GridSim: A Toolkit for the Modeling and Simulation of Global Grids

Manzur Murshed[†], Rajkumar Buyya[*], and David Abramson[*]

[†]Gippsland School of Computing and IT
Monash University, Gippsland Campus
Churchill, Vic 3842, Australia
Manzur.Murshed@infotech.monash.edu.au

[*] School of Computer Science and Software Eng.
Monash University, Caulfield Campus
Melbourne, Vic 3145, Australia
{rajkumar, davida}@csse.monash.edu.au

Abstract: Clusters, Grids, and Peer-to-Peer (P2P) networks have emerged as popular paradigms for next generation parallel and distributed computing. They enable aggregation of distributed resources for solving large-scale problems in science, engineering, and commerce. In Grid and P2P computing environments, the resources are usually geographically distributed in *multiple administrative domains*, managed and owned by different organizations with different policies, and interconnected by wide-area networks or the Internet. This introduces a number of resource management and application scheduling challenges in the domain of security, resource and policy heterogeneity, fault tolerance, continuously changing resource conditions, and politics. The resource management and scheduling systems for Grid computing need to manage resources and application execution depending on either resource consumers' or owners' requirements, and continuously adapting to changes in resource availability.

The management resources and scheduling of applications in such a large-scale distributed systems is complex undertaking. In order to prove the effectiveness of resource brokers and associated scheduling algorithms, their performance need to evaluated under different scenarios such as varying number of resources and users with different requirements. In Grid environment, it is hard and even impossible to perform scheduler performance evaluation in a *repeatable* and *controllable* manner as resources and users are distributed across multiple organizations with their own policies. To overcome this limitation, we have proposed and developed a Java-based discrete-event Grid simulation toolkit called *GridSim*. The toolkit supports modeling and simulation of heterogeneous Grid resources (both time and space-shared), users and application models. It provides primitives for creation of application tasks, mapping of tasks to resources and their management. To demonstrate suitability of the GridSim toolkit, we have simulated a Nimrod-G like grid resource broker and evaluated the performance of deadline and budget constrained cost- and time-minimization scheduling algorithms.

1 Introduction

The proliferation of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we do large-scale parallel and distributed computing. The interest in coupling geographically distributed (computational) resources is also growing for solving large-scale problems, leading to what is popularly called the Grid [3] and peer-to-peer (P2P) computing [4] networks. These enable sharing, selection and aggregation of suitable computational and data resources for solving large-scale data intensive problems in science, engineering, and commerce. A generic view of Grid computing environment is shown in Figure 1. The Grid consists of four key layers of components: fabric, core middleware, user-level middleware, and applications [6]. The Grid fabric includes computers (low-end and high end computers including clusters), networks, scientific instruments, and their resource management systems. The core Grid middleware provides services that are essential for securely accessing remote resources uniformly and transparently. The services they provide include security and access management, remote job submission, storage, and resource information. The user-level middleware provides higher-level tools such as resource brokers, application development and adaptive runtime environment. The Grid applications include those constructed using Grid libraries or legacy applications

that can be Grid enabled using user-level middleware tools.

The user essentially interacts with a resource broker that hides the complexities of grid computing [7][8]. The broker discovers resources that the user can access using information services, negotiates for access cost using trading services, maps tasks to resources (scheduling), stages the application and data for processing (deployment), starts job execution, and finally gathers the results. It is also responsible for monitoring and tracking application execution progress along with adapting to the changes in Grid runtime environment conditions and resource failures.

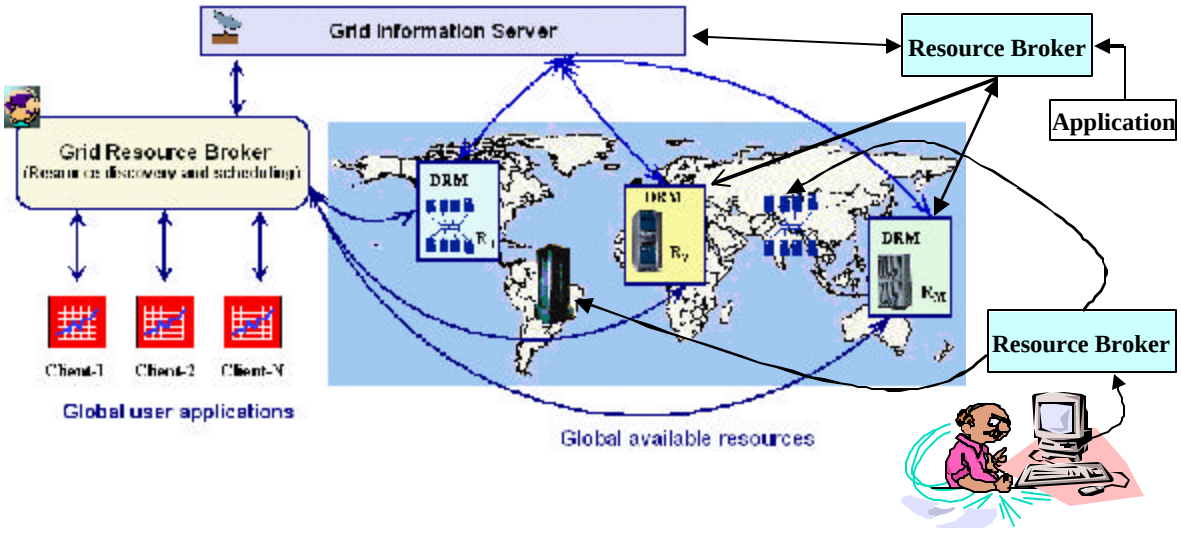

Figure 1: A generic view of Grid computing environment.

The computing environments comprise heterogeneous resources (PCs, work-stations, clusters, and supercomputers), fabric management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering, and commercial) with varied requirements (CPU, I/O, memory, and/or network intensive). The users: *producers* (also called *resource owners*) and *consumers* (also called *end-users*) have different goals, objectives, strategies, and demand patterns. More importantly both resources and end-users are geographically distributed with different time zones. In managing such complex Grid environments, traditional approaches to resource management that attempt to optimize system-wide measures of performance cannot be employed. This is because traditional approaches use centralized policies that need complete state information and a common fabric management policy, or decentralized consensus based policy. In large-scale Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy. Apart from the centralized approach, two other approaches that are used in distributed resource management are: *hierarchical* and *decentralized* scheduling or a combination of them [9]. We note that similar heterogeneity and decentralization complexities exist in human economies where market driven economic models have been used to successfully manage them. Therefore, in [9] and [10], we proposed and explored the use economics as a metaphor for management of resources in Grid computing environments.

