FACOLTA' DI SCIENZE MM.FF.NN.
UNIVERSITA' DEGLI STUDI DELL'INSUBRIA - VARESE

Thesis Degree

Assessment of dependability scenarios in large-scale grids, using GridSim toolkit

Stefano Barboni

Supervisor:	Prof. Dr. Alberto Trombetta
Co-examiners:	Prof. Dr. Dieter Kranzlmüller
	Dr. Michael Schiffers
Deadline:	30. March 2011

Thesis project developed at:

INSTITUT FÜR INFORMATIK

LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Thesis Degree

Assessment of dependability scenarios in large-scale grids, using GridSim toolkit

Stefano Barboni

I assure, that I made this Master's Thesis in complete autonomy and that my only help has been the indicated references.

Assicuro, che ho fatto questa Tesi Specialistica in maniera autonoma e ho utilizzato come aiuto solo le fonti indicate.

Varese, 30 March 2011

. . . . . .
(Signature of candidate)

Acknowledgements

To my family, especially my parents Sergio and Mariella Barboni, I gratefully dedicate this thesis. Thank you for always supporting me. I think it was worth it.

Alla mia famiglia, specialmente ai miei genitori Sergio e Mariella Barboni, io vi dedico con gratitudine questa tesi. Grazie per avermi sempre supportato. Penso ne sia valsa la pena.

I would like to thank the Institute of Computer Science of the Ludwig Maximilians Universität of Munich, for giving me the opportunity to develop this thesis at their institution.
I would also like to thank Dr. Michael Schiffers and Prof. Dr. Dieter Kranzmüller for all their support. It was absolutely invaluable.
More thanks to Prof. Dr. Alberto Trombetta, he has supported my work.
A special thanks goes to my team-mate of this thesis, Andrea Castiglioni, in the end I think it was a great experience.
I also thank for all the material, Prof. Dr. Rajkumar Buyya and all the contributors who have enabled the creation and development of the GridSim toolkit.
Many thanks also to Jaime Kelleher, with her precious help I could write a better thesis.
Above all, many thanks to all my friends for all the incredible times we had in recent years. They are really too many to mention, but I want to keep in mind that for me you were really importan: from the "Nucleo Team", to the University fellow students (and not only) and, during the last six unforgettable months, the friends met in Erasmus.

Abstract

Grid computing refers to coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations (VOs). As grid users (i.e., the members of the virtual organizations) organizations) are basing their work more and more on grid technology, grid systems must exhibit a high degree of dependability, i.e., they must be able to deliver service that can justifiably be trusted.
Although there have been several research efforts established to address dependability issues in distributed systems, most of the characteristics inherent to grids are considered only recently. One such characteristic is the large-scaleness of grids which presents a major challenge in understanding dependability.
In this thesis we simulate large-scale grids in order to enable a systematic study of their inherent dynamics (VOs, applications, middleware, resources, and networks).
The goal is to obtain the scenarios where we can get the description of the behavior of its components on a large scale.
In this thesis we propose the use of modeling and simulation, because various grid scenarios need to be evaluated and repeated. Hence, this thesis describes the development of GridSim, a discrete-event grid simulation tool, which allows modeling and simulation of various properties. The simulation is based on the GridSim tools provided by the University of Melbourne.
The tool provide a virtual grid infrastructure that enables experimentation with dynamic resource management techniques and adaptive services by supporting controllable, repeatable, observable experiments.

Sommario

Il Grid computing è una tecnologia che si riferisce alla condivisione di risorse coordinate e al problem solving in organizzazioni multi-istituzionale virtuali e dinamiche. Poichè gli utenti della rete (ovvero, i membri delle organizzazioni virtuali) basano sempre più il loro lavoro sulla tecnologia grid, i sistemi di grid devono presentare un grado elevato di affidabilità, ovvero, devono essere in grado di fornire un servizio che può essere giustamente considerato attendibile.
Nonostante siano state stabilite diverse attività di ricerca per affrontare i problemi di affidabilità nei sistemi distribuiti, la maggior parte delle caratteristiche inerenti al grid computing sono state prese in considerazione solo di recente. Una di queste caratteristiche è la grande scalabilità delle reti grid, che presenta una sfida importante nella comprensione dell'affidabilità.
In questo progetto si simula grid su larga scala al fine di consentire uno studio sistematico delle loro dinamiche intrinseche (VO, applicazioni, middleware, risorse e reti). L'obiettivo è di ottenere uno scenario dal quale possiamo ottenere la descrizione del comportamento dei suoi componenti su larga scala.
In questa tesi si propone l'utilizzo della simulazione, in quanto i vari scenari di rete devono essere valutati e ripetuti.
Quindi, questa tesi descrive lo sviluppo di GridSim, uno strumento di simulazione di un sistema grid ad eventi discreti, che permette la modellazione e la simulazione di varie proprietà. La simulazione si basa sullo strumento GridSim messo a disposizione dall'Università di Melbourne.
Esso fornisce una infrastruttura virtuale di Grid che permette la sperimentazione di tecniche di gestione dinamica delle risorse e di servizi adattabili sostenendo esperimenti controllabili, ripetibili e osservabili.

Contents

1 Introduction

1.1 What is the goal that we want to reach?

In this thesis we study the behavior of the grid using large scale scenarios.
It's impossible to actually test on a large network, so we use GridSim toolkits provided by the University of Melbourne to simulate it (please refer to Chapter 2.2).
We will discuss in detail the GridSim toolkit in Chapter 3.
Originally, the GridSim toolkit was designed to develop large-scale simulations using some random functions.
These random functions allow the developer to develop simulations without having to set up a series of parameters (that we will explain in detail later) such as:

- link the newly created regional Grid Information Service (GIS) with any router on the network at random;
- link the newly created resource with any regional GIS in the network;
- link the newly created user with any router on the network at random;
- link the newly created user with any regional GIS in the network;
- set the initial time when a gridlet is submitted to a resource;
- set the gridlet ID to a *gridelt_submit* event;
- set in which resource to send one or more gridlets;
- set the number of users per resource.

Our goal is to make these features adaptable, in this way we can choose how to implement the simulation network by manipulating all of its components.
In fact, to construct a scenario that is as close as possible to reality, we have to know with certainty where and when certain events will occur before the simulation begins.
In this way we can have a virtual grid infrastructure that enables experimentation with dynamic resource management techniques and adaptive services by supporting controllable, observable and, especially, repetable experiments.
With this, we will be able to build a series of scenarios on a large scale, and all data will be included at our discretion.
The ultimate goal is to analyze the results of these experiments in several aspects, by changing the settings of some parameters of the the network topology in order to obtain an overview of the behavior of software in different ways.
As we have said, with changes to the toolkit, we can get any kind of scenario we want.
But to obtain a clear view of GridSim, we concentrate our efforts in two different scenarios.

The first one is based on the EU DataGrid Testbed 1[1], and the second one is a large scenario that represents a typical network topology.
The choice of the first scenario comes from an experiment carried out by researchers at the University of Merlbourne in which they tested the proper operation of the toolkit with an extension of GridSim[2].
The second scenario is based on a network topology of our invention and wider than its predecessor.
This is because the first scenario must serve as a control model through which to obtain reliable results from a second scenario, that reflects a hypothetical future structure of a network at a large scale.

1.2 How do we reach the goal (methodology)?

In recent years, there have been many examples in which the software GridSim was used for modeling and simulating many interesting systems and ideas.
For example, IBM Research uses DataGrid package to simulate a grid meta-scheduler that tightly integrates the compute and data transfer times of each job[3].
Another example is Universidad de Santiago de Compostela's extension of GridSim to optimize execution of parallel applications on a grid[3].

In our project we simulate several tests in which we evaluate the result of grid computing systems.
To simulate grid resources using the GridSim toolkit, we need to create new entities that exhibit the behavior of grid users and scheduling systems.
GridSim base classes are extended by user-defined entities to inherit the properties of concurrent entities capable of communicating with other entities using events.
The detailed steps involved in modeling resources and applications, and simulating using the GridSim toolkit are discussed below.

1.2.1 A general view to understand the application scheduling

In this section we present high-level steps, to demonstrate how GridSim can be used and modified to simulate a grid environment to analyze scheduling algorithms.
As shown in Figure 1.1, the high-level steps can be summarized in five points.

In the first step we declare the most important entities of GridSim.
Indeed we have to set the number of the users that join the network, the number of the gridlets that each user has and, finally, we have to specify in how many regional GIS our network is shared.

The second step is important because we build the network topology among the routers.
After obtaining the network topology, GridSim reads it and initializes the simulation depending on the number of routers present.

The third, fourth and fifth steps are similar.
All three are creating and setting up an entity (GIS, resource or user) and then connect it to a router.

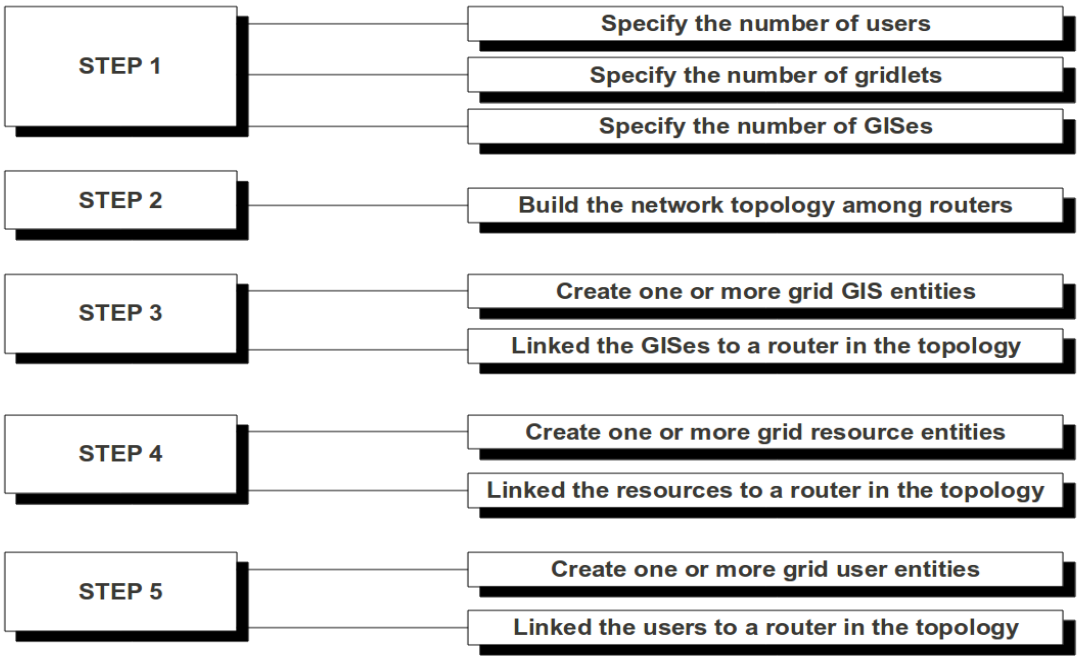


Figure 1.1: GridSim high-level steps

Moreover, the resources and users are also connected to various GISs in the network.
One or more resources which contain one or more users can be inserted in any GIS.
Various routers link the entities, drawing the character of the network that will be created.
In fact, as we shall see, based on our changes we can choose adaptively all five steps listed above, unlike original toolkit that provides a random choice.

