proxy-init`, and (3) submit a job to `grid-v` using `globus-job-run`.

Pegasus WMS uses the *Grid universe* for a job to allow its execution and monitoring on remote Grid-site in the condor way. Condor-g component of HTCondor allows the Grid universe job to be submitted using `condor_submit`. The condor-g component gets automatically installed when HTCondor is installed. Though a Condor job can be submitted to a remote Condor LRM using `globus-job-run`, the Pegasus WMS uses Condor-g for submission of remote jobs to avail facilities of job monitoring and fault tolerance for the submitted jobs on the submit site in the unified way -- the Condor way.

6. Deployment of the grid testbed: Pegasus WMS and its configuration for the grid testbed

Pegasus WMS was developed by University of Southern California in collaboration with the Condor team of University of Wisconsin Madison. Pegasus WMS is maintained by the Pegasus team. Pegasus WMS is available freely in binary form for different platforms; moreover, it is open source and it relies on other open source software. Pegasus WMS is made available for use since 2001. We use Pegasus WMS version 4.1 for preparing the Grid testbed supporting workflow scheduling. In this section, we first concisely describe Pegasus WMS and then present the configuration that we do in the Grid testbed to achieve scheduling of scientific workflow application.

6.1 Pegasus WMS as a workflow planner and DAGMan as a workflow executor

Pegasus WMS is a workflow execution and management software for workflow jobs that are represented as Directed Acyclic Graph. It manages dependencies of jobs. It can allow use of one or more Grid-sites for execution of the jobs of a workflow application. Pegasus WMS can use any Grid resource or a Grid-site that can be accessed using GRAM. Pegasus WMS uses, on the workflow submit site, Condor-g for job submission and Condor DAGMan for execution of a DAG. Pegasus WMS does planning of the jobs of a workflow, represented in .dax format, and generates a concrete workflow in form of .dag file and Condor-g submit files, and passes them to DAGMan for execution.

DAGMan itself cannot decide about which Grid-site runs which jobs, though the site can utilize any available machine of the cluster once the job is submitted to a particular Grid-site. DAGMan can not take decision about which Grid-site is best to run a particular job. Therefore, Pegasus WMS complements DAGMan for supporting scheduling and execution of the jobs of a workflow on Grid resources. Pegasus WMS chooses appropriate Grid resources for execution of jobs, which is done by a site-selector algorithm, and generates codes for file movement, data cleanup, data registration, etc. Moreover, Pegasus WMS provides the monitoring of a running workflow and also provides the provenance and performance related information.

6.2 Working of Pegasus WMS

We briey highlight on working of PegasusWMS. Pegasus WMS's `pegasus-plan` command takes an abstract workflow (DAX) as an input and does planning (deciding which task runs on which Grid-site) of tasks of the workflow based on the information provided in catalogs. This selection decision is taken by a site-selector algorithm, which we configure in the property file -- `.pegasusrc` file in our testbed, of Pegasus WMS. The `pegasus-plan` command uses physical locations specified for input files from the replica catalog. The `pegasus-plan` command prepares the DAGMan .dag file and Condor job submit files based on the information provided in the transformation catalog. The prepared Condor submit files are in fact Condor-G submit files having `Grid` as `Universe`. The `pegasus-plan` command adds the additional jobs for creating a workflow directory, transferring intermediate files, registering the final data and intermediate data in replica catalog, and cleanup activities. Pegasus WMS's `pegasus-run` command runs the jobs of the planned workflow on the chosen Grid-sites. The `pegasus-run` command itself does not run the jobs of the workflow, rather Pegasus WMS relies on DAGMan, which is a workflow executor for Pegasus WMS.

When a workflow is started using the `pegasus-run` command, the command starts the monitoring daemon (`pegasus-monitord`) in the directory (called workflow directory and is created by `pegasus-plan`) containing the condor submit files. The `pegasus-monitord` daemon parses the condor output files and updates the status of the workflow to a database and to the `jobstate.log` text file.