We have developed a Grid resource broker called Nimrod-G that performs scheduling of parameter sweep, task-farming applications on geographically distributed resources. It supports deadline and budget based scheduling driven by market-based economic models. To meet users' quality of service requirements, our broker dynamically leases Grid resources and services at runtime depending on their capability, cost, and availability. We have conducted many scheduling experiments that performed the execution of data-intensive, science applications such as molecular mo deling for drug design under a few Grid scenarios (like 2 hours deadline and 10 machines for a single user). Our ability to experiment with a large number of Grid scenarios was limited by the number of resources that were available to us in the WWG (world wide grid) test test-bed [20]. Also, it was impossible to create *repeatable* and *controlled* environment for experimentation and evaluation of scheduling strategies. This is because resources in Grid span across multiple administrative domains, each with their own policies, users, and priorities.

The researchers and students investigating resource management and scheduling for large scale distributed computing need a simple framework for deterministic modeling and simulation of resources and applications to evaluate of scheduling strategies. For most who do not have access to ready-to-use testbed infrastructure, building them is expensive and time consuming. Also, even for those who have access, the testbeds contain a few resources, testing ideas for scalability and evaluating scheduler performance for various application and resource scenarios is harder and impossible to trace. To overcome these limitations, we have proposed and developed a Java-based Grid Simulation toolkit called GridSim. The Grid computing researchers and educators also recognized the importance and the need for such a toolkit for modeling and simulation environments [13]. It should be noted that this paper has much orientation towards Grid, however, we believe that our discussion and thoughts apply equally well to P2P systems since resource management and scheduling issues in both systems are quite similar.

The GridSim toolkit provides facilities for the modeling and simulation of resources and network connectivity with different capabilities, configurations, and domains. It provides information services, application composition through the creation of *Gridlets* that contain job program properties, processing power requirements, the size of input and output files; and necessary interfaces for mapping Gridlets to resources along with the support for their management and instrumentation. These features can be used to develop resource brokers or Grid schedulers that help in the performance evaluation of scheduling algorithms. We have used GridSim toolkit features to simulate Nimrod-G resource broker [8] for design and evaluation of deadline and budget constrained scheduling algorithms with cost and time optimizations.

The GridSim toolkit supports modeling and simulation of a wide range of heterogeneous resources, such as single or multi processors, shared and distributed memory machines such as PCs, workstations, SMPs, and clusters with different capabilities and configurations. It can be used for modeling and simulation of application scheduling on various classes of parallel and distributed computing systems such as clusters [2], Grids [3], and peer-to-peer (P2P) networks [4]. The resources in clusters are located in a single administrative domain and managed by a single entity whereas, in grid and P2P systems, resources are geographically distributed across multiple administrative domains with their own management policies and goals. Another key difference between cluster and Grid/P2P systems arises from the way application scheduling is performed. The *schedulers* in cluster systems focus on enhancing overall system performance and utility, as they are responsible for the whole system. Whereas, schedulers in Grid/P2P systems called *resource brokers*, focus on enhancing performance of a specific application in such a way that its end-users requirements are meet.

The rest of this paper is organized as follows. Section 2 discusses related work with highlights on unique features that distinguish our toolkit from other packages. The GridSim architecture and internal components that make up GridSim simulations are discussed in Section 3. In Section 4, we discuss how to build GridSim based scheduling simulations. Sample results of simulation of a resource broker similar to Nimrod-G with deadline and budget constrained, cost minimization scheduling algorithm is discussed in Section 0. The final section summarizes the paper along with conclusions and our thoughts for future work.

2 Related Work

Simulation has been used extensively for modeling and evaluation of real world systems, from business process and factory assembly line to computer systems design. Accordingly over the years, modeling and simulation has emerged as an important discipline and many standard and application-specific tools and technologies have been built. They include simulation languages (e.g., Simscript [14]), simulation environements (e.g., Parsec [15]), simulation libraries (SimJava [1]), and application specific simulators (e.g., OMNet++ network simulator [19]). While there exists a large body of knowledge and tools, there are very few tools available for application scheduling simulation in Grid computing environments. The notable ones are: Bricks [16], MicroGrid [18], Simgrid [17], and our GridSim toolkit.

The Bricks simulation system [16] developed at the Tokyo Institute of Technology in Japan helps in simulating client-server like Global computing systems that provide remote access to scientific libraries and packages running on high performance computers. Similar to our work, it is multi-user, but focuses on overall system performance and service rates. The schedulers follow centralized global scheduling methodology as opposed to our work in which each application scheduling is managed by general purpose, private own resource broker.

The MicroGrid emulator [18] in development at the University of California at San Diego (UCSD) is modeled after Globus [12]. It allows execution of applications constructed using Globus toolkit in a controlled virtual Grid emulated environment. The results produced by emulation can be precise, but modeling numerous applications, grid environments, and scheduling scenarios for realistic statistical analysis of scheduling algorithms is time consuming as applications run on emulated resources. Also, scheduling algorithms designers generally work with application models instead of constructing actual applications. Therefore, MicroGrid's need for an application constructed using Globus imposes significant development overhead. However, when an actual system is implemented by incorporating scheduling strategies that are evaluated using simulation, the MicroGrid emulator can be used as a complimentary tool for verifying simulation results with real applications.

The Simgrid toolkit [17], developed in the University of California at San Diego (UCSD), is a C-language based toolkit for the simulation of application scheduling. It supports modeling of resources that are *time-shared* and the load can be injected as constants or from real traces. It is a powerful system that allows creation of tasks in term of their execution time and resources with respect to standard machine capability. Using Simgrid APIs, tasks can be assigned to resources depending on scheduling policy being simulated. It has been used for a number of real studies, and demonstrates the power of simulation. However, because Simgrid is restricted to a single scheduling entity and time-shared systems, it is difficult to simulate multiple competing users, applications, and schedulers, each with their own policies when operating under market like grid computing environment, without extending the toolkit substantially. Also, many large-scale resources in the Grid environment are space-shared machines and they need to be support in simulation. Hence, our GridSim toolkit extends the ideas in existing systems and overcomes their limitations accordingly.

Finally, we have chosen to implement GridSim in Java by leveraging JavaSim's [1] basic discrete event simulation infrastructure. This feature is likely to appeal to educators and students since Java has emerged as one of a popular programming language for network computing.