These five steps concerning the entities of GridSim toolkit, are the first of a series of stages required to simulate a topology of the performing network.
Depending on the choice of the entities of the simulation, other parameters will be set that affect performance.

1.3 The contents of the thesis

The rest of this thesis consists of seven chapters listed below.
In Chapter 2 we will talk about grid computing and simulation in grid computing. Chapter 3 will be totally dedicated to GridSim toolkit.
In Chapter 4 we will explain in detail the topologies of the two scenarios that we have chosen for our simulations.
Chapter 5 contains all the results and comments on tests performed on the The Scenario of EU DataGrid Test Bed 1. While Chapter 6 contains all the results and comments on tests performed on the Scenario Artificial EU.
In Chapter 7 we will insert all the comments about the comparison among the two scenarios.
And Chapter 8 will be totally dedicated to the final conclusions.

2 Simulation in grid computing

2.1 What is grid computing

Grid computing is the act of sharing tasks over multiple computers.
Tasks can range from data storage to complex calculations and can be spread over large geographical distances.
In some cases, computers within a grid are used normally and only act as part of the grid when they are not in use.
These grids scavenge unused cycles on any computer that they can access, to complete given projects.[4]
SETI@home is perhaps one of the best-known grid computing projects, and a number of other organizations rely on volunteers offering to add their computers to a grid.
These computers join together to create a virtual supercomputer as shown in Figure 2.1.

Networked computers can work on the same problems, traditionally reserved for supercomputers, and yet this network of computers is more powerful than the supercomputers built in the seventies and eighties.

Modern supercomputers are built on the principles of grid computing, incorporating many smaller computers into a larger whole.

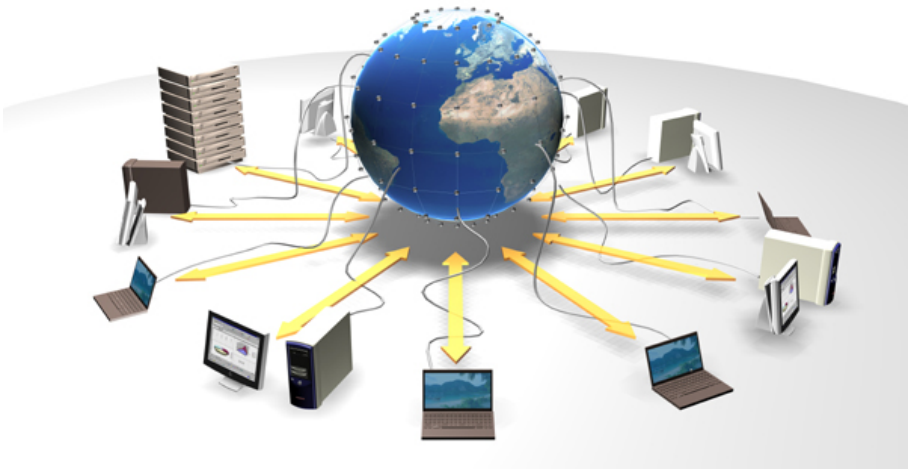


Figure 2.1: How grid computing works[5]

The idea of grid computing originated with Ian Foster, Carl Kesselman and Steve Tuecke. They got together to develop a toolkit to handle computation management, data movement, storage management and other infrastructure that could handle large grids without restrict-

ing themselves to specific hardware and requirements.
The technique is also exceptionally flexible.
Grid computing techniques can be used to create very different types of grids, adding flexibility as well as power by using the resources of multiple machines.

An equipment grid will use a grid to control a piece of equipment, such as a telescope, as well as analyze the data that equipment collects. A data grid, however, will primarily manage large amounts of information, allowing users to share access.

Grid computing is similar to cluster computing, but there are a number of distinct differences.
In a grid, there is no centralized management; computers in the grid are independently controlled, and can perform tasks unrelated to the grid at the operator's discretion.
The computers in a grid are not required to have the same operating system or hardware.
Grids are also usually loosely connected, often in a decentralized network, rather than contained in a single location, as computers in a cluster often are.[6]

2.2 Introduction to the simulation in grid computing

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.[7]

In recent years the demand for resources on a large scale has led to the development of two possible solutions: the grid and peer-to-peer (P2P) computing networks.

The grid consists of four key layers of components: fabric, core middleware, user-level middleware, and applications. 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.[7]

In large-scale grid environments, 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.

To meet the quality of the userâs requirements, the broker dynamically leases grid resources and services at runtime depending on their capability, cost, and availability. Several experiments were conducted with the dates-insensitive implementation schedule, particularly in different branches of science.

The ability to experiment with a large number of grid scenarios was limited by the number of resources that were available in the WWG (World-Wide Grid) testbed. Also, it was impossible to create a repeatable and controlled environment for experimentation and evaluation of scheduling strategies. This is because resources in the grid span across multiple administrative domains, each with their own policies, users, and priorities.[7]

The simulation can solve many (all) difficulties of the grid research, indeed it doesn't need to build a real system, it conducts controlled/repeatable experiments, in principle there

aren't limits to experimental scenarios and it's possible for anybody to reproduce results.

What is simulation? (2.1) It's a "representation of the operation of one system (A) through the use of another (B)".

$$Computer\ Simulation \rightarrow\ B\ \equiv\ a\ computer\ program. \tag{2.1}$$

The key question is the validation, namely the correspondence between simulation and real-world. In the computer science world we can find many types of simulation, as Microprocessor Design, where a few standard "cycle-accurate" simulators are used extensively with the possibility to reproduce simulation results; or Networking where there are a few standards of "packet-level" simulators, well known datasets for network topologies, and well-known generators of synthetic topologies with the possibility to reproduce simulation results.[8]

In the grid computing, until a few years ago, we had none of the above, because most people built their own solutions; but we have promising recent developments.
The simulation of parallel platforms was used throughout the last ten years, and it represents the simplistic platform model where the topology is fully connected (no communication interference) or a bus (simple communication interference), and the communication and computation are perfectly overlappable.

Furthermore it represents the simplistic application model where all computation is CPU intensive, where there are clear-cut communication and computation phases, and where the application is deterministic.

In the grid computing simulations, the simple models are used. The simulation grids hardly suffice for grid platforms because the network topologies are complex and wide-reaching, they involve an overhead of middleware, it is complex to access/manage the policies and there can be an interface of communication and computation.

The goals of simulations are essentially two: the first is to simulate the platforms beyond the ones at hand and the second is to perform the sensitivity analyses. To reach these goals we need a synthetic platform which examines the real platforms, discovers the principles and implements the "platform generators".[8]

The generation of synthetic grids consists of three main elements: Network Topology, Compute Resources, "Background" Conditions. The network topology (illustrated in Figure 2.2) is composed of:

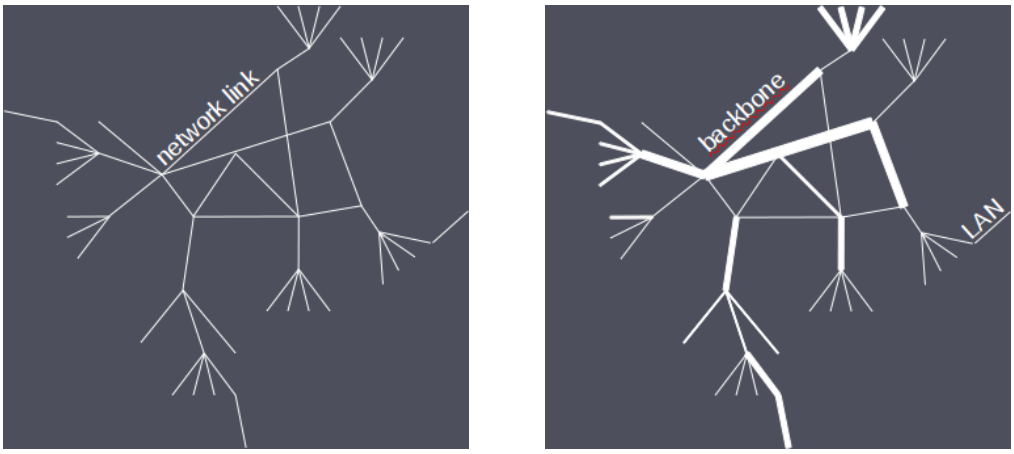
- Graph Figure 2.2(a)
- Bandwidth and Latencies Figure 2.2(b)

The network community has wondered about the properties of the Internet topology for years, beacuse Internet grows in a decentralized fashion with seemingly complex rules and incentives.

The compute resources and the other resources are illustrated in Figure 2.3.