6.3 Installation and configuration of Pegasus WMS in the grid testbed

6.3.1 Installation of Pegasus WMS

As the `root` user, we install Pegasus WMS on the machines, which are having 32-bit Ubuntu OS, using `pegasus 4.1.0-2 i386.deb` file, which we generated from .rpm file using alien tool, as .deb file for 32-bit, Intel architecture was not available. As discussed earlier, we use the four Grid nodes as four Grid-sites. We assign the names of these sites

as ddu_grid-b, ddu_grid-v, ddu_grid-m, and ddu_ca, and configure these names in the *Site Catalog*. All these four Grid-sites play role of an Executor and ddu_grid-b plays role of a Submit and an Output site, in addition to Executor.

6.3.2 Configuration of directory structure and files

The Pegasus WMS needs a *scratch* directory and a *storage* directory on each Gridsite. The `pegasus-plan` creates create dir jobs, one per Grid-site, which will create a workflow directory under *scratch* directory on each Grid-site that will run jobs of the workflow. The *storage* directory is used to hold output data of workflows. We create `/scratch` as a *scratch* directory and `/storage` as a *storage* directory, both having all permissions, on each Grid-site of the testbed. We configure the locations of these directories in *Site Catalog*.

We configure Pegasus WMS for the gtuser user on the submit Grid-site, as we have configured the `gtuser` user as the user of the Grid testbed. Under the home directory of the `gtuser` user, i.e., `/home/gtuser`, we create the `.pegasusrc` file containing initialization of various properties. The `.pegasusrc` property file contains the configuration of locations of Replica Catalog, Site Catalog, and Transformation Catalog. Furthermore, we also specify the `pegasus.selector.site` property indicating which site-selector algorithm is used for mapping the jobs of a workflow in this file.

Figure 5 shows the directory structure used for keeping the configuration files, the input files to Pegasus WMS, and the output files generated by Pegasus WMS. The `rc.data` file under the directory `/home/gtuser/pegasus-wms/config` is a text based *Replica Catalog*. The `sites.xml3` file under `/home/gtuser/pegasu-s-wms/config` is an XML based *Site Catalog*. The `tc.data.text` file under `/home/gtuser/pegasus-wms/config` is a text based *Transformation Catalog*. The `/home/gtuser/pegasus-wms/local-scratch` is the directory to hold temporary files created for the local site and the directory `/home/gtuser/pegasus-wms/local-storage` is used to store data files. When we use the `pegasus-plan` command to do planning of an abstract workflow, we make the `/home/gtuser/pegasus-wms/` directory as the current working directory so that the paths to the .dax file and the workflow run directory can be specified relative to this directory. The `/home/gtuser/pegasus-wms/dax` directory is used to keep abstract workflow files in DAX (XML) format. The `workflow-run` directory is used, specified using `--dir` option to `pegasus-plan`, by the `pegasus-plan` to store generated submit files produced by planning of the workflow. The `pegasus-plan` creates a separate directory for each planning done by it under `workflow-run` directory, which `pegasus-run` uses for executing the planned concrete workflow. The `input-data` directory is used to hold data files of workflows, whose locations are specified in *Replica Catalog*.

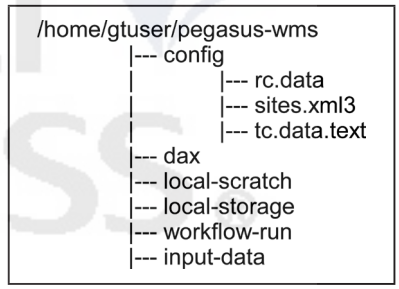

Figure 5 Directory Structure Used on Submit-Site (ddu_grid-b) for Configuration of Pegasus WMS

7. Testing and use of the testbed

7.1 Methodology of experimentations

Figure 6 shows the steps of the methodology used to carry out experiments on the Grid testbed supporting workflow scheduling, which includes four major phases of performing experimentation. These phases in sequence are as follows: (1) experiment configuration, (2) workflow planning, (3) workflow execution, and (4) result generation. During experiment configuration in Pegasus WMS, the steps of computations of a scientific workflow are specified as an abstract workflow in DAX, an XML file. Next, site catalog, transformation catalog, and replica catalog are set up. Next, the site-selector algorithm is chosen by assigning appropriate name of siteselector class, i.e., workflow scheduling algorithm, to the `pegasus.selector.site` property in `.pegasusrc` file. The next two steps are related to planning of a workflow.