3 GridSim: Grid Modeling and Simulation Toolkit

The GridSim toolkit provides a comprehensive facility for simulation different classes of heterogeneous resources, users, applications, and resource brokers. In Grid-like environment, resource brokers perform resource selection and aggregation depending on users requirements and hence they are *user-centric* in nature. Whereas, in single administrative domain PDC systems such as clusters, resource *schedulers* manage whole resource and hence they are generally *system-centric* in nature as they aim to optimize system throughput for enhancing utilization of the entire system. The GridSim toolkit can be used for simulating application schedulers for different parallel and distributed computing systems such as clusters and grids.

3.1 Features

Salient features of the GridSim toolkit include the following:

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space- or time-shared mode.
- Resource capability can be defined (in the form of MIPS as per SPEC benchmark).
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on resource's local time to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU and/or I/O intensive.
- It does not limit number of application tasks that can be submitted to a resource.

- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

3.2 Design and Architecture

GridSim is built on a general-purpose discrete event simulation package called SimJava [1], which is implemented in Java. A layered design and modular architecture of the GridSim toolkit is shown in Figure 2. GridSim's highly specialized grid simulation base classes are developed extending SimJava's "simulation foundation classes" that are useful for constructing and animating general-purpose discrete event simulation models. Rather than building graphically, models in SimJava are built using the Java programming language. SimJava is, therefore, highly suitable for modeling computer architectures and software that usually require the full flexibility of a programming language, for expressing behaviors of components, rather than some queuing models. SimJava, can simulate a network of active entities that communicate by sending and receiving passive event objects efficiently.

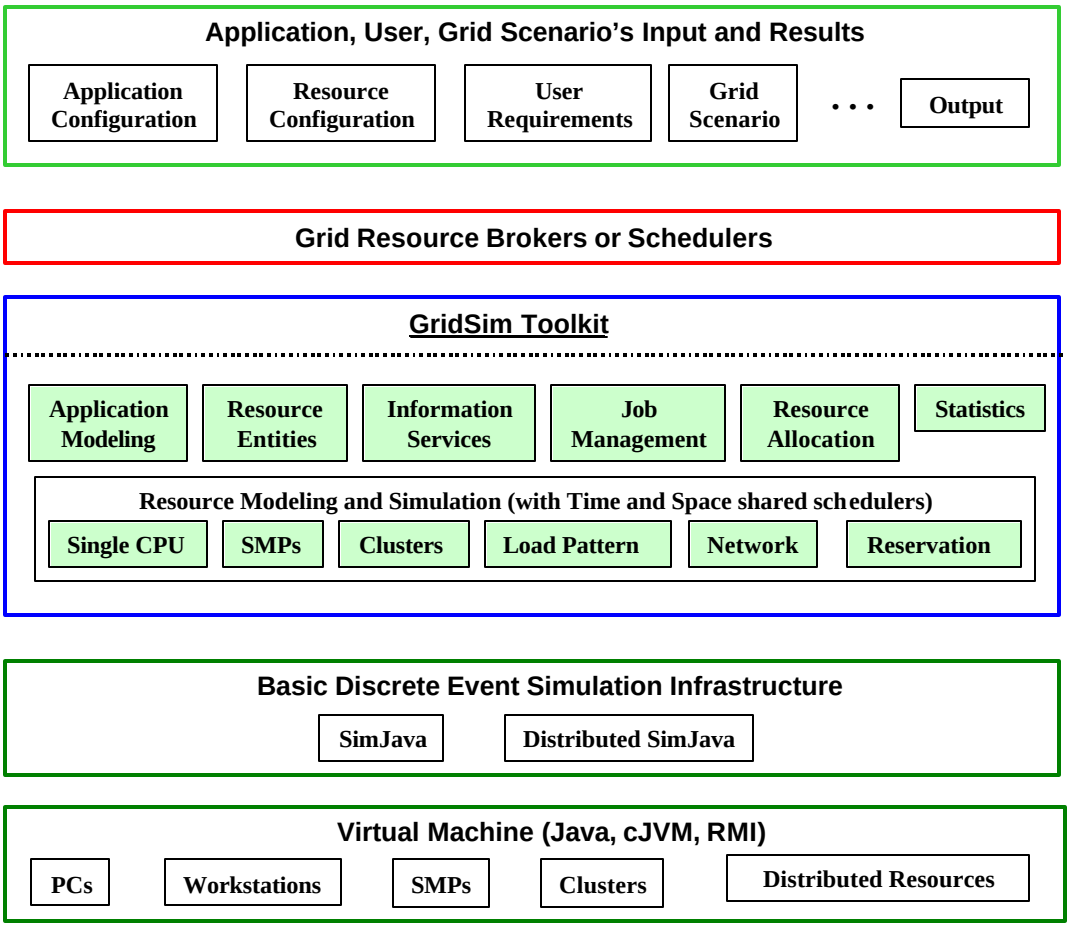

Figure 2: A modular architecture for GridSim platform and components.

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous

resources that can be configured as time or space shared systems. It allows setting their clock to different time zones to simulate geographic distribution of resources. It supports entities that simulate network used for communication among resources. During simulation, GridSim creates a number of multi-threaded entities, each of which runs in parallel in its own thread. An entity's behavior is encoded in Java using its body() method. Entities have access to a small number of simulation primitives provided by SimJava:

- sim_schedule() sends event objects to other entities;
- sim_hold() holds for some simulation time;
- sim_wait() waits for an event object to arrive; etc.

The basic sequential discrete event simulation algorithm, in SimJava, is as follows. A central object Sim_system maintains a timestamp ordered queue of future events. Initially all entities are created and their body() methods are put in run state. When an entity calls a simulation function, the Sim_system object halts that entity's thread and places an event on the future queue to signify processing the function. When all entities have halted, Sim_system pops the next event off the queue, advances the simulation time accordingly, and restarts entities as appropriate. This continues until no more events are generated. If the Java virtual machine supports native threads, then all entities starting at exactly the same simulation time may run in parallel.