The background conditions (illustrated in Figure 2.4) that may occur are:

- Load and unavailability Figure 2.4(a)
- Failures Figure 2.4(b)



(a) Graph of a network topology [8]

(b) Bandwidth and Latencies [8]

Figure 2.2: Global network topology

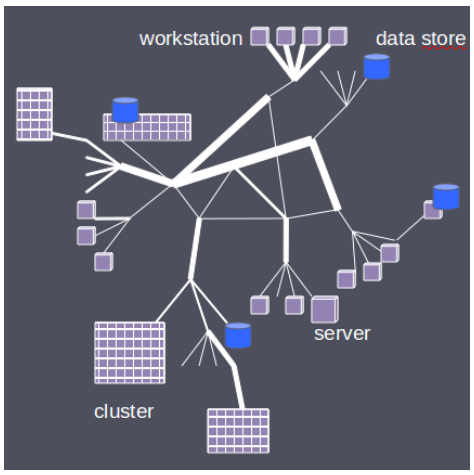
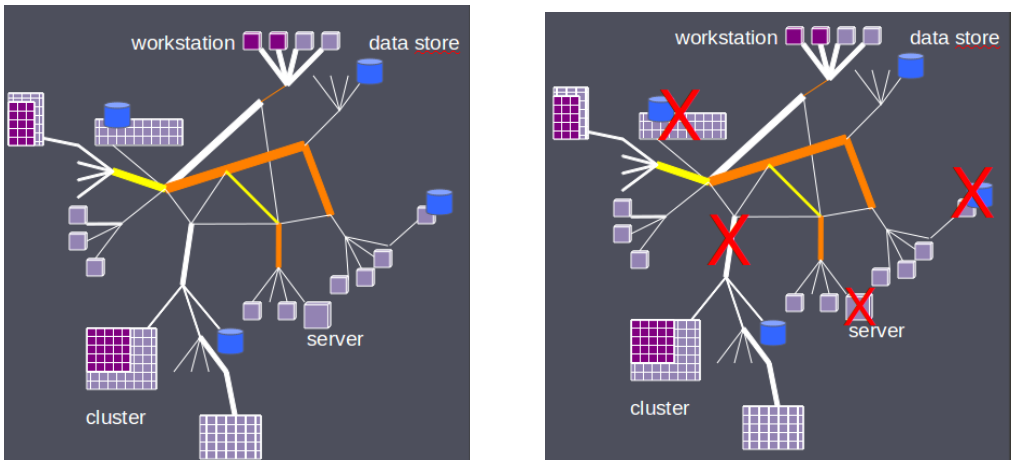


Figure 2.3: Resources [8]

2.3 What do we need for the simulation in grid computing?

As seen above, the simulation is a complex process.
It requires planning in advance for any component that will create and implement the process simulation.
The main components of a well-designed simulation are:

- A *network topology* is the representation of the geometrical structure of a telecommunications network.
  A network topology represents a geometric model (graph) of a telecommunications network whose elements are the nodes and branches.
  A node detects a network element characterized by specific features, which in our case will be routers and resources.
  A branch is an element of connection between two nodes, which in our case are the



(a) Load and unavaiability [8]

(b) Failures [8]

Figure 2.4: Background conditions

links.
A node can have one or more connections with others according to different schedules.
The connection used in GridSim is bidirectional.
The network topology is determined only by the configuration of connections between nodes.
In the network topology we can have: the distance between nodes, physical interconnections, transmission rates and the length of the communication that you want to enter the network.
Two nodes can be placed in communication through a physical connection between two nodes when there is a physical channel that connects them directly.

- A *router* in a logical level, is a deputy to the switching network node level 3 of the OSI (Open Systems Interconnection) model.
  In other words, the physical level is a network device that takes care of routing information packets working at level 3 (network) of the OSI model or as a route between two or more neighboring subnets through their interfaces, each with their address, or to other subnets through non-adjacent routing tables on the transport network.
  The type of routing function is called indirect addressing opposite instead to the direct addressing typical of transport within the subnet.
  The placement of the routers in the network topology is a fundamental choice in order to achieve optimal performance from the entire network.
  In the simulation of grid computing the router is never an end node, but only serves as a connector between other nodes of the network topology, the resources.

- A *resource* represents the final node in the network topology of a simulation of grid computing.
  It has the task of receiving the information, processing it and then returning it to the sender.
  In the network topologies in large-scale, resources are usually indicated with the city that have research centers grid and therefore possess the right tools to be able to start

a simulation.
Then we can provide the truthful parameters.

- A *user* represents an initial node of the network topology.
  In reality a user resides in a resource, and as a result can not be considered a node in its own right.
  In fact it is the user who sends a signal the network because his own resource is unable to perform the jobs required by it.

- A *regional GIS* is a region of the network topology.
  In it there can be multiple nodes, then one or more routers that are connected to one or more resources.
  A grid information service (GIS) is an entity that provides grid resource registration, indexing and discovery services.
  The grid resources tell their readiness to process jobs by registering themselves with this entity.

3 GridSim Toolkit

The GridSim toolkit is one of the most widely used grid simulation tools.
It has been used for simulating and evaluating VO-based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global grids.[9]

3.1 Formation of GridSim

One of the many features of the toolkit GridSim, is the ability to simulate application schedulers for single or multiple administrative domains.
The resource brokers are the application schedulers in the GridSim and in all the grid enviroment. They perform selection, resource discovery and aggregation of several sets of resource for every user.
In few words every user has its own private resource broker, and each resource can be targeted to optimize for the requirements of its owner. Thus all the users need to submit their gridlets (their jobs) to the central scheduler. After that the gridlets can be targeted to perform the global optimization.

3.1.1 GridSim Features

We now list the main features of the toolkit GridSim that were relevant to our simulations.

- Heterogeneous resources can be modeled in GridSim.
- Resource capability can be defined in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark.
  In our simulations we will use MIPS as the standard units, in line with the decisions of the University of Melbourne.
- Each resource can be located in any time zone, always.
- Each resource can be booked for an advance reservation.
- The applications with different parallel application models can be simulated.
- The application tasks can be heterogeneous and they can be CPU or I/O intensive.
- Every gridlet (jobs) can be submitted to a resource every time without any kind of limit.
- Multiple users can simultaneously send multiple gridlets to the same resource.
  This makes it possible to build different models for any kind of solution.
- The network speed between resources and between routers can be specified.
- GridSim makes it possible to record serious and very broad statistics related to testing in simulation.

3.1.2 GridSim system architecture

The multi-layer architecture and abstraction for the development of GridSim platform and its applications is shown in Figure 3.1.

In the *first layer* we can find the JVM (Java Virtual Machine) that represents the interface and runtime machinery of Java. The implementation of JVM is avaible for single-multi processor systems including clusters.

The first layer provides interfaces that are used by the *second layer*. Indeed a basic discrete event infrastructure is built using these interfaces.

The most important grid entities are treated in the *third layer*, in which they are modeled and simulated. With the most important grid entities we mean resources, information servicies, and so on. The GridSim toolkit focuses on this layer that simulates system entities using the discrete-event services offered by the lower-level infrastructure.

In the *fourth layer* we can find the simulation of resource aggregators called grid resource brokers or schedulers.

The *fifth and final layer* is focused on application and resource modeling with different scenarios. This is possible using the services provided by the two lower-level layers for evaluating scheduling and resource management policies and algorithms.

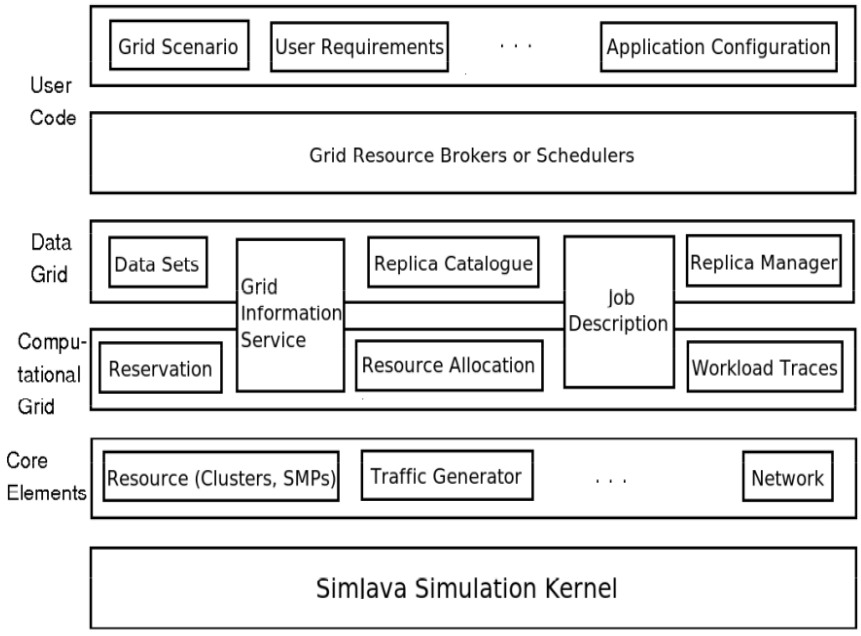


Figure 3.1: Architecture of GridSim [9]

A discrete event model: SimJava

SimJava[10] is a general purpose discrete event simulation package implemented in Java.[7]
In SimJava the simulations have some entities and each entity runs with its own process in parallel. They use its *body()* method to encoded the behaviour.
These entities have access to some simulation primitives such as:
sim schedule() that sends event objects to other entities through ports;
sim hold() has the task of holding for some simulation time;
instead *sim wait()* is a primitive that waits for an event object to arrive.
In this way it is possible to build a topology network that is able to support entities that communicate through sending and receiving passive event objects efficiently. The algorithm that describes the simulation of events is explained below.
In SimJava there is an object, called Sim system, whose task is to maintain a timestamp of orderly queues for all future events. After that:

1. All entities will be created.

2. The method *body ()* of the entities will be put in run state.

3. When a simulation function is called by an entity, the thread of that entity will be stopped and an event will be placed in the queue of future events. In this way the function is in progress.

4. Only when all the entities halt, the Sim system object will pop the next event off the queue, if the simulation times accordingly, and will restart entities as appropriate.

5. The previous step will be repeat until no more events are generated.

- JVM can support native threads. Only in this case all entities can start at the same simulation time.

The entities of GridSim

One of the main characteristics of GridSim toolkit is that it supports entities for simulation of single processors and multiprocessors.. It also supports simulation of heterogeneous resources that are to be configured according to time-sharing, when they are configured according to the space-sharing.
GridSim also allows the possibility to set the time according to different areas where it is carrying out the simulation.

A simulation environment needs to abstract all the entities and their time-dependent interactions in the real system. It needs to support the creation of user-defined time-dependent response functions for the interacting entities. The response function can be a function of the past, current, or both states of entities.[7]

GridSim based simulations contain entities for the users, brokers, resources, information service, statistics, and network based I/O, as shown in Figure 3.2. The design and implementation issues of these GridSim entities are discussed below.