Once the concrete workflow is generated at the end of workflow planning, the execution and monitoring of the workflow can be started. For workflow execution, first, the Grid user, in our testbed the `gtuser` user, needs to create proxy credentials to allow access of Grid resources to the jobs submitted by `pegasus-run` on behalf of the user. Next, the `pegasus-run` command is executed to start execution of the jobs of the planned workflow on the Grid-sites. The execution of the workflow can be monitored by running the `pegasus-status` command under control of watch command. An optional step of using `pegasus-analyzer`, which is not shown in the diagram, can be used to debug a failed workflow. If the execution of the workflow is completed, various plots are generated using the `pegasus-plots` command and statistics of the workflow run is collected using the `pegasus-statistics` command.

7.2 Testing of various components and workflow scheduling and execution

The complete testing of the whole testbed includes testing of many constituent

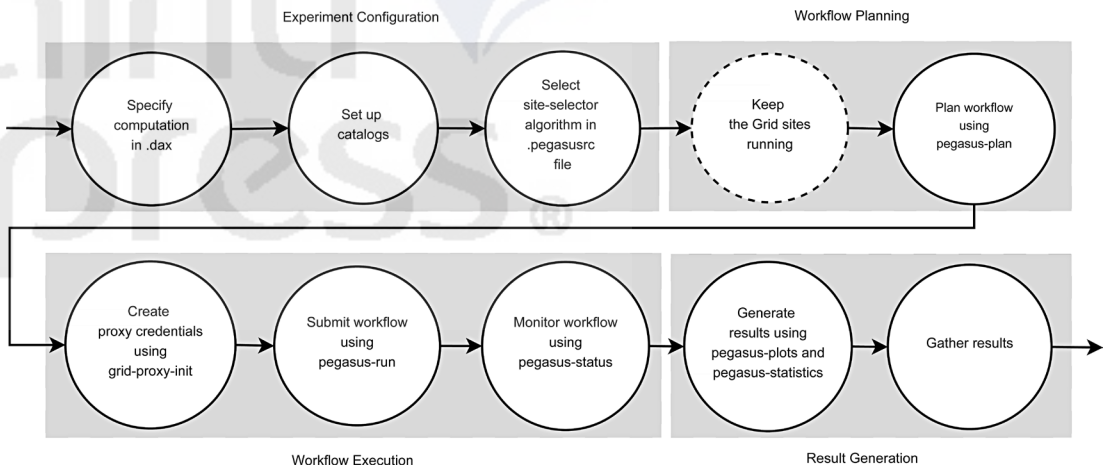

Figure 6 Methodology for Performing Experiments Using the Grid Testbed and Pegasus WMS. The Step in Dotted Line Can Be Performed after Pegasus-Plan, but before Pegasus-Run

components and their functionalities. We list out various testings that we carried out for our testbed. Then, we provide testing of workflow scheduling and workflow execution. The work in (Prajapati & Shah, 2014b) addresses remote job submission to LRM using Grid computing mechanisms. We have performed following testings on each Grid-site to test working of various components.

- Testing of GSI for proxy credential generation using `grid-proxy-init` and `grid-proxy-info`
- Testing of Condor LRM for job execution using `condor_submit` and monitoring using `condor_q`
- Testing of gridftp using `globus-url-copy` for for file transfer
- Testing of fork job manager using `globus-job-run`
- Testing of Condor job manager (Globus-Condor adapter) for job execution using `globus-job-run`
- Testing of Condor-g for remote job submission using `condor_submit` and `Universe=Globus` in condor submit file

We perform testing of Pegasus WMS by scheduling and running the blackdiamond workflow on the four Grid-sites: `ddu_grid-b`, `ddu_grid-v`, `ddu_grid-m`, and `ddu_ca`. We create the concrete workflow for RoundRobin site-selector algorithm using `pegasus-plan` by passing the four Grid-sites to `--sites` option. If the planning is successful,

`pegasus-plan` will concretize the workflow and will create a unique directory, for which time-stamp value is used as the name of the directory, containing job submit files and DAGMan submit file. To start execution of the planned workflow, we run the workflow using the `pegasus-run` command. While the workflow is executing, we can see status of executing workflow using `pegasus-status`. To check status of activities the four Grid-sites are doing for black diamond workflow at particular instant of time, we log on to three other machines (`grid-v`, `grid-m`, and `ca`) using `ssh` from the `grid-b` machine.