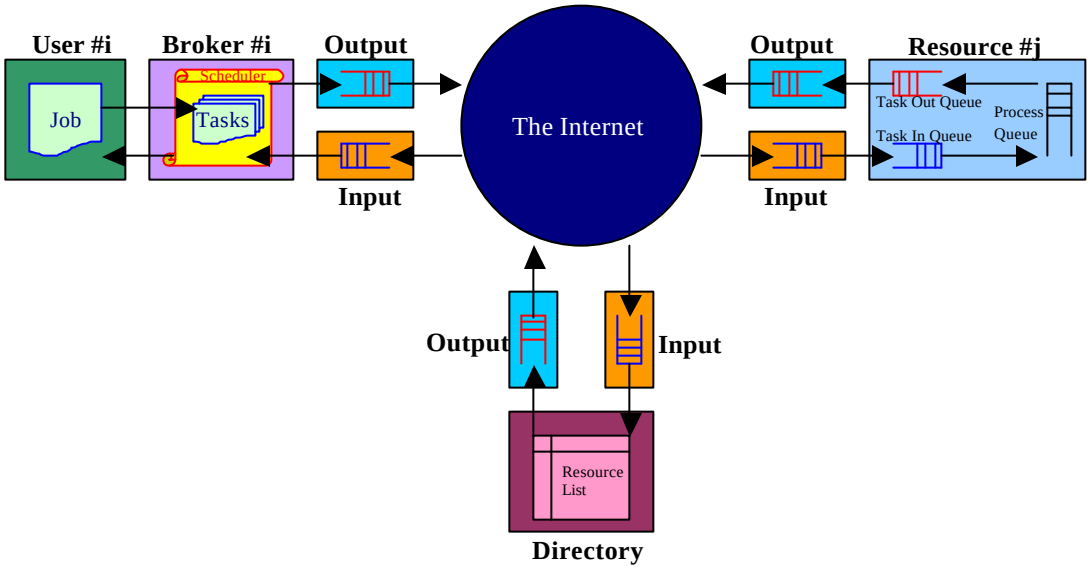

Figure 3: A flow diagram in GridSim based simulations.

Generally a GridSim based simulation contains entities for the users, brokers, resources, grid information service, statistics, and network based I/O as shown in Figure 3. The design and implementation issues of these GridSim entities are discussed below:

User – Each instance of the User entity represents a Grid user. Each user may differ from the rest of users with respect to the following characteristics:

- Types of job created e.g., job execution time, number of parametric replications etc.,
- Scheduling policy e.g., minimization of cost, time, or both,
- Activity rate e.g., how often it creates new job,
- Time zone, and
- D- and B-factors, measured in the range [0,1], express deadline and budget affordability of the user.

A D- factor close to 1 signifies that the user is willing to set the deadline of a job as long as required even when only the few slowest resources are available for the job. Similarly a B-factor close to 1 signifies that the user is capable of spending as much money as required even when only the few expensive resources are available for the job. These factors are basically useful for determining deadline and budget values for a given scenario at runtime. Instead, users can also explicitly specify values for deadline and budget constraints similar to the way it is currently done in Nimrod-G broker.

A user dispatches jobs to the Grid through its own Broker.

Broker – Each user is connected to an instance of the Broker entity. Every job of a user is first submitted to its broker and the broker then schedules the parametric tasks according to the user's scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global Directory entity. Every broker tries to optimize the policy of its user and therefore, brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market's demand-supply situation.

Resource – Each instance of the Resource entity represents a Grid resource. Each resource may differ from the rest of resources with respect to the following characteristics:

- Number of processors;
- Cost of processing;
- Speed of processing;
- Internal process scheduling policy e.g., time shared or space shared;
- Local load factor; and
- Time zone.

Cost and speed of processing a task by a resource and the estimated execution time of a task from a user are all measured with respect to a standard **unit** machine. Upon obtaining a resource index from the Grid information service, brokers can query resources directly for their static and dynamic properties.

Grid Information Service – It provides resource registration services and keeps track of a list of resources available in the Grid. The brokers can query this for resource handle and contact information.

Input and Output – Information flows among the brokers, the active resources, and the Trader through the network of networks known as the Internet. To model networking though I/O channels, the *networked entities*, i.e., the Broker, the Resource, and the Directory entities, are connected to the Sim_System through their own separate Input and Output entities. Using separate entities for input and output enables a networked entity to model full duplex communications, if necessary. More importantly, using I/O entities not only makes task execution of each networked entity completely independent of I/O processing, as required, but also keeps the design of new economic models and scheduling algorithms simple.

3.3 GridSim Java Package Implementation

A class diagram of the gridsim package is shown in Figure 4. The gridsim package implements the following base classes:

`class gridsim.Input` – This class extends the eduni.simjava.Sim_entity class. This class basically defines a port through which a simulation entity receives data from the simulated network. It maintains an event queue to serialize the data-in-flow. Simultaneous input, in parallel, can be modeled simply by using multiple instances of this class.

`class gridsim.Output` – This class is very similar to the gridsim.Input class and it defines a port through which a simulation entity sends data to the simulated network. It maintains an event queue to serialize the data-out-flow. Simultaneous output, in parallel, can be modeled by using multiple instances of this class.