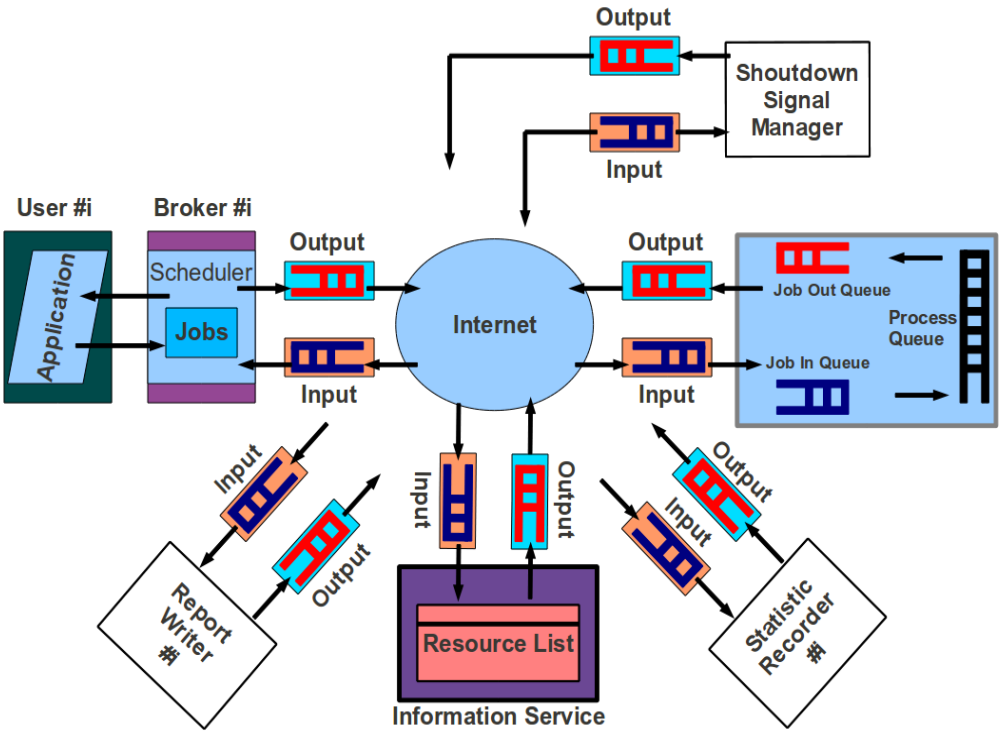


Figure 3.2: A flow diagram in GridSim based simulations [7]

Entity user

Every user in GridSim is represented as an instance of the entity grid user. Each user has some parameters that can differentiate from others; in our case, they are:

- types of job created, e.g. job execution time, number of parametric replications, etc.;
- scheduling optimization strategy, e.g. minimization of cost, time, or both;
- activity rate, e.g. how often it creates new job;
- time zone.

Entity broker

Through the entity broker, each user connects to the network topology. Each user's job is first submitted to its broker and after that the broker ccan schedule the tasks according to the policy of the user. But before scheduling the tasks, the broker must receive the list of resources available 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 supply and demand situation.[7]

Entity resource

Every resource in GridSim is represented as an instance of the entity grid resource. Each resource has some parameters that can differentiate it from others; in our case, they are:

- the number of processors;
- the cost of processing;
- the speed of processing;
- the time zone.

MIPS and SPEC are the units of measure of the resource's speed and the job execution time. Moreover they can be defined with respect to the standard machine. Upon obtaining the resource contact details from the grid information service, brokers can query resources directly for their properties.

Entity grid information service

The grid information service is a very important entity that provides the registration service to a resource and moreover it keeps track of all resources available in the grid by a list.
This list may be questioned by the broker to obtain information such as resource contact, configuration, and status.

Entities input and output

The input and output entities control the flow of information between the entities of GridSim toolkit. In each of the GridSim entities there are channels or ports of input and output, and their task is to establish a link between the entity and its entities of input and output.
An important fact to keep in mind is that the GridSim entity and its Input and Output entities are threaded entities, i.e. they have their own execution thread with *body()* method that handles events.
Figure 3.3 illustrates the architecture for the entity communication model in GridSim.

3.2 GridSim model

One of the features that made us opt for GridSim is the ability to specify the application model through schedulers and resource brokers. Now we describe the most important objects of GridSim.

3.2.1 Gridlet

In GridSim toolkit toolkit, each time we send in a new piece of business, we must keep in mind that it may vary depending on process time and size of the input file. This activity and all its components are created through an object called a gridlet.
A *gridlet* is a package that contains all the information related to the job and its execution management details such as job length expressed in MIPS, disk I/O operations, the size of input and output files, and the job originator.[7]
All these parameters play a key role in GridSim. In fact, thanks to them we can determine a range of information such as:

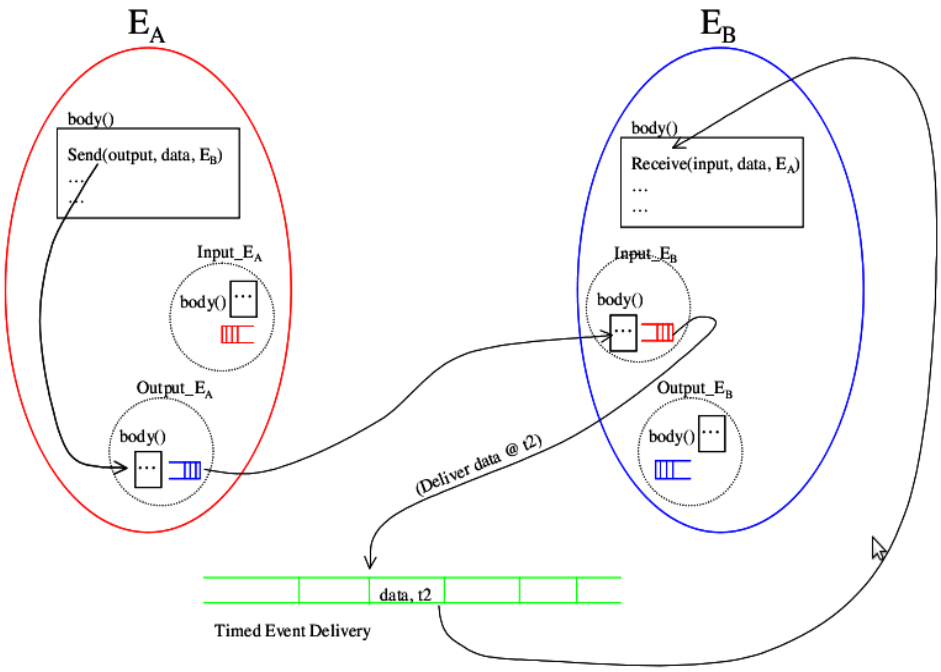


Figure 3.3: Entity communication model via its Input and Output entities [7]

- the execution time;
- the time required to transport input file between users and resources;
- the time required to transport output file between users and resources;
- the time required to return the processed gridlet back with the results.

3.2.2 Event

The *event* is an important object because it allows interaction between the various entities of GridSim.
Events are used by entities for the service request and for the service delivery.

All the entities of GridSim (user, broker, resource, information service, statistics, shutdown, and report writer) use events to send service requests to other entities, to deliver results, or to raise some actions.
We must remember that GridSim implements core entities that simulate resource, information service, statistics, and shutdown services. This range of services is used to simulate some behavior, such as a user with application, a broker for scheduling, and a report writer for creating statistical reports at the end of a simulation.
At the beginning of the simulation an entity sends an event to the grid information service (GIS) to encode itself and go into action.
After that the GIS entity returns a list of all registered resources and their contact details.
In the next step the broker entity sends an event to all the resources which contains a request for the resource configuration and properties.

The resources respond with information that could be dynamic as resource cost, availability, load, capability and other parameters of the configuration.
At the end of the gridlet processing, the resource entity updates the processing time and the gridelt status and then it sends it back to the broker who had raised the event. This means for the broker that the event is finished.

Each entity must be implemented so that they can receive all events that have meaning for it. However, it is the entity that decides the associated actions.

3.3 The multitasking and multiprocessing mode

The Processing Elements (PEs) represent a fundamental unit in GridSim toolkit. Indeed it is through the PEs that GridSim can create machines, stitching together one or more PEs. Each PE has different speed and it is measured in MIPS or SPEC-like ratings.
Moreover, all of the machines can be put together to create a grid resource.
Thus we can obtain a grid resource that can be:

- a single processor;
- a multiprocessor, or better a shared memory multiprocessor (SMP);
- a distributed memory cluster of computers.

These Grid resources can simulate time- or space-shared scheduling depending on the allocation policy.

To manage all the distributed memory multiprocessing systems (such as clusters), GridSim uses a queuing system called a space-shared scheduler.
This scheduler executes a gridlet by running it on a dedicated PE when allocated.
The most common policies for allocating the resources in the space-shared systems are: first-come-first-served (FCFS), back filling, shortest-job-first-served (SJFS).
Multitasking and multiprocessing systems allow concurrently running tasks to share system resources such as processors, memory, storage, I/O, and network by scheduling their use for very short time intervals.
In the real systems a detailed simulation of scheduling tasks would be very complex and very costly in terms of time.
The events that simulate the execution of jobs, can be sent, received, or scheduled events by GridSim.
The GridSim schedules self-events for simulating resource allocation depending on the scheduling policy and the number of jobs in the queue or in execution.

3.4 GridSim Java package design

The GridSim may be represented by a hierarchy diagram.
The specification of each class contains up to three parts: attributes, methods, and internal classes.
The GridSim package implements the following classes.

- *class gridsim.Input*: this class allows us to define a port in which the simulation entity receives some data from the simulated nerwork.
  It maintains an event queue to serialize the data in flow.
  Simultaneous inputs can be modeled using multiple instances of this class.

- *class gridsim.Output*: this class is very similar to the *gridsim.Input* class. The output class allows us to define a port in which the simulation entity sends some data from the simulated nerwork.
  It maintains an event queue to serialize the data-out-flow.
  Simultaneous outputs can be modeled by using multiple instances of this class.

- *class gridsim.GridSim*: this is the main class of the GridSim package. In every class that extends the GridSim class, a method called *body()* must be implemented.
  This method is automatically invoked since it is expected to be responsible for simulating entity behavior. The entities that extend the GridSim class can be instantiated with or without networked I/O ports.
  The classes of input and output (*gridsim.Input* and *gridsim.Output*) provide communication capabilities to networked GridSim through the I/O objects of GridSim. Every I/O entity has a unique name because we assume each GridSim entity that the user creates has a unique name.

  In the *GridSim* class there are methods that support simulation initialization, management, and flow control. Before creating any other GridSim entities at the user level, the GridSim environment has to be initialized to set-up the simulation environment.
  This method also prepares the system for simulation by creating three GridSim internal entities: *GridInformationService*, *GridSimShutdown*, and *GridStatistics*.
  The *GridSim* class supports static methods for sending and receiving messages between entities directly or via network entities, managing and accessing handles to various GridSim core entities, and recording statistics.

- *class gridsim.PE*: in this class, the CPU/PE is represented, which is the capability defined in terms of MIPS rating.