Next, we generate various charts using the `pegasus-plots` command. The `pegasus-plots` command will generate various plots under the plots sub-directory under the workflow run-directory. Figure 7 shows Gantt chart showing the assignment of the workflow jobs on the four Grid-sites using RoundRobin site-selector algorithm.

7.3 Workflow scheduling and execution of molecular dynamics code workflow

Using the way of creating the black-diamond workflow, we prepare two workflows of 41 tasks based on the structure of the task-graph of Molecular dynamics code, which is available in (Topcuouglu et al., 2002). For both the test workflows, we choose the runtime of each task randomly in the range 243 ~ 357 seconds. For both the workflows, the cumulative runtime of all the tasks is 12,219 seconds and the average runtime is 298.02 seconds. However, we keep different amount of data communication among the tasks of these two test workflows to observe their effect on makespan. For the first test workflow, the amount of data communication between two tasks is chosen randomly in the range 9,879,000 ~ 15,698,940 Bytes; the Communication to Computation Ratio (CCR) for this workflow is $CCR \approx 0.65$. For the second test workflow, the amount of data communication between each pair of tasks is small, specifically, it is in the range 987,900 ~ 1,569,894

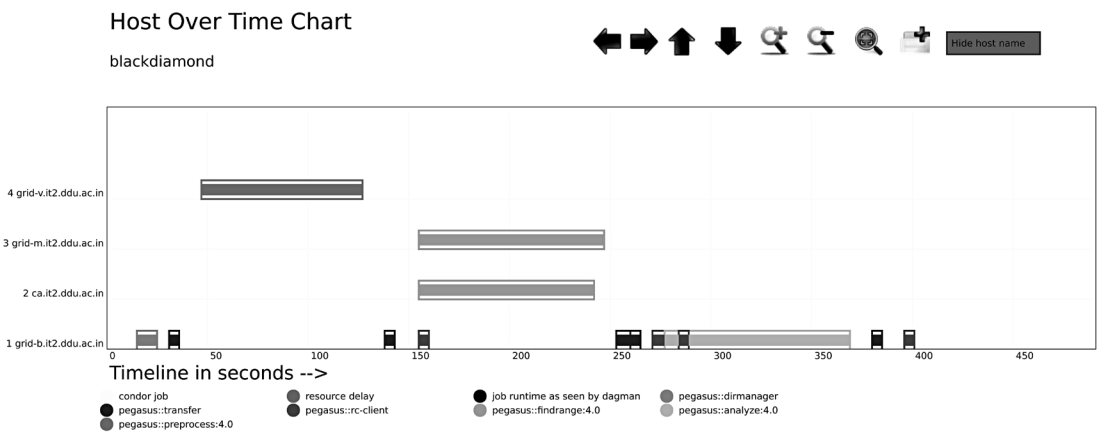

Figure 7 Host over Time Chart for Black-Diamond Workflow Using RoundRobin Site-Selector Algorithm

Bytes. Thus, the CCR for the second workflow is $CCR \approx 0.065$. We schedule and execute both the test workflows using Random site-selector algorithm in the similar way, as discussed earlier. For the first test workflow, the makespan of 23,261 seconds is observed. However, for the second test workflow, having small data communication, the makespan of 6,308 seconds is observed. Figure 8 shows the Gantt chart showing the assignment of the jobs of the second test workflow on the four Grid-sites using Random site-selector algorithm. From the experimental results, we can understand that by increasing the amount of data communication among the workflow tasks, the makespan of workflow execution increases due to relatively higher time spent in data communication. Thus, the prepared testbed can allow experimentation of workflow scheduling and execution on a real Grid environment.

This testbed was used in (Prajapati & Shah, 2014a) for performing bandwidth-aware workflow scheduling of scientific workflow.