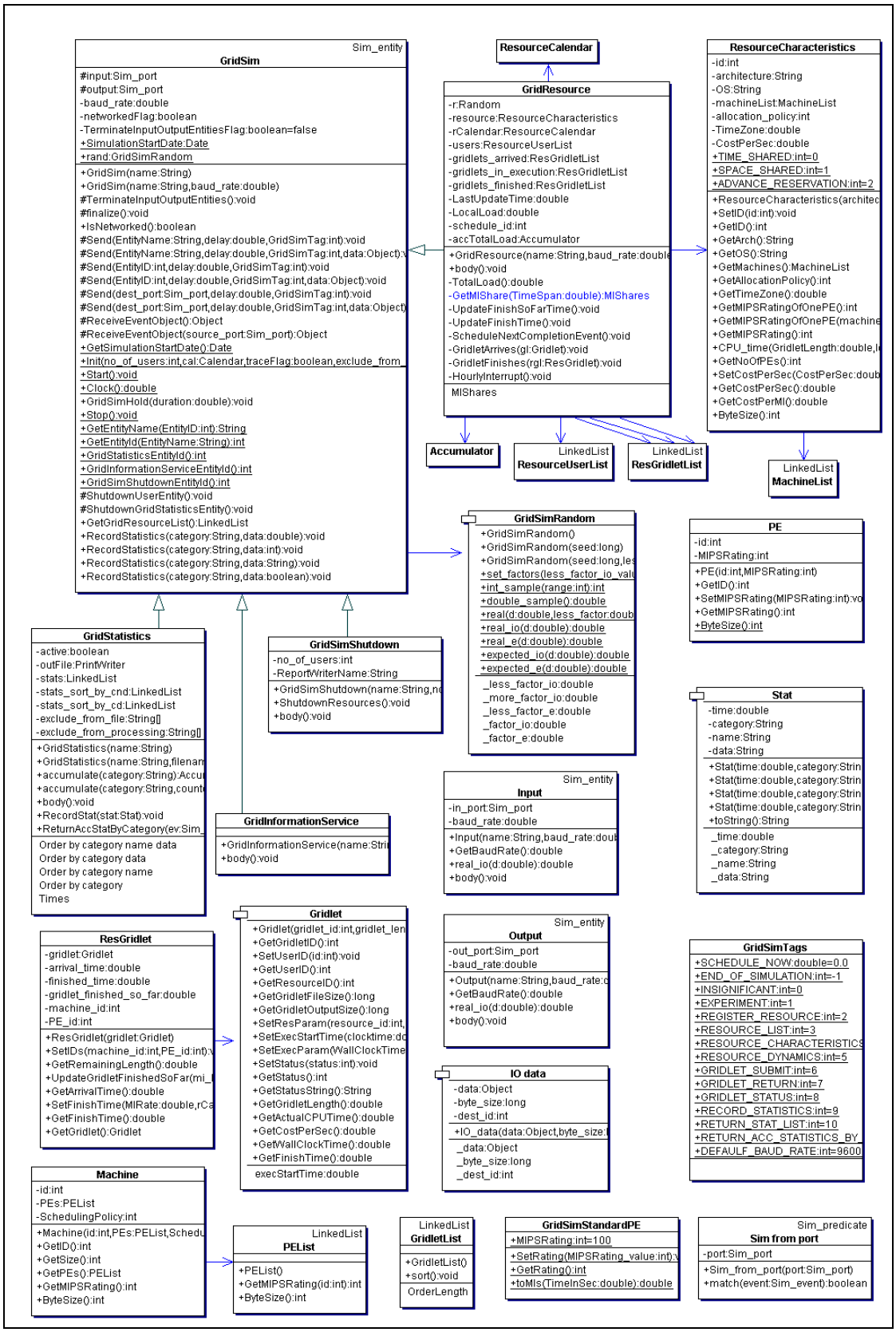

Figure 4: Class diagram of gridsim package.

`class gridsim.GridSim` – This is the main class of gridsim package that must be extended by GridSim entities. It inherits events management features from the eduni.simjava.Sim_entity class. Extending this class creates new GridSim entities that can be instantiated with or without networked I/O ports. A networked GridSim entity with name (say) "Resource10" creates one gridsim.Input object with name "Input_Resource10" and one gridsim.Output object with name "Output_Resource10."

All classes that extend the GridSim class must implement a method called `body()`, which is automatically invoked for processing events. This class implements two important static methods—Init () and Start (). The GridSim.Init() must be invoked before creating any GridSim entities. This method prepares the system by creating three important GridSim entities—GridInformationService, GridSimShutdown, and GridStatistics. As explained in Section 3.2, the GridInformationService entity simulates the Directory that dynamically keeps a list of all the resources available in the Grid. The GridSimShutdown entity helps in wrapping up a simulation by systematically closing all the opened GridSim entities. The GridStatistics entity provides standard services during the simulation to accumulate statistical data. Invoking the GridSim.Start () method starts the Grid simulation. All the resource and user entities must be instantiated in between invoking the above two methods.

The GridSim class supports static methods for sending and receiving messages between entities directly or via network entities, managing and accessing handle to various GridSim core entities, and recording statistics.

`class gridsim.PE` – An instance of this class simulates a single CPU (also called *Processing Elements* (PE)). The PE capability is defined using MIPS rating.

`class gridsim.PEList` – An instance of this class simulates a collection of CPUs.

`class gridsim.Machine` – An instance of this class simulates a machine with one or more CPUs. A Machine object can model both uniprocessor and multiprocessor, e.g., SMP, single-unit machine.

`class gridsim.MachineList` – An instance of this class simulates a collection of machines. It is up to the GridSim users to define the connectivity among the machines in a collection. Therefore, this class can be instantiated to model simple LAN to Cluster to WAN.

`class gridsim.ResourceCharacteristics` – An instance of this class represents all static properties of a Resource such as resource architecture, OS, management policy (time or space shared), cost, and time zone at which resource is located.

`class gridsim.GridResource` – An instance of this class represents a simulated resource with its properties indicated by an instance of `gridsim.ResourceCharacteristics` class. The process of creating a Grid resource is as follows: first create PE objects with a suitable MIPS rating, second assemble them together to create a Machine and finally group multiple Machine objects together to form a resource. A resource having a single machine with one or more PEs is managed as time-shared system using round robin scheduling algorithm. A resource with multiple machines is treated a distributed memory cluster and is managed as space-shared system using first-come first served scheduling policy or its variants.

`class gridsim.ResourceCalendar` – This class implements a mechanism to support modeling non-Grid (local) load on Grid resource entities that may vary according to time zone, time, weekends, and holidays.

`class gridsim.GridInformationService` -- An instance of this class maintains a pointer to a list of resources that are ready to accept Grid jobs. This class basic provides grid resource index services. The resource pointer/index can be used by scheduling entities for interacting with resources either for resource static or dynamic information and submitting Gridlets for execution by passing appropriate messages.

`class gridsim.Gridlet` – An instance of this class represents one Grid-enabled task. Individual users create tasks that are processed by Grid resources through resource brokers that implement various scheduling policies.

`class gridsim.GridletList` – An instance of this class represents a collection of similar tasks.

`class gridsim.Experiment` – An instance of this class represents an experiment that consists of a GridletList and an optimization policy. On receiving an experiment form its user, a resource broker starts scheduling individual Gridlets according to the optimization policy set for the experiment.

`class gridsim.GridSimTags` – It contains various static command tags that indicate a type of action that needs to be undertaken by GridSim entities when they receive events. The different types of tags supported in GridSim along with comments indicating possible purpose is shown in code sample below:

```
public class GridSimTags {
  public static final double SCHEDULE_NOW = 0.0;  // 0.0 indicates NO delay
  public static final int END_OF_SIMULATION = -1;
  public static final int INSIGNIFICANT = 0; // ignore tag
  public static final int EXPERIMENT = 1; // User <-> Broker
  public static final int REGISTER_RESOURCE = 2; // GIS -> ResourceEntity
  public static final int RESOURCE_LIST = 3; // GIS <-> Broker
  public static final int RESOURCE_CHARACTERISTICS = 4; // Broker <-> ResourceEntity
  public static final int RESOURCE_DYNAMICS = 5; // Broker <-> ResourceEntity
  public static final int GRIDLET_SUBMIT = 6; // Broker -> ResourceEntity
  public static final int GRIDLET_RETURN = 7; // Broker <- ResourceEntity
  public static final int GRIDLET_STATUS = 8; // Broker <-> ResourceEntity
  public static final int RECORD_STATISTICS = 9; // Entity -> GridStatistics
  public static final int RETURN_STAT_LIST = 10; // Entity <- GridStatistics
  public static final int RETURN_ACC_STATISTICS_BY_CATEGORY = 11;
  public static final int DEFAULF_BAUD_RATE = 9600; // Default Baud Rate for entities
}
```

4 Building Simulations with GridSim

The Java-based GridSim discrete event simulation toolkit provides Java classes that represent entities essential for application, resource modeling, scheduling of jobs to resources, and their execution along with management. A schematic representation of interaction between GridSim entities during simulator execution is shown in Figure 3. The process of resource and application modeling for developing Grid schedulers/brokers is discussed in this section.