- *class gridsim.PEList*: this class maintains a list of PEs that make up a machine.

- *class gridsim.GridResource*: as well as extending the class *GridSim*, it gains communication and concurrent entity capability.
  An instance of this class simulates a resource with properties defined in an object of the class in which the characteristics of the resource are defined.
  (*class gridsim.ResourceCharacteristics*).
  The process of creating a Grid resource is as follows:

  1. create PE objects with a suitable MIPS/SPEC rating;
  2. assemble them together to create a machine;
  3. group one or more objects of the machine to form a resource.

  A resource having a single machine with one or more PEs is managed as a time-shared system using a round-robin scheduling algorithm.

A resource with more then one machines is treated as a distributed memory cluster and is managed as a space-shared system using FCFS scheduling policy or some of its variants.

- *class gridsim.GridInformationService*: this is a a GridSim entity that provides grid resource registration, indexing and discovery services. GridSim entities such as the resource broker can contact this entity for resource discovery service, which returns a list of registered resource entities and their contact addresses. The grid resources register their readiness to process gridlets by registering themselves with this entity.

- *class gridsim.Gridlet*: in this class there is a job package that contains job length in MI, the length of input and output data in bytes, execution start and end time, and the originator of the job.
  The users create gridlets to send to the resource grid according to their needs.

- *class gridsim.GridletList*: can be used to maintain a list of Gridlets and support methods for organizing them.

- *class gridsim.ResGridlet*: this class represents a Gridlet that is submitted to the resource to be processed. It acts as a placeholder for maintaining the amount of resource share allocated at various times for simulating time-shared scheduling using internal events. It contains a Gridlet object along with its arrival time and the ID of the machine and the PE allocated to it.

- *class gridsim.GridStatistics*: in this class, all of the statistical data reported by other entities is provided. This class stores data objects with their label and timestamp.
  When the simulation ends, the entity user-defined report-writer can query the statistics of interest to obtain a general report.

- *class gridsim.GridSimShutdown*: this is a GridSim entity that waits for termination of all user entities to determine the end of simulation.
  It then signals the user-defined report-writer entity to interact with the GridStatistics entity to generate a report.
  Finally, it signals the end of simulation to other GridSim core entities.

- *class gridsim.GridSimRandom*: this class provides static methods for incorporating randomness in data used for any simulation.

In this section we have listed and briefly described only some classes of GridSim Java package (the most relevant to our work).
For further discussion and for a comprehensive description of GridSim Java package, refer to [7].

3.5 How did we modify the toolkit?

As we mentioned in Chapter 1 and described briefly in the previous section 3.4, much of the original toolkit is based on a set of random functions that automatically set certain parameters.
The GridSim toolkit was designed this way to reflect the uncertainty that is present in the

prediction/estimation process and the randomness that exists in the nature.
Our goal is to construct scenarios that are first of all solid but mostly reliable on a large scale. For this we need to simulate the tests by manually entering a number of parameters.

In this section we present our principal modifications to the GridSim toolkit.
Most of the changes concern the possibility to render some methods manually instead of randomly, in order to simulate the possible case studies in certain network topologies.

As we mentioned in the 3.1.2, GridSim supports entities for simulation and an entity's behavior needs to be simulated within its *body()* method. Therefore our changes are focused on the methods of the entities, inside the toolkit's code.
Now we briefly describe the key variables for our simulations.

- ***User***: an application or a broker tthat schedules jobs onto grid resources is considered as a user.
  Such components are able to query and request dataset transfers, submit jobs and register for events.[11].

- ***Resource***: in grid computing, any hardware or software component such as a cluster, a supercomputer or a storage repository is called a resource.
  Computing resources allow users to execute the required application.

- ***Router***: users and resources are connected to routers.

- ***Machine***: a Grid resource contains one or more machines. Similarly, a machine contains one or more PEs.

- ***PE***: Processing Elements or CPUs.

- ***Gridlet***: GridSim already has the ability to schedule compute-intensive jobs, which are represented by a gridlet class.
  Each data-intensive job has a certain execution size (expressed in Millions Instructions (MI)) that will be used by a resource to determine how much simulation time is required [11].

- ***RegionalGIS***: a Grid Information Service (GIS) is an entity that provides grid resource registration, indexing and discovery services.
  The Grid resources tell their readiness to process Gridlets by registering themselves with this entity.
  In addition, GIS is responsible for notifying all the registered entities, such as GridResource and network entities for shutting down at the end of a simulation.

- ***Link***: defines connection between Router-Router, Router-Resources, Router-Users.

We will simulate a small network topology in which we have two Regional GIS, two Routers, two Users and two Resources.
But first of all we introduce in detail the process of creating a simulation of a network topology.

3.5.1 Simulation steps in the original GridSim

In this section we introduce briefly the generic process in GridSim to set up and start a simulation.
This will make it easier to understand the various changes that are introduced in the following sections.

- First, we need to create grid resources of different capabilities and configurations (a single or multiprocessor with time/space-shared[1] resource manager).
  We also need to create users with different requirements (application and quality of service requirements).
  A sample code for creating a grid environment is given in List of Figures 3.1.

Listing 3.1: A sample code segment for creating grid resource and user entities in GridSim

```
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();
  String ReportWriterName = "MyReportWriter";
  GridSim.Init(no_of_users, calender, true, eff, efp,
     ReportWriterName);
  String[] category = {"*.USER.TimeUtilization", "*.USER.
     GridletCompletionFactor", "*.USER.BudgetUtilization"};

  // Create Report Writer Entity and category indicates types
     of information to be recorded.
    new ReportWriter(ReportWriterName, no_of_users,
       no_of_resources, ReportFile, category,
    report_on_next_row_flag);

  // 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++)
```

[1] If the failure only affects some of the machines in a resource, what happens next depends on the allocation policy of this resource.
If the resource runs a space-shared (first come first serve) allocation policy, the jobs that are currently running on the failed machines will be terminated and sent back to users.
However, when the resource runs a time-shared (round-robin) allocation policy, no jobs will be failed, as their execution will continue in the remaining machines of the resource.
For both allocation policies, the remaining machines are responsible for responding to polling requests from users and GIS.

```
        peList.add(new PE(0, 100));

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

      // 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++)
      {
        Random r = new Random(seed*997*(1+i)+1);
        GridletList glList = Application1(r); // it creates
            Gridlets and returns their list
        Experiment expt = new Experiment(0, glList, policy, true
            , B_factor, D_factor);
        new UserEntity("U"+i, expt, 28000.0, how_long, seed
            *997*(1+i)+1, i, user_entity_report);
      }

    // Perform Simulation
      GridSim.Start();
}
```

- Second, we need to model applications by creating a number of gridlets and define all parameters associated with jobs as shown in List of Figures 3.2.
  The gridlets need to be grouped together depending on the application model.
- Then, we need to create a GridSim user entity that creates and interacts with the resource broker to schedule an entity to coordinate execution experiment.

It can also directly interact with GIS and resource entities for grid information and submitting or receiving processed gridlets.
However, best way is to implement a separate resource broker entity by extending the GridSim class.

Listing 3.2: The gridlet method in GridSim

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

- Finally, the scheduler accesses the GIS, and then inquires for resource capability including cost.
  Depending on the processing requirements, it develops a schedule for assigning gridlets to resources and coordinates the execution.
  The scheduling policies can be systems-centric like those implemented in many grid systems

3.5.2 Initialize the GridSim Package

Original toolkit

When we want to initialize the GridSim package in the original toolkit, as shown in List of Figures 3.3, we have to check that the external file with the network topology exists and contains all of the information that we need to know.
Then we have to specify the number of users, resources and GISs, that in our example case are all set at two, and initialize the random variable.

The second step consists of initializing a flag that denotes whether to trace GridSim events or not, and to initialize the calendar variable.

The third and last step of this section regards the real initialization of the GridSim package. It should be called before creating any entities.

We can't run this example without initializing GridSim first, or we will get a run-time exception error.

Listing 3.3: Code segment for initializing the GridSim Package

```
// [1] Check the network topology

      if (args.length < 1)
      {
        System.out.println("Usage: java Ex03 network_ex03.txt");
        return;
      }

// specify the number of users, resources and GISs
```

```
        int num_user = 2;              // number of grid users
        int totalResource = 2;         // number of resources
        int num_GIS = 2;               // number of Reg GIS entity
        Random random = new Random(); // randomize the selection

// [2] Initialize a flag

        boolean trace_flag = false;  // don't use SimJava trace
        Calendar calendar = Calendar.getInstance();

// [3] Initialize the GridSim package

        System.out.println("Initializing GridSim package");
        GridSim.init(num_user, calendar, trace_flag);
        trace_flag = true;
```

Our toolkit

The only difference in this section regards the initialization of the random variable. Being a manual process, the *Random()* instace should not be initialized.

3.5.3 Builds the network topology among routers

Original toolkit and Our toolkit

This section consists of reading and building the network topology from an external file, as shown in List of Figures 3.5

This file contains all of the data of the network that we want to simulate.

Listing 3.4: Network Topology

```
# total number of Routers
2

# specifies each router name and whether to log its activities
   or not
# by default no logging is required
Router0
Router1

# specify the link between two Routers
# The format is:
# Router_name1  Router_name2  baud_rate  prop_delay   mtu
                              (GB/s)       (ms)      (byte)
  Router0         Router1        1          10.0       1500
```

In our case, this file is shown in List of Figures 3.4 and contains the total number of the routers, the name of these routers and the links between routers. The format of the

link provides, for every two routers, the baud rate, the propagation of the delay and the maximum transmission unit.
The baud rate represents the network communication speed between two routers and it's expressed in GB/s.
The propagation delay is expressed in milliseconds.
The maximum transmission unit is expressed in bytes.

Listing 3.5: Code for getting and building a network topology

```
String filename = args[0];  // get the network topology
System.out.println("Reading network from " + filename);
LinkedList routerList = NetworkReader.createFIFO(filename);
```

With the action *String filename = args[0]* we get the network topology and we put all the information in the variable *filname*.
Then we create a list of routers (*routerList*) type *LinkedList* with the routers contained in the network topology. Furthermore the type *LinkedList* automatically connects the routers present in the external file.

3.5.4 Regional GIS Code Changes

Original toolkit

In this section we want to show how to create a new regional GIS entity in the original toolkit, as shown in List of Figures 3.6.
In the first action we have to set the variables that we need to create a regional GIS entity: baud rate, propagation delay, maximum transmission unit and the total number of machines for each resource.
In the second passage, for each regional GIS, we set the name.
In the third step, we create for each regional GIS, a network link with all the informations set before.