7.4 Scalability and failure handling in the testbed

Two important problems: scalability of a testbed and failure of resources in the testbed can affect to workflow scheduling and execution. In our testbed, we have added a single node under each Grid site, as show in Figure 2, for demonstration purpose; however, it is possible to add many nodes under each Grid site to make the testbed more scalable. Workflow scheduling time depends on the number of Gridsites, not on the number of computing nodes. Therefore adding computing nodes under a Grid-site will not affect to workflow scheduling time. Moreover, adding Grid-sites will affect workflow scheduling time polynomially depending upon the workflow scheduling algorithm. Failure

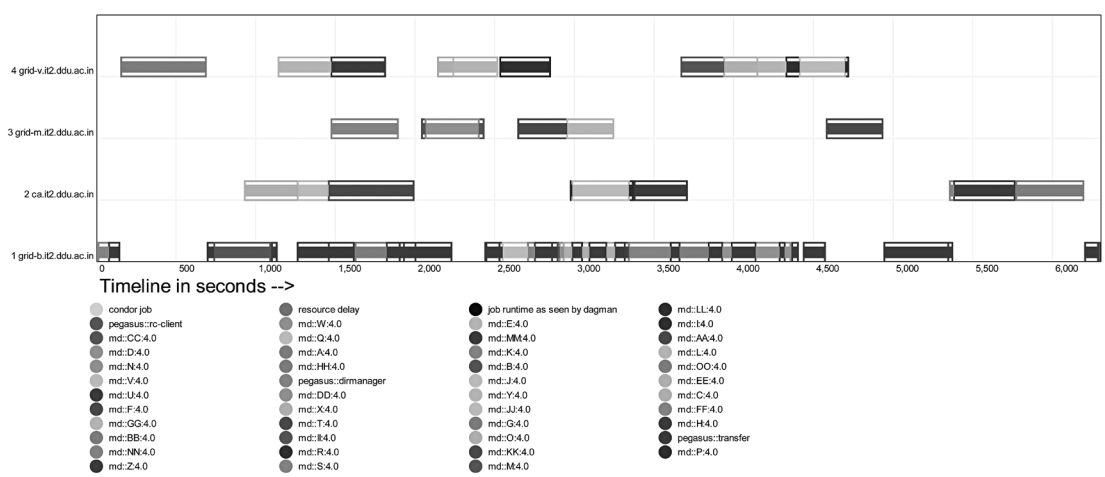

Figure 8 Host over Time Chart for Molecular Dynamics Code Workflow Having Small Data Communication Using Random Site-Selector Algorithm

of resources can happen at two levels: (1) failure of a computing node and (2) failure of a Grid-site. The failure of a computing node does not affect to workflow execution, as the failed job can be handled by the LRM by restarting it on some other available computing node under its control. However, failure of a Grid-site can affect to workflow execution depending upon how the used WMS deal with the Grid-site failure. In case of Pegasus WMS, the failure of a job is detected by the monitoring daemon (`pegasus-monitord`). The user can debug the failed workflow execution using `pegasus-analyzer`, which provides information related to the failure.

8. Conclusion

Our work designed and developed an in-house Grid testbed using widely used open-source software packages/tools, including Globus toolkit 5.2.3, HTCondor 7.8.7, NTP, and Pegasus WMS 4.1.0, to experiment workflow scheduling and execution. Furthermore, the testbed emulated a real Grid scenario of bandwidth variation among various Grid-sites using dummynet. Building a Grid environment requires the understanding of the concepts related to network, protocols, services, etc. However, due to proper study and experimentation with individual software components, we are able to produce a usable Grid testbed, having minimal physical resources, for carrying workflow scheduling and execution. Furthermore, through the experimental work, we are able to provide concise understanding of various involved software, their installation, their configuration, and their testing.

Our testbed has only four computing machines, in which each machine works as a Grid-site as well as a computing node. It is possible to add additional computing nodes in each Grid-site. The added computing machines need to become part of the batch-queue cluster that is under control of a particular Grid-site. For example, in our testbed we need to install Condor on each additional computing machine and we need to make each added computing machine to respond to the central manager of a particular Grid-site. Thus, the presented Grid testbed can easily be replicated or adapted, as the work concisely included all important details pertaining to the development and the deployment of the Grid architecture supporting workflow scheduling. Moreover, we can easily include a large number of computing nodes under each Grid-site to achieve better reliability. There are other WMSs, e.g., Askalon (Fahringer et al., 2005), Kepler (Altintas et al., 2004), and Karajan (von Laszewski & Hategan, 2005), for which similar deployment and testing of a Grid testbed can provide substantial help to novice researchers.