4.1 Resource Modeling

In the GridSim toolkit, we can create Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Then, one or more PEs can be put together to create a machine (a single CPU/SMP node). Similarly, one or more machines can be put together to create a Grid resource. Thus, the resulting Grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These Grid resources can be managed by time-shared or space shared scheduling algorithms depending on type of the resource. Generally, a single PE or SMP type Grid resource is managed by time-shared operating systems using round-robin scheduling policy and a cluster-like Grid resource is managed by space-shared Q-schedulers using different scheduling policies such as first-come-first-served (FIFO), back filling, shortest-job-first (SJF), and so on.

For every Grid resource, the non-Grid (local) workload is estimated based on typically observed load conditions as well as the time zone of the resource. The network communication speed between a user and the resources is defined in terms of a data transfer speed (baud rate). When a resource entity is created, it registers resource information and contact details with the Grid Information Service (GIS) entity. This resource registration process is similar to GRIS (Grid Resource Information Server) registering with GIIS (Grid Index Information Server) in Globus system. The GIS can then be queried for list of resource in a given Grid domain to get resource handles, which can be used to query resources directly for their capabilities, costs, and other configurations.

4.2 Application Modeling

GridSim does not explicitly define any specific application model. It is up to the developers (of schedulers and resource brokers) to define them. We have experimented with task-farming application model and we believe that other parallel application models such as process parallelism, DAGs (Directed Acyclic

Graphs), divide and conquer, etc., described in [5], can also be modeled and simulated using GridSim.

In GridSim, each independent task can require varying processing time and input files size. Such tasks can be created and their requirements are defined through *Gridlet* objects. A Gridlet is a task that contains all the information related to jobs and job execution management details such as job length expressed in MIPS, disk I/O operations, the size of input and out files. These basic parameters make it possible to compute the execution time on the remote resource and the time required to transport input and output files between users and remote resources. The GridSim toolkit supports a wide range of Gridlet management protocols and services that allow schedulers to map a Gridlet to a resource and manage it through out its life cycle.

4.3 Steps for Simulating Application Scheduling

In this section we present high-level steps, with sample code clips, to demonstrate how GridSim can be used to simulate a Grid environment to analyze some application scheduling algorithms:

- First, we need to create Grid resources of different capability/speed like those in the real environment at different time zones and different policies (like time- or space-shared resources in the World-Wide Grid (WWG) testbed [11]). We also need to create users with different requirements. A sample code, serving the above purposes, is given in Figure 5.

```
public static void CreateSampleGridEnvironement(int no_of_users, int no_of_resources,
  double B_factor, double D_factor, int policy, double how_long, double seed) {
  Calendar now = Calendar.getInstance();

  GridSim.Init(no_of_users, calender, true, eff, efp, ReportWriterName);

  // Create Resources
  for(int i=0; i<no_of_resources; i++) {
     // Create PEs
     PEList peList = new PEList();
     for(int j=0; j<(i*1+1); j++)
        peList.add(new PE(0, 100));

     // Create machine list
     MachineList mList = new MachineList();
     mList.add(new Machine(0, peList, ResourceCharacteristics.TIME_SHARED));

     // Create a resource containing machines
     ResourceCharacteristics resource = new ResourceCharacteristics("INTEL", "Linux",
         mList, ResourceCharacteristics.TIME_SHARED, 0.0, i*0.5+1.0);
     LinkedList Weekends = new LinkedList();
     Weekends.add(new Integer(Calendar.SATURDAY));
     Weekends.add(new Integer(Calendar.SUNDAY));
     LinkedList Holidays = new LinkedList(); // no holiday is set!

     // Setup resource as simulated entity with a name (e.g. "Resource_1").
     new GridResource("Resource_"+i, 28000.0, seed, resource,
                                          0.0, 0.0, 0.0, Weekends, Holidays);
  }
  Random r = new Random(seed);
  // Create Application, Experiment, and Users
  for(int i=0; i<no_of_users; i++)
  {
    GridletList glList = Application1(r);
    Experiment expt = new Experiment(0, glList, policy, B_factor, D_factor);
    new UserEntity("User_"+i, expt, 28000.0, how_long, seed*997*(1+i)+1, 0);
  }
  // Perform Simulation
  GridSim.Start();
}
```

Figure 5: A sample code segment for creating Grid resource and user entities in GridSim.

- Second, we need to model applications by creating a number of Gridlets (that appear similar to Nimrod-G jobs) and define all parameters associated with jobs as shown in Figure 6. The Gridlets can be grouped together depending on application model for processing.

```
Gridlet gl = new Gridlet(user_id, Gridlet_id, Gridlet_length, GridletFileSize,
                    GridletOutputSize);
```

Figure 6: The Gridlet method in GridSim.

- Finally, we need to implement resource brokers as shown in Figure 7. First, inquire the Grid Information service (GIS), then inquire for resource capability including cost, and then depending on scheduling heuristics, strategy, or algorithms assign Gridlets to resources for execution. The scheduling policies can be systems-centric like those implemented in many Grid systems such as Condor-G or user-centric like the Nimrod-G broker quality of service (QoS) algorithms [10].

```
class Broker extends GridSim {
  private Experiment experiment;
  private LinkedList ResIDList;
  private LinkedList BrokerResourceList;

  public Broker(String name, double baud_rate)
  {
    super(name, baud_rate);
    GridletDispatched = 0;
    GridletReturned = 0;
    Expenses = 0.0;
    MaxGridletPerPE = 2;
  }

  ... // Gridlet management code

  public void body() {

    Sim_event ev = new Sim_event();
    // Accept User Commands and Process
    for( sim_get_next(ev); ev.get_tag()!=GridSimTags.END_OF_SIMULATION; sim_get_next(ev))
    {
      experiment = (Experiment) ev.get_data();
      int UserEntityID = ev.get_src();

      // RESOURCE DISCOVERY
      ResIDList = (LinkedList) GetGridResourceList();

      // RESOURCE TRADING and SORTING
      // SCHEDULING
      while (glFinishedList.size() < experiment.GetGridletList().size())
      {
        if((GridSim.Clock()>=experiment.GetDeadline())||(Expenses>=experiment.GetBudget()) )
          break;

        scheduled_count = ScheduleAdviser();
        dispatched_count = Dispatcher();
        received_count = Receiver();

        // Heurisitics for deciding hold condition
        if(dispatched<=0 && received<=0 && glUnfinishedList.size()>0)
        {
          double deadline_left = experiment.GetDeadline()-GridSim.Clock();
          GridSimHold(Math.max(deadline_left*0.01, 1.0));
        }
      }
    }
  ...  // Code for actual scheduling policy
  ...  // Code for dispatch policy
  }
```

Figure 7: A sample code segment for creating a Grid resource broker in GridSim.