Listing 3.6: Code segment for creating regional GIS entity and linking it to a router

```
[...]

// [1] Creates one RegionalGIS entity, linked to a router in the
    topology

  double baud_rate = 100000000;
  // 100Mbps, i.e. baud rate of links

  double propDelay = 10;
  // propagation delay in milliseconds

  int mtu = 1500;
  // max. transmission unit in byte

  int totalMachines = 16;
  // num of machines each resource has
```

```
    for (int i = 0; i < num_GIS; i++)
    {

// [2] GIS name
        String gisName = NAME + "Regional_GIS_" + i;

// [3] a network link attached to this regional GIS entity
        Link link = new SimpleLink(gisName + "_link", baud_rate,
                                   propDelay, mtu);

// [4] creates a new Regional GIS entity

        RegionalGIS gis = new RegionalGIS(gisName,link);
        gis.setTrace(trace_flag);   // record this GIS activity
        gisList.add(gis);   // add into the list

// [5] link these GIS to a router
        String routerName = null;
        if (random.nextBoolean() == true)
        {
            linkNetwork(router0, gis);
            routerName = router0.get_name();
        }
        else
        {
            linkNetwork(router1, gis);
            routerName = router1.get_name();
        }

// [6] print some info messages
        System.out.println("Created a REGIONAL GIS with name " +
                gisName + " and id = " + gis.get_id() +
                ", connected to " + routerName);
```

The goal of the fourth step is to create a new instace of the class *RegionalGIS*:
RegionalGIS gis = new RegionalGIS(gisName, link);,
where *gisName* and *link* are variables set before.
Subsequently we have to record the activity of GIS just created and finally add the new GIS into a list of all the existing GIS.
The fifth step consists of linking the GIS to a router in the random way. To obtain the random way, we use the random algorithm that we have specified in 3.4.
In this example, if the random algorithm returns true, we link the current GIS to the first router (*router0*), else we link the current GIS to the second router (*router1*).
The sixth and last step is to print on the screen all the information of the newly created GIS.

Our toolkit

To reach our goal, we have to change the way through which the GISs are linked to the router. In this manner we can set and control the network. As shown in List of Figures 3.7, the first four steps and the last step are the same in both in the toolkit.
What changes is the portion of code in the fifth step.

Listing 3.7: Creating regional GIS entity and linking it to a router manually

```
[...]

// [1] Creates one RegionalGIS entity, linked to a router in the
    topology

  double baud_rate = 100000000;
  // 100Mbps, i.e. baud rate of links

  double propDelay = 10;
  // propagation delay in milliseconds

  int mtu = 1500;
  // max. transmission unit in byte

  int totalMachines = 16;
  // num of machines each resource has

  for (int i = 0; i < num_GIS; i++)
  {

// [2] GIS name
      String gisName = NAME + "Regional_GIS_" + i;

// [3] a network link attached to this regional GIS entity
      Link link = new SimpleLink(gisName + "_link", baud_rate,
                                 propDelay, mtu);

// [4] creates a new Regional GIS entity

      RegionalGIS gis = new RegionalGIS(gisName,link);
      gis.setTrace(trace_flag);    // record this GIS activity
      gisList.add(gis);    // add into the list

// [5] link these GIS to a router
      String routerName = null;
      switch (i){
        case (0):
              linkNetwork(router0, gis);
              routerName = router0.get_name();
```

```
                    break;

            case (1):
                    linkNetwork(router1, gis);
                    routerName = router1.get_name();
                    break;

            default:
                    System.out.println("Not a recognized test case."
                        );
                    break;
            }

    // [6] print some info messages
            System.out.println("Created a REGIONAL GIS with name " +
                    gisName + " and id = " + gis.get_id() +
                    ", connected to " + routerName);
```

Indeed here we substitude the *if* control with a *switch-case* structure.
The value of the variable given into switch represents the current GIS and, in our example, we want to link *GIS0* to *router0* and *GIS1* to *router1*.
In this simple (but not too flexible) way, we can decide the structure of all the GISs instead of obtaining it by the random mode.

3.5.5 Resource Code Changes

Original toolkit

After initializing the GIS instance, the next procedure is to create the instance for the resource. This phase is represented in List of Figures 3.8.
The first step consists of setting the variables necessary to formulate the grid resource. The variables considered are: the total number of processing elements that each machine has; the rating of PEs, expressed in million instruction per second; and the baud rate expressed in gigabytes.
In the second step, we set the name of the resource and then we proceed with the initialization of the new instance of the resource. Subsequently we have to add the new resource into a list of all the existing resources and, finally, record the activity of the resource just created.
In the third passage we link the instance of the resource to a router in a random way (as the same algorithm of the previous section).

Listing 3.8: Creating a grid resource and linking it to a router

```
    [...]
    // [1] Set all the variables

        ArrayList resList = new ArrayList(totalResource);

        // Each resource may have different baud rate,
        // totalMachine, rating, allocation policy and the router to
```

```
    // which it will be connected. However, in this example, we
       assume
    // all have the same properties for simplicity.
    int totalPE = 4;          // num of PEs each machine has
    int rating = 49000;       // rating (MIPS) of PEs
    int GB = 1000000000;      // 1 GB in bits
    baud_rate = 2.5 * GB;

    for (int i = 0; i < totalResource; i++)
    {
// [2] creates a new grid resource

        String resName = NAME + "Res_" + i;
        GridResource res = createGridResource(resName,
                    baud_rate, propDelay, mtu, totalPE,
                       totalMachines,
                    rating, sched_alg);

        if (i % 2 == 0) {
            trace_flag = true;
        }
        else {
            trace_flag = false;
        }

        resList.add(res);    // add a resource into a list
        res.setTrace(trace_flag);    // record this resource
           activity

//[3] link these RES to a router

        String routerName = null;
        if (random.nextBoolean() == true)
        {
            linkNetwork(router0, res);
            routerName = router0.get_name();
        }
        else
        {
            linkNetwork(router1, res);
            routerName = router1.get_name();
        }

//[4] randomly select which GIS to choose

        int index = random.nextInt( gisList.size() );
        RegionalGIS gis = (RegionalGIS) gisList.get(index);
```

```
            res.setRegionalGIS(gis); // set the regional GIS entity

    // [5] print some info messages

            System.out.println("Created " + resName + " with id = "
                +
                    res.get_id() + ", linked to " + routerName +
                    " and registered to " + gis.get_name() );
        }
```

The goal of the fourth step is to select which GIS to choose for each resource. Also in this phase we operate in a random way.
These two steps show us that it is impossible to know the structure of the network, because of the total randomness in the choice of their components.
The fifth and last passage prints the results of the previous steps on the screen.

Our toolkit

For our objectives, we can't utilize the random selection to link the resource to a router and to choose the GIS.
We have to modify the code in order to be able to insert our data manually.
Thus we change the code from the second to the fourth point (List of Figures 3.9) and we obtain a new second step.

Listing 3.9: Creating a grid resource and linking it to a router manually

```
    [...]
    // [1] Set all the variables

        ArrayList resList = new ArrayList(totalResource);

        // Each resource may have different baud rate,
        // totalMachine, rating, allocation policy and the router to
        // which it will be connected. However, in this example, we
           assume
        // all have the same properties for simplicity.
        int totalPE = 4;          // num of PEs each machine has
        int rating = 49000;       // rating (MIPS) of PEs
        int GB = 1000000000;      // 1 GB in bits
        baud_rate = 2.5 * GB;

        for (int i = 0; i < totalResource; i++)
        {

    // [2] creates a new grid resource

            String resName = NAME + "Res_" + i;

            String routerName = null;
```

```
        RegionalGIS gis = null;
        GridResource res = null;

        switch(i)
        {
          case (0):
            res = createGridResource(resName,
            baud_rate, propDelay, mtu, totalPE, totalMachines,
              rating, sched_alg);
            linkNetwork(router0, res);
            routerName = router0.get_name();
            gis = (RegionalGIS) gisList.get(0);
            res.setRegionalGIS(gis);
            break;

          case (1):
            res = createGridResource(resName,
            baud_rate, propDelay, mtu, totalPE, totalMachines,
              rating, sched_alg);
            linkNetwork(router0, res);
            routerName = router0.get_name();
            gis = (RegionalGIS) gisList.get(0);
            res.setRegionalGIS(gis);
            break;

          default:
            System.out.println("Not a recognized test case.");
            break;
        }

        if (i % 2 == 0) {
            trace_flag = true;
        }
        else {
            trace_flag = false;
        }

        resList.add(res);  // add a resource into a list
        res.setTrace(trace_flag);  // record this resource
           activity


// [3] print some info messages

        System.out.println("Created " + resName + " with id = "
           +
                res.get_id() + ", linked to " + routerName +
```

```
                    " and registered to " + gis.get_name() );
        }
```

In point number two, we have changed all of the code.
We inizialize the instace of the objects at null:
String routerName = null;
RegionalGIS gis = null;
GridResource res = null;
and then we substitude the *if* control with a *switch-case* structure.
For every resource we create a new *GridResource* with all the datas declared in the previous variables. The value of the variable given into switch represents the current resource and, in our example, we want to put *resO* in the *GIS0* and link it to *router0*, and put *res1* in the *GIS1* and link it to *router1*.

3.5.6 User Code Changes

Original toolkit and Our toolkit

The construction of the user object is similar to the construction of the resource object. Thus we show only the code of the modified toolkit. In the first step (as we can see at List of Figures 3.10) we set the variables that we need to create a user entity:
totalGridlet indicates the total number of the jobs;
pollTime represents the time between the various polls;
glSize provides the size of the gridlets;
glLength is the length of the standard gridlets, expressed in million instructions.