Acknowledgements

The authors would like to thank the Pegasus WMS team for providing answers of various questions related to deployment of Pegasus WMS. The answers of raised questions have helped a lot to the authors in quickly digesting the internal working of Pegasus WMS.

References

Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L. and Tuecke, S. (2003), 'GridFTP: protocol extensions to FTP for the grid', GFD-RP 20, Global Grid Forum, Muncie, IN.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B. and Mock, S. (2004), 'Kepler: an extensible system for design and execution of scientific workflows', *Proceedings of 16th International Conference on Scientific and Statistical Database Management*, Santorini Island, Greece, pp. 423-424.

Amin, K., Von Laszewski, G., Hategan, M., Zaluzec, N.J., Hampton, S. and Rossi, A. (2004), 'Gridant: a client-controllable grid workflow system', *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Big Island, HI, doi: 10.1109/HICSS.2004.1265491.

Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., et al. (2001), 'The grads project: Software support for high-level grid application development', *International Journal of High Performance Computing Applications*, Vol. 15, No. 4, pp. 327-344.

Bester, J., Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S. (1999), 'Gass: a data movement and access service for wide area computing systems', *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, Atlanta, GA, pp. 78-88.

Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D. and Luitz, J. (2014), *Wien2k. An Augmented Plane Wave Plus Local Orbitals Program for Calculating Crystal Properties*, Vienna University of Technology, Vienna, Austria.

Buyya, R. (1999), *High Performance Cluster Computing: Architectures and Systems, Volume 1*, Prentice Hall, Upper Saddle River, NJ.

Buyya, R. and Murshed, M. (2002), 'Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing', *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp. 1175-1220.

Buyya, R. and Venugopal, S. (2004), 'The Gridbus toolkit for service oriented grid and utility computing: an overview and status report', *Proceedings of 1st IEEE International Workshop on Grid Economics and Business Models*, Seoul, Korea, pp. 19-66.

Cao, J., Jarvis, S.A., Saini, S. and Nudd, G.R. (2003), 'GridFlow: workflow management for grid computing', *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Tokyo, Japan, pp. 198-205.

Carbone, M. and Rizzo, L. (2010), 'Dummynet revisited', *ACM SIGCOMM Computer Communication Review*, Vol. 40, No. 2, pp. 12-20.

Casanova, H., Legrand, A. and Quinson, M. (2008), 'Simgrid: a generic framework for large-scale distributed experiments', *Proceedings of Tenth International Conference on Computer Modeling and Simulation*, Cambridge, UK, pp. 126-131.

Chen, W. and Deelman, E. (2012), 'WorkflowSim: a toolkit for simulating scientific workflows in distributed environments', *Proceedings of 2012 IEEE 8th International Conference on E-Science (e-Science)*, Chicago, IL, pp. 1-8.

Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., et al. (2005), 'Pegasus: a framework for mapping complex scientific workflows onto distributed systems', *Scientific Programming*, Vol. 13, No. 3, pp. 219-237.

Dong, F. and Akl, S. G. (2006), 'Scheduling algorithms for grid computing: state of the art and open problems', Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Canada.

Dummynet. (n.d.), 'The dummynet project', available at http://info.iet.unipi.it/~luigi/dummynet/ (accessed 10 February 2015).

Erwin, D.W. and Snelling, D.F. (2001), 'UNICORE: a grid computing environment', *Proceedings of Euro-Par 2001*, Manchester, UK, pp. 825-834.

European Grid Infrastructure. (n.d.), 'European Grid Infrastructure', available at http://www.egi.eu/ (accessed 10 February 2015).

Exon. (n.d.), 'Blast workflow', available at http://exon.niaid.nih.gov/cas/manual/blast-workflow.html (accessed 10 February 2015).

Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Seragiotto, C. and Truong, H.L. (2005), 'Askalon: a tool set for cluster and grid computing', *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pp. 143-169.

Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., Takagi, M., et al. (2003), *Introduction to Grid Computing with Globus*, IBM, New York, NY.

Foster, I. and Kesselman, C. (1997), 'Globus: a metacomputing infrastructure toolkit', *International Journal of High Performance Computing Applications*, Vol. 11, No. 2, pp. 115-128.

Foster, I. and Kesselman, C. (Eds.). (2003), *The Grid: Blueprint for a New Computing Infrastructure*, 2nd ed., Elsevier, Oxford, UK.

Foster, I., Kesselman, C. and Tuecke, S. (2001), 'The anatomy of the grid: enabling scalable virtual organizations', *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200-222.

Frey, J. (2002), 'Condor DAGMan: handling inter-job dependencies', Technical Report, Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI.

Frey, J., Tannenbaum, T., Livny, M., Foster, I. and Tuecke, S. (2002), 'Condor-g: a computation management agent for multi-institutional grids', *Cluster Computing*, Vol. 5, No. 3, pp. 237-246.

Furmento, N., Lee, W., Mayer, A., Newhouse, S. and Darlington, J. (2002), 'ICENI: an open grid service architecture implemented with Jini', *Proceedings of 2002 ACM/IEEE Conference on Supercomputing*, Baltimore, MD, doi: 10.1109/SC.2002.10027.

FutureSystems. (n.d.), 'FutureGrid is now FutureSystems', available at https://portal.futuresystems.org/ (accessed 10 February 2015).

Gentzsch, W. (2001), 'Sun grid engine: towards creating a compute power grid', *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, pp. 35-36.

Globus Toolkit. (n.d.), 'Installing GT 5.2.3', available at http://www.globus.org/toolkit/docs/5.2/5.2.3/admin/install/ (accessed 10 February 2015).

Henderson, R.L. (1995), 'Job scheduling under the portable batch system', *Proceedings of IPPS 1995*, Santa Barbara, CA, pp. 279-294.

HTCondor. (n.d.), 'Computing with HTCondor™', available at http://research.cs.wisc.edu/htcondor/ (accessed 10 February 2015).

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M.R., Li, P., et al. (2006), 'Taverna: a tool for building and running workflows of services', *Nucleic Acids Research*, Vol. 34, pp. W729-W732.

Lai, C.L. and Yang, C.T. (2003), 'Construct a grid computing environment on multiple Linux PC clusters', *Tunghai Science*, Vol. 5, pp. 107-124.

LIGO. (n.d.), 'LIGO laser interferometer gravitational-wave observatory', available at http://www.ligo.caltech.edu/ (accessed 10 February 2015).

Litzkow, M.J., Livny, M. and Mutka, M.W. (1988), 'Condor-a hunter of idle workstations', *Proceedings of 8th International Conference on Distributed Computing Systems*, San Jose, CA, pp. 104-111.

Montage. (n.d.), 'Grid tools', available at http://montage.ipac.caltech.edu/docs/gridtools.html (accessed 10 February 2015).

NTP. (n.d.), 'NTP: the network time protocol', available at http://www.ntp.org/ (accessed 10 February 2015).

Pegasus WMS. (n.d.), 'Pegasus', available at http://pegasus.isi.edu/projects/pegasus (accessed 10 February 2015).

Pinedo, M.L. (2008), *Scheduling: Theory, Algorithms and Systems*, 3rd ed., Springer, New York, NY.

Pop, F., Dobre, C. and Cristea, V. (2008), 'Decentralized dynamic resource allocation for workflows in grid environments', *Proceedings of 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, pp. 557-563.

Pordes, R., Petravick, D., Kramer, B., Olson, D., Livny, M., Roy, A., et al. (2007), 'The open science grid', *Journal of Physics: Conference Series*, Vol. 78, doi: 10.1088/1742-6596/78/1/012057.

Prajapati, H.B. and Shah, V.A. (2013), 'Advance reservation based dag application scheduling simulator for grid environment', *International Journal of Computer Applications*, Vol. 61, No. 7, pp. 45-51.

Prajapati, H.B. and Shah, V.A. (2014a), 'Bandwidth-aware scheduling of workflow application on multiple grid sites', *Journal of Computer Networks and Communications*, Vol. 2014, doi: 10.1155/2014/529835.