Within such GridSim simulations, we can create multiple user entities with their own applications and requirements for scheduling on the same groups of resources. This can help in developing brokers like in those required in the real Grid. Having the ability to allow brokers to compete enables the design and evaluation of a market-based economic models and corresponding scheduling algorithms to create a Grid marketplace as discussed in [11].

5 Sample Simulation Results

We have used the GridSim toolkit to simulate a grid resource broker with the deadline and budget constrained *cost-optimisation* scheduling algorithm [10]. This algorithm attempts to complete an experiment of a large number of jobs as economically as possible within a user specified deadline. High-level steps of this adaptive scheduling algorithm are discussed below:

1. RESOURCE DISCOVERY: Identify resources that can be used in this execution with their capability through the Grid Information Service.
2. RESOURCE TRADING: Identify cost of each of the resources in terms of CPU cost per second and capability to be delivered per cost-unit.
3. SORT resources by increasing order of cost.
4. SCHEDULING: Repeat while there exists unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event

   [SCHEDULE ADVISOR with Policy]

   a. For each resource predict and establish job consumption rate through measure and extrapolation.
   b. For each resource based on its job consumption rate, predict and establish number of jobs a resource can consume by application deadline.
   c. For each resource in order
      i. If the number of jobs currently assigned to a resource is less than predicated number of jobs that a resource can consume, assign more jobs from unassigned job queue or from most expensive machines based on job state and feasibility.
      ii. Alternatively, if a resource has excess number of jobs that it can complete, move those extra jobs to unassigned job queue.

   [DISPATCHER with Policy]

   d. The dispatcher takes care of submission of jobs to remote machines with submission and resource policy and constraints depending on resource type (time or space shared).

We simulated a Grid with 21 independent users that are competing for 25 resources using a normal distribution from 10 units per second (G$/sec.) to 20 G$/sec. Each user application contains 20 jobs with variation of ±2 with random submission. Each job was modelled to a resolution of 50 time units on a standard machine. The resources capability rating varied with normal distribution from 0.5 to 1.5 (with 1 for standard machine). For the sake of simplicity in analysing the result, we assumed all the users and resources are stochastically similar within the respective groups with very small variances in their characteristics.

Some of the results obtained from the experiment are plotted in Figure 8. Job completion rate, time utilization, and budget utilization are plotted in Figure 6(a)-(c) respectively, for the maximum number of users and resources used in the experiment, against the B- and D-factors that are varied from 0.5 to 2.5. These figures show that both the job completion rate and the time utilization are linearly proportional to only the D-factor that determines the deadline. On the contrary, the budget utilization varies with both the B- and D-factors. For a fixed B-factor, the budget utilization increases linearly with the D-factor; but for a fixed D-factor, decreases with the B-factor linearly, when the D-factor is low, and logarithmically, when the D-factor is reasonably high.

In Figure 6(d)-(f), we plotted job completion rate, time utilization, and budget utilization, for fixed (110%) B- and D-factors, against the number of users and resources that are varied in the ranges 1 to 21 and 1 to 25 respectively. These figures show that the job completion rate decreases with the increase in the number of users due to the obvious reason of competition. But it is also observed that both the time and budget utilizations also decrease with the increase in the number of users. This observation is against the obvious expectation that utilization of time and budget should increase with the competition. However, this unexpected result can be explained because both the time and budget utilizations were calculated only for successful jobs and when the job utilization is low, successful jobs, in reality, could see relatively under utilized resources.