Listing 3.10: Creating a grid user and linking it to a router manually

```
[...]
// [1] Set all the variables

    ArrayList resList = new ArrayList(totalResource);

    int totalGridlet = 4;     // total jobs
    double pollTime = 100;    // time between polls
    int glSize = 100000;      // the size of gridlets
    long glLength = 420; // the length (MI) of gridlets standard

    for (int i = 0; i < num_user; i++)
    {

// [2] creates a new user entities

        trace_flag = false;
        String userName = NAME + "User_" + i;

        // a network link attached to this entity
        Link link2 = new SimpleLink(userName + "_link",
            baud_rate, propDelay, mtu);
```

```
// only keeps track activities from User_0
switch (i){
      case 0: trace_flag = true;
       break;

      case 1: trace_flag = false;
       break;

      default:
       System.out.println("Not a recognized test case.");
       break;
}

GridUser user = null;
String routerName = null;
RegionalGIS gis = null;

switch(i)
{
  case (0):
    user = GridUser(userName, link2,
              pollTime, glLength, glSize, glSize,
                 trace_flag);
    user.setGridletNumber(totalGridlet);
    linkNetwork(router0, user);
    routerName = router0.get_name();
    gis = (RegionalGIS) gisList.get(0);
    user.setRegionalGIS(gis);
    break;

  case (1):
    user = GridUser(userName, link2,
              pollTime, glLength, glSize, glSize,
                 trace_flag);
    user.setGridletNumber(totalGridlet);
    linkNetwork(router1, user);
    routerName = router1.get_name();
    gis = (RegionalGIS) gisList.get(1);
    user.setRegionalGIS(gis);
    break;

  default:
    System.out.println("Not a recognized test case.");
    break;
}
```

```
// [3] print some info messages

        System.out.println("Created " + resName + " with id = "
            +
                res.get_id() + ", linked to " + routerName +
                " and registered to " + gis.get_name() );
    }
```

In the second passage initially, we decide that we want to keep track of activities only from User_0, and then we proceed with the standard operations.
We inizialize the instace of the objects at null:
GridUser user = null;
String routerName = null;
RegionalGIS gis = null;
and then we use the *switch-case* structure.
For every resource we create a new *User* with all the data declared in the previous variables and we set the number of gridlets for each user. The value of the variable given into switch represents the current user and, in our example, we want to put *userO* in the *GIS0* and link it to *router0*, and put *user1* in the *GIS1* and link it to *router1*.
Finally, we print on the screen the result of the operations.

3.5.7 Result

What we have obtained using these particular changes to the code (in our example), is a simple network topology, which can be displayed in Figure 3.4.

The ovals represent the regional GIS (*RegionalGIS0* and *RegionalGIS1* respectively) that we have built and initialized in 3.5.4.
Inside the ovals we find:

- resources (respectively *Resource0* and *Resource1*) that we have implemented and initialized in 3.5.5.
- users (respectively *User0* and *User1*) that we have created and initialized in 3.5.6.
- routers (respectively *Router0* and *Router1*) that we have extracted from an external file in 3.5.3.

Finally the link between the routers was specified in an external file in 3.5.3.

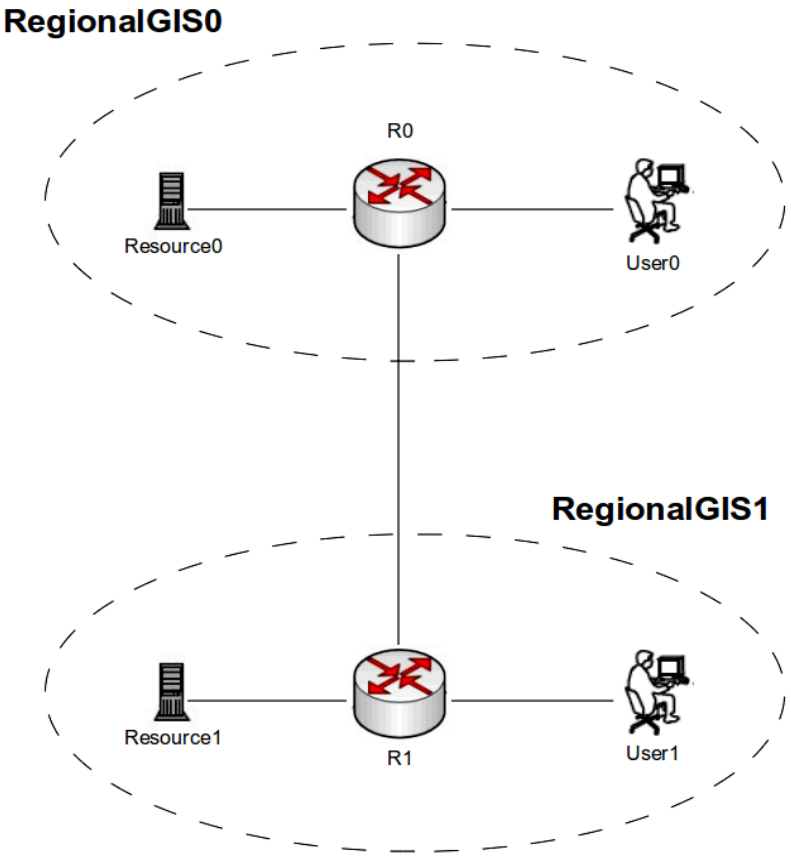


Figure 3.4: Diagram of a simple network topology

4 The Test Scenarios

4.1 The Scenario of EU DataGrid

We created an experiment based on EU DataGrid TestBed 1[12]. All data that we use for this scenario are shown by an example constructed from the University of Melbourne.[11]
In the Figure 4.1 we have reconstructed the network topology of the testbed.

DataGrid is a project funded by European Union. The objective is to build the next generation computing infrastructure providing intensive computation and analysis of shared large-scale databases, from hundreds of TeraBytes to PetaBytes, across widely distributed scientific communities.[13]

This phrase, quoted by the official website DataGrid Project, indicates the objective of the project.
It also indicates why we chose this scenario for our tests:

- it is a project founded and recognized by the European Union;
- it is a large-scale project;
- the aims to exchange a large amount of data;
- it tries to provide intensive computation.

All these features make this the ideal test bed for our tests.
In the LHC (Large Hadron Collider) experiment, for which the EU DataGrid has been constructed, most of the data is read-only. Therefore, to model a realistic experiment, we make these files to be read-only.
Furthermore, we assume the atomicity of the files, i.e. a file is a non-splittable unit of information, to simulate the already processed raw data of the LHC experiment.

We now describe the components of this scenario.

Resources. In the Table 4.1 reports all of the relevant information about resources.
In GridSim, total processing capability of a resource's CPU or CPU rating is modeled in the form of MIPS (Million Instructions Per Second) as devised by Standard Performance Evaluation Corporation (SPEC)[14].
The resource settings were obtained from the current characteristics of the real LHC testbed[15], except the number of users was slightly decreased to obtain a fast simulation.
We took the data about the resources and scaled them down. The computing capacities were scaled down by 10 and the storage capacities by 20.
The scaling was done primarily for one reason: the simulation of the real computing capacities is not possible because of memory limitation of the computer we ran the simulation on.

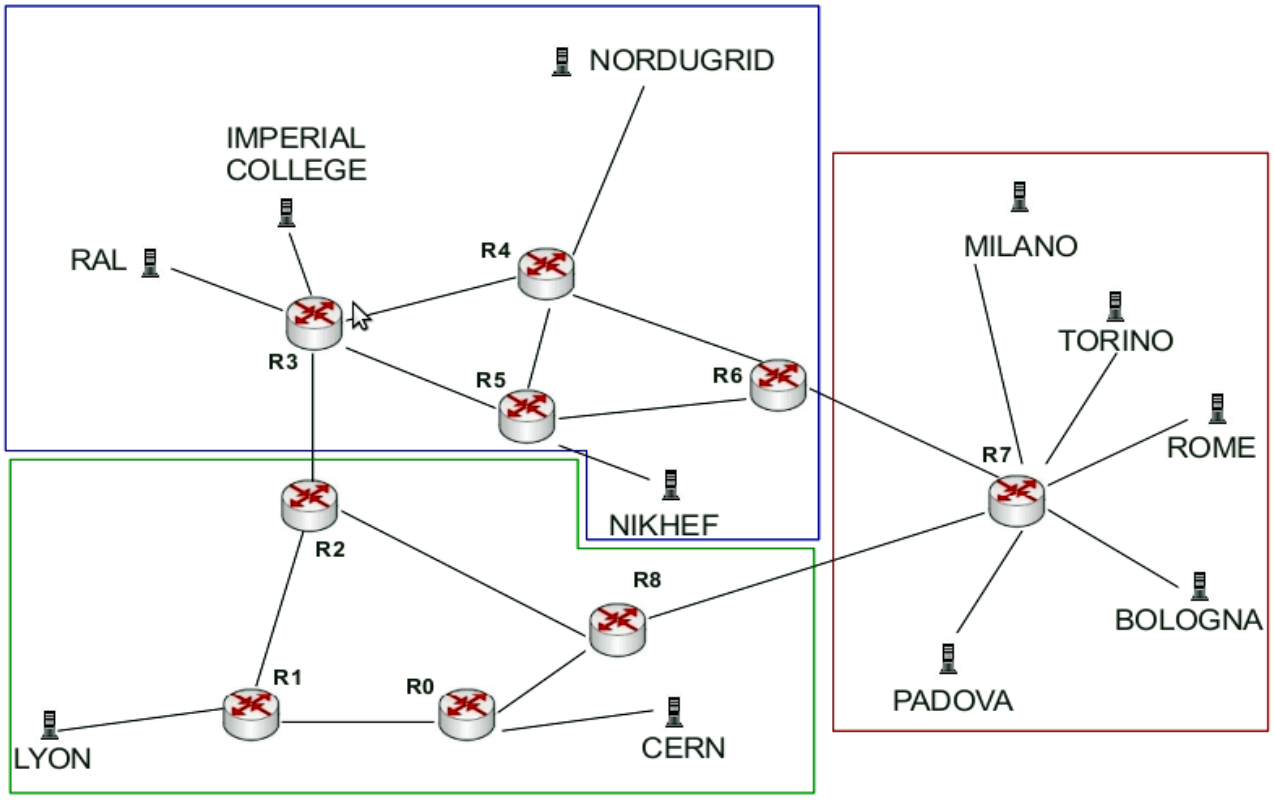


Figure 4.1: The simulated topology of EU DataGrid TestBed 1

The complete simulation would require more than 2 GB of memory.
Some parameters are identical for all network links, i.e. the Maximum Transmission Unit (MTU) is 1,500 bytes and the latency of links is 10 milliseconds.

Users. We simulated 100 users in total, where each resource is assigned a certain number of users as depicted in Table 4.1.
The users arrive with a Poisson distribution; four random users start to submit their jobs approximately every 5 minutes. Each user has between 20 and 40 jobs.

Gridlet. The data gridlet class represents a data intensive job.
As a result, each data-intensive job has a certain execution size (expressed in Millions Instructions - MI) and requires a list of files that are needed for execution.
This execution size (in MI) will be used by a resource to determine how much simulation time is required.