Prajapati, H.B. and Shah, V.A. (2014b), 'Experimental study of remote job submission and execution on LRM through grid computing mechanisms', *Proceedings of 2014 Fourth International Conference on Advanced Computing & Communication Technologies*, Rohtak, India, pp. 335-341.

Prajapati, H.B. and Shah, V.A. (2014c), 'Scheduling in grid computing environment', *Proceedings of 2014 Fourth International Conference on Advanced Computing & Communication Technologies*, Rohtak, India, pp. 315-324.

Sajat, M.S., Hassan, S., Ahmad, A.A., Daud, A.Y. and Ahmad, A. (2012), 'Implementing a secure academic grid system -- a Malaysian case', *Proceedings of the 10th Australian Information Security Management Conference*, Perth, Australia, pp. 59-65.

SHIWA. (n.d.), 'SHaring interoperable workflows for large-scale scientific simulations on available DCIs', available at http://www.shiwa-workflow.eu/ (accessed 10 February 2015).

Simion, B., Leordeanu, C., Pop, F. and Cristea, V. (2007), 'A hybrid algorithm for scheduling workflow applications in grid environments (ICPDP)', *Proceedings of OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007*, Vilamoura, Portugal, pp. 1331-1348.

Taylor, I., Deelman, E., Gannon, D.B. and Shields, M. (Eds.). (2007), *Workflows for e-Science: Scientific Workflows for Grids*, Springer, New York, NY.

Taylor, I., Shields, M. and Wang, I. (2004), 'Resource management for the Triana peer-to-peer services', in Nabrzyski, J., Schopf, J.M. and Węglarz, J. (Eds.), *Grid Resource Management*, Springer, New York, NY, pp. 451-462.

Topcuouglu, H., Hariri, S. and Wu, M.-y. (2002), 'Performance-effective and lowcomplexity task scheduling for heterogeneous computing', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, pp. 260-274.

Turner, K.J. and Lambert, P.S. (2014), 'Workflows for quantitative data analysis in the social sciences', *International Journal on Software Tools for Technology Transfer*, doi: 10.1007/s10009-014-0315-4.

von Laszewski, G. and Hategan, M. (2005), 'Workflow concepts of the java cog kit', *Journal of Grid Computing*, Vol. 3, No. 3-4, pp. 239-258.

Wieczorek, M., Prodan, R. and Fahringer, T. (2005), 'Scheduling of scientific workflows in the askalon grid environment', *ACM SIGMOD Record*, Vol. 34, No. 3, pp. 56-62.

Yu, J., Buyya, R. and Ramamohanarao, K. (2008), 'Workflow scheduling algorithms for grid computing', in Xhafa, F. and Abraham, A. (Eds.), *Metaheuristics for Scheduling in Distributed Computing Environments*, Springer, Berlin, Germany, pp. 173-214.

About the authors

Harshadkumar B. Prajapati is an Associate Professor at the Department of Information Technology, Faculty of Technology, Dharmsinh Desai University, India. He received a BE in Electronics and Communication from Gujarat University, India, in 2000 and an ME in

Computer Engineering from Dharmsinh Desai University in 2007. He recently completed PhD in Computer Engineering from Dharmsinh Desai University. He has published several papers in international conferences and journals. He has served as a reviewer in international journals and conferences. His research areas include distributed computing, grid computing, and workflow scheduling.

Corresponding author. Department of Information Technology, Dharmsinh Desai University, Nadiad-387001, Gujarat, India. Tel: +91-268-2520502. E-mail address: harshad.b.prajapati@gmail.com

Vipul A. Shah is a Professor at the Department of Instrumentation and Control Engineering, Faculty of Technology, Dharmsinh Desai University, India. He received a BE degree in Instrumentation and Control Engineering in the year 1991 and an ME degree in Microprocessor Systems and Applications in the year 1995. He received a PhD in Instrumentation and Control Engineering from Dharmsinh Desai University in the year 2006. His research areas include Artificial Intelligence, Robotics, Machine Control, System Design, Advanced Control Theory, Process Control, and Distributed Control. E-mail address: vashahin2010@gmail.com