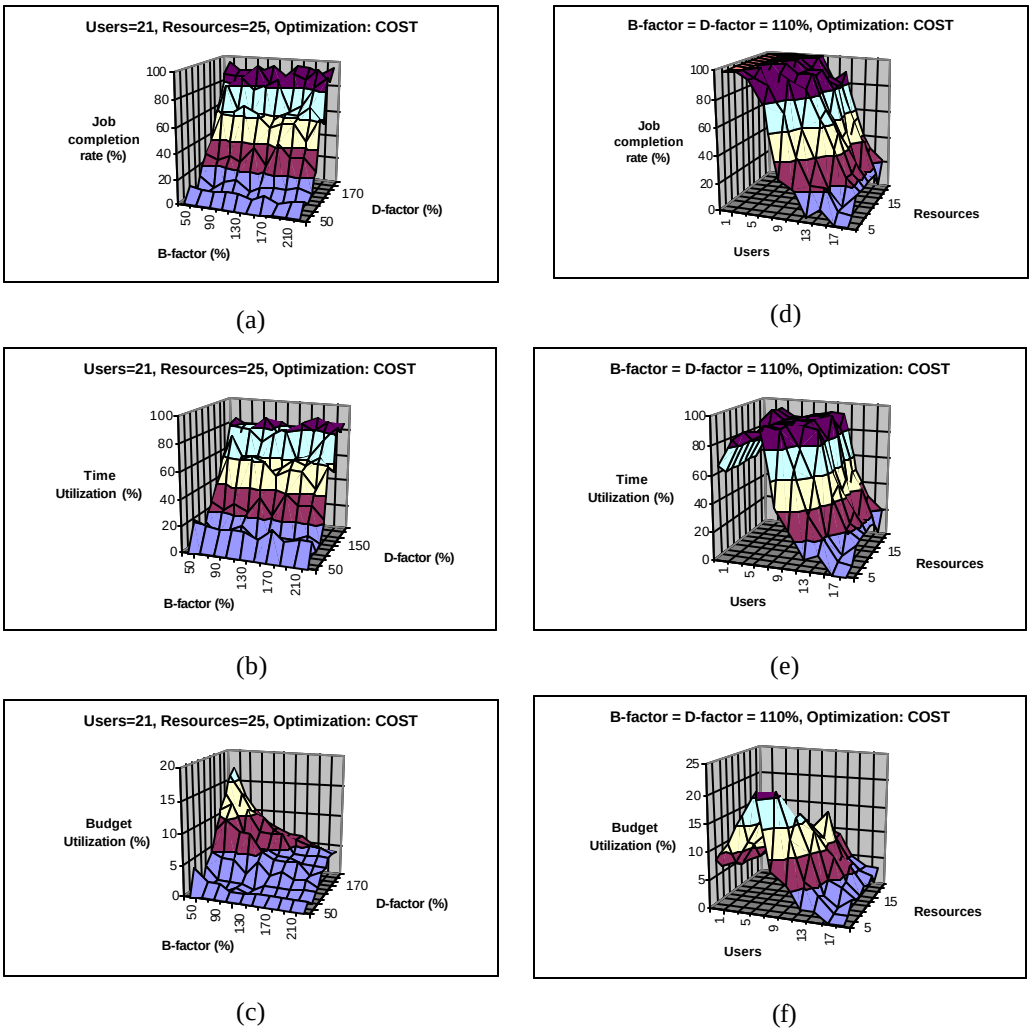

Figure 8: Job completion rate (a), time utilization (b), and budget utilization (c) of our COST_OPTIMIZATION scheduling algorithm simulated with a fixed number of users and resources while both the B- and D-factors are varied. Job completion rate (d), time utilization (e), and budget utilization (f) of the same algorithm simulated with fixed B- and D-factors while both the number of users and resources are varied.

6 Conclusion and Future Work

In this paper we have discussed an object oriented Grid simulation toolkit called GridSim. GridSim allows the simulation of time- and space-shared resources with different capabilities, time zones, and configurations. It supports different application models that can be mapped to resources for execution by developing simulated application schedulers. We have discussed the architecture and components of the GridSim toolkit along with steps involved in creating GridSim based application-scheduling simulators.

We have simulated a Nimrod-G like Grid resource broker using GridSim and evaluated a number of scheduling algorithms based on deadline and budget based constraints. This gave us ability to test our application scheduling policies with different Grid configurations such as varying number of resources, users, and requirements. The results are promising and hence we believe that scheduling researchers can use GridSim toolkit for evaluating scheduling algorithms through simulation with a suitable application model.

We believe that our implementation of GridSim toolkit in Java is an important move since Java-based applications are portable across many platforms. Java is a popular programming language for network computing and many easy-to-use and friendly development environments are available. Notably, the JVM (Java Virtual Machine) is available for single, multiprocessor shared or distributed machines such as clusters, which means our system is scalable and large-scale simulations can be performed. We have leveraged the JavaSim toolkit in producing GridSim, and thus it was not necessary to build the discrete event simulation mechanism itself..

Future work will focus on strengthening the network model by supporting various types of networks with different static and dynamic configurations and cost-based quality of services. The resource models also need to be enhanced by interfacing with off-the-shelf storage I/O simulators. In order to enable simulation of schedulers with economic models such as tenders and auctions, we plan to incorporate the FIPA (Foundation for Intelligent Physical Agents) agents standards [21] based interaction protocol infrastructure. To enable creation of resources with varying quality of service metrics, we are currently working on developing an economics driven scheduler for computational clusters. This scheduler can become the foundation for enhancing resources with quality of service guarantees.

Software Availability

The GridSim toolkit software with source code is available for download:

`http://www.csse.monash.edu.au/~rajkumar/gridsim/`

Acknowledgements

We thank John Newsome Crossley, Monash University and Rob Gray, Distributed System Technology Centre (DSTC) for their critical comments and suggestions that helped in improving the quality of our paper. We thank Ajith Abraham and his family for their hospitality during Rajkumar's visit to Gippsland. We thank our family members for allowing us to work even on weekends!

References

[1] F. Howell and R. McNab, *SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling*, First International Conference on Web-based Modelling and Simulation, San Diego, CA, Society for Computer Simulation, Jan 1998.

[2] R. Buyya (editor), *High Performance Cluster Computing: Architectures and Systems*, Volume 1, Prentice Hall - PTR, USA, 1999.

[3] I. Foster and C. Kesselman (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[4] Andy Oram (editor), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly Press, USA, 2001.

[5] L. Silva and R. Buyya, *Parallel Programming Paradigms*, Chapter 2 in High Performance Cluster Computing: Programming and Applications (Vol. 2), Prentice Hall, NJ, USA, 1998.

[6] M. Baker, R. Buyya, D. Laforenza, *The Grid: International Efforts in Global Computing*, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, Italy, 2000.

[7] D. Abramson, J. Giddy, and L. Kotler, *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* International Parallel and Distributed Processing Symposium (IPDPS), IEEE Computer Society Press, 2000.

[8] R. Buyya, D. Abramson and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, 4th Intl. Conf. on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), China.

[9] R. Buyya, D. Abramson and J. Giddy, *Economy Driven Resource Management Architecture for Computational Power Grids*, Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), USA.

[10] R. Buyya, J. Giddy, D. Abramson, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications*, The Second Workshop on Active Middleware Services (AMS 2000), In conjunction with HPDC 2001, August 1, 2000, Pittsburgh, USA (Kluwer Academic Press).

[11] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson, *Economic Models for Management of Resources in Peer-to-Peer and Grid Computing*, SPIE International Conference on Commercial Applications for High-Performance Computing, August 20-24, 2001, Denver, USA.

[12] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11(2): 115-128, 1997.

[13] J. Weissman, Grids in the Classroom, *IEEE Distributed Systems Online*, Volume 1, No. 3, 2000, http://www.computer.org/dsonline/archives/ds300/ds3eduprint.htm

[14] CACI, *Simscript: a simulation language for building large-scale, complex simulation models*, CACI Products Company, San Diego, CA, USA, http://www.simscript.com/simscript.cfm

[15] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, H. Song, *Parsec: A Parallel Simulation Environment for Complex Systems*, Vol. 31(10), IEEE Computer, October 1998.

[16] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima, *Performance evaluation model for scheduling in a global computing system*, The International Journal of High Performance Computing Applications, vol. 14, No. 3, 2000.

[17] H. Casanova, *Simgrid: A Toolkit for the Simulation of Application Scheduling*, Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia.

[18] H. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura and A. Chien, *The MicroGrid: a Scientific Tool for Modeling Computational Grids*, Proceedings of IEEE Supercomputing (SC 2000), Nov. 4-10, 2000, Dallas, USA.

[19] A. Varga, *The OMNeT++ Discrete Event Simulation System*, Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic.

[20] R. Buyya, The World Wide Grid, http://www.csse.monash.edu.au/~rajkumar/ecogrid/wwg/

[21] Foundation for Intelligent Physical Agents (FIPA), *Interaction and Negotiation Protocols*, http://www.fipa.org/