RegionalGIS. In the EU DataGrid TestBed 1 there are three regional GISs.
The first one, shown in Figure 4.1 with green, includes two resources (CERN and Lyon).
The second (the blue one) includes four resources (Ral, Imperial College, NorduGrid and NIKHEF).
The regional GIS shown in red is the last and it includes five resources (Milano, Rome, Torino, Bologna, Padova).

Table 4.1: Resource specifications of EU DataGrid TestBed 1

Resource Name (Location)	# Nodes	CPU Rating	# Users
RAL (UK)	41	49000	12
Imperial College (UK)	52	49000	16
NorduGrid (Norway)	17	49000	4
NIKHEF (Netherlands)	18	49000	8
Lyon (France)	12	49000	12
CERN (Switzerland)	59	49000	24
Milano (Italy)	5	49000	4
Torino (Italy)	2	49000	2
Rome (Italy)	5	49000	4
Padova (Italy)	1	49000	2
Bologna (Italy)	67	49000	12

Baud rate. As we mentioned in 3.5.3, we have to set the baud rate between the routers outside the source code.
For our goal we will change in some examples the values of the different baud rate (these changes are visible in Chapter 5 and Chapter 6).
Differently from what was showed in the 3.5.5, for this scenario we need to change the baud rate for every different resource.

In this scenario we assume that the CPU rating is the same for every processing element (PE), thus each machine has the same power.

4.2 The Scenario EU Artificial

For the second scenario, we created an experiment based on EU Artificial.
EU Artificial is a scenario of our own creation, larger than the scenario described above, which aims to offer a scenario for comparison with EU DataGrid TestBed 1. The Figure 4.2 shown the network topology of the EU Artificial.

The network topology is similar to the scenario of EU DataGrid. It has a branched structure that connects most of Europe.
As resources have been chosen some European metropolis.
The numbers of resources, routers, and regional GIS users are greater than the previous case: this means that the EU DataGrid scenario can be a model of comparison for a large scenario.

We now describe the components of this scenario.

Resources. Table 4.2 summarizes all the resources relevant information in the EU Arti-

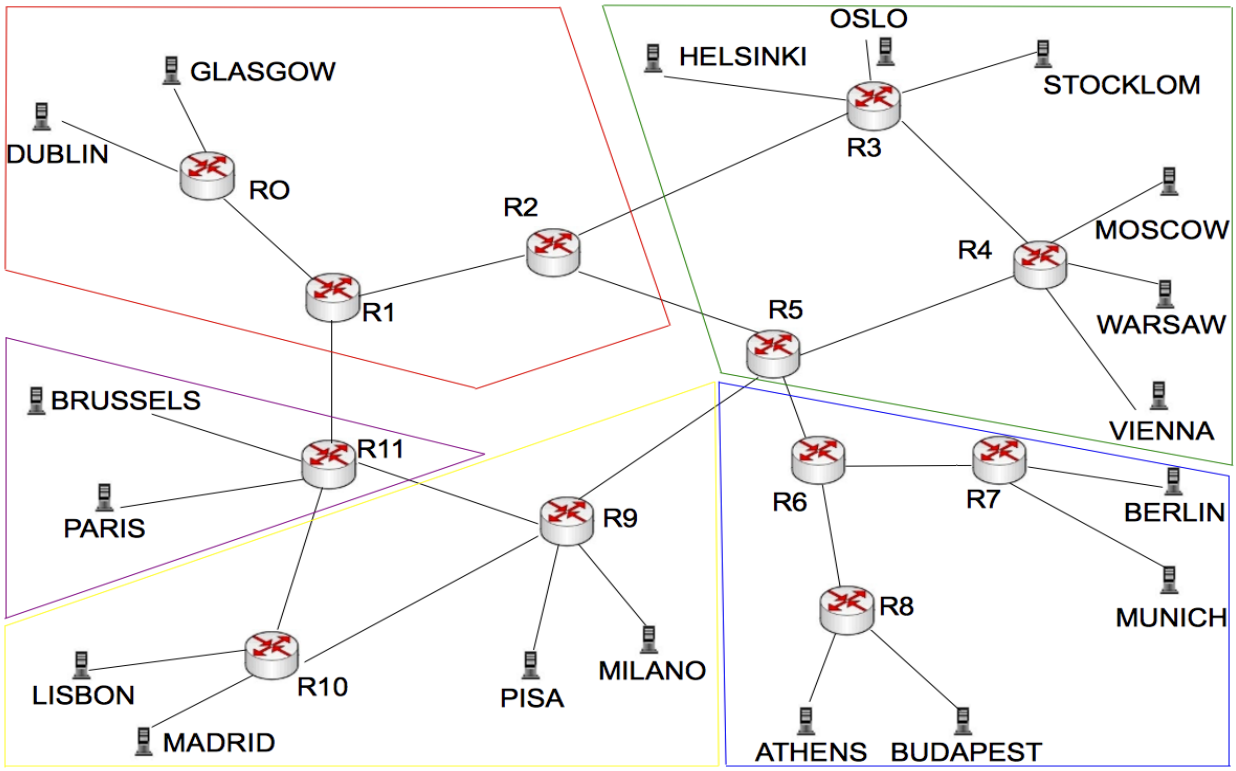


Figure 4.2: The simulated topology of EU Artificial

ficial.
As in the previous scenario, some parameters are identical for all network links, i.e. the Maximum Transmission Unit (MTU) is 1,500 bytes and the latency of links is 10 milliseconds.

Users. In this scenario we want to simulate 120 users in total, where each resource is assigned a certain number of users as depicted in Table 4.2.
As in the previous scenario, the users arrive with a Poisson distribution; four random users start to submit their jobs approximately every 5 minutes. Each user has between 20 and 40 jobs.

Gridlet. As already seen for the previous scenario, we implemented a DataGridlet class to represent a data-intensive job. As a result, each data-intensive job has a certain execution size (expressed in Millions Instructions - MI) and requires a list of files that are needed for execution. This execution size (in MI) will be used by a resource to determine how much simulation time is required.

RegionalGIS. In the EU Artificial scenario there are five regional GISs.
The first one, shown in Figure 4.2 with red, includes two resources (Dublin and Glasgow); the second (the green one) includes six resources (Helsinki, Oslo, Stockhlom, Moscow, Warsaw

Table 4.2: Resource specifications EU Artificial

Resource Name (Location)	# Nodes	CPU Rating	# Users
Dublin (Ireland)	35	49000	11
Glasgow (Scotland)	43	49000	9
Helsinki (Finland)	12	49000	4
Oslo (Norway)	15	49000	3
Stockhlom (Sweden)	8	49000	6
Moscow (Russia)	15	49000	4
Warsaw (Poland)	9	49000	3
Vienna (Austria)	2	49000	5
Berlin (Germany)	40	49000	6
Munich (Germany)	45	49000	13
Budapest (Hungary)	4	49000	5
Athens (Greece)	10	49000	6
Milano (Italy)	42	49000	11
Pisa (Italy)	38	49000	6
Madrid (Spain)	35	49000	7
Lisbon (Portugal)	20	49000	3
Paris (France)	7	49000	8
Brussels (Belgium)	8	49000	10

and Vienna), blu regional GIS consists of four resources (Berlin, Munich, Budapest and Athens), the yellow includes the regional GIS with four resources (Milano, Pisa, Madrid and Lisbon), and the regional GIS observed with the violet is the last and it includes two resources (Paris and Brussels).

Baud rate. The baud rate has the same explanation as in the previous.
We have to set the baud rate between the routers outside the source code. For our goal we will change in some examples the values of the different baud rates (these changes are visible in Chapter 5) and Chapter 6).

In this scenario, as in the previous, we assume that the CPU rating is the same for every processing element (PE), thus that each machine has the same power.
The number of the nodes (as we can see in the Table 4.2) was chosen on the basis of a hypothetical structure that sees some of the resources provided more machines than others.

Thus the main characteristics in common of the two scenarios are:

- both are located all over the Europe;
- both are divided into regionalGISs;
- both have a branched structure of the routers.

While the differences are focussed on the main level of GridSim toolkit:

- the number of the users is raised from 100 to 120;
- the number of the resources is raised from 11 to 18;
- the number of the regionalGISs is raised from 3 to 5.
- the number of the routers is raised from 8 to 12.

Moreover, as discussed in Chapter 5 and Chapter 6, there will be several tests that differentiate the two scenarios.

5 Results of Thesis - Scenario EU DataGrid TestBed 1

In this chapter we insert the most important results of some tests, of the scenarios EU DataGrid TestBed 1.
This network topology will be subjected to five different tests.

In the first test will use this scenario with the standard settings of the GridSim, thus we can have an initial operation of the toolkit.

In the second test, we will introduce a introduced a variant that will change the values of the different baud rates as specified in the original test bed.

Test number three introduces the data of the real scenario, the total number of gridlets and the number of machines available for each resource.

In test number four, we will change the values of Processing Elements (PEs) in order to assess the behavior of network topology with different computing powers.

The fifth and final test will be subject to changes in the length of gridelt, to study how GridSim behaves in different series.

5.1 Test One - The original settings

As the first test on the Scenario EU Datagrid TestBed1, we have implemented the network in which the baud rate between routers and between router and resource doesn't change.
In this manner we want to show how the grid simulation works before changing some data of the network topology.
The value of the baud rate between routers is 1 Gb/s, instead the value of the baud rate between routers and resources is 3,5 Gb/s. In this simulation test, we use two gridlets and only one machine per resource.
The first gridlet, *Gridlet_0*, was sent to different resources, while the second gridlet, *Gridlet_1*, was always sent to the same resource.

- From *User_0* to *User_9* the *Gridlet_0* was sent to resource *NORDUGRID*.
- From *User_10* to *User_81* the *Gridlet_0* was sent to resource *TORINO*.
- From *User_82* to *User_99* the *Gridlet_0* was sent to resource *PADOVA*.
- From *User_0* to *User_99* the *Gridlet_1* was sent to resource *NYKHEF*.

This arrangement allows us to evaluate four different user behaviors: in the first all users reside in the same resource and gridlets are sent to the same resource, so it is interesting to analyze the different response times; in the second, a large number of users residing on different resources, send their gridlets same resource; in the third a small number of users residing on different resources, sent their gridlets same resource; in last all users sent their second gridlet the same resource.

For the first ten users we have a foregone conclusion, in fact all the users are located in the resource *CERN* and all the *Gridlet_0* are send to *NORDUGRID* and all the *Gridlet_1* are send to *NYKHEF*; so in Figure 5.1 we obtained almost the same latency in both gridlets for all the users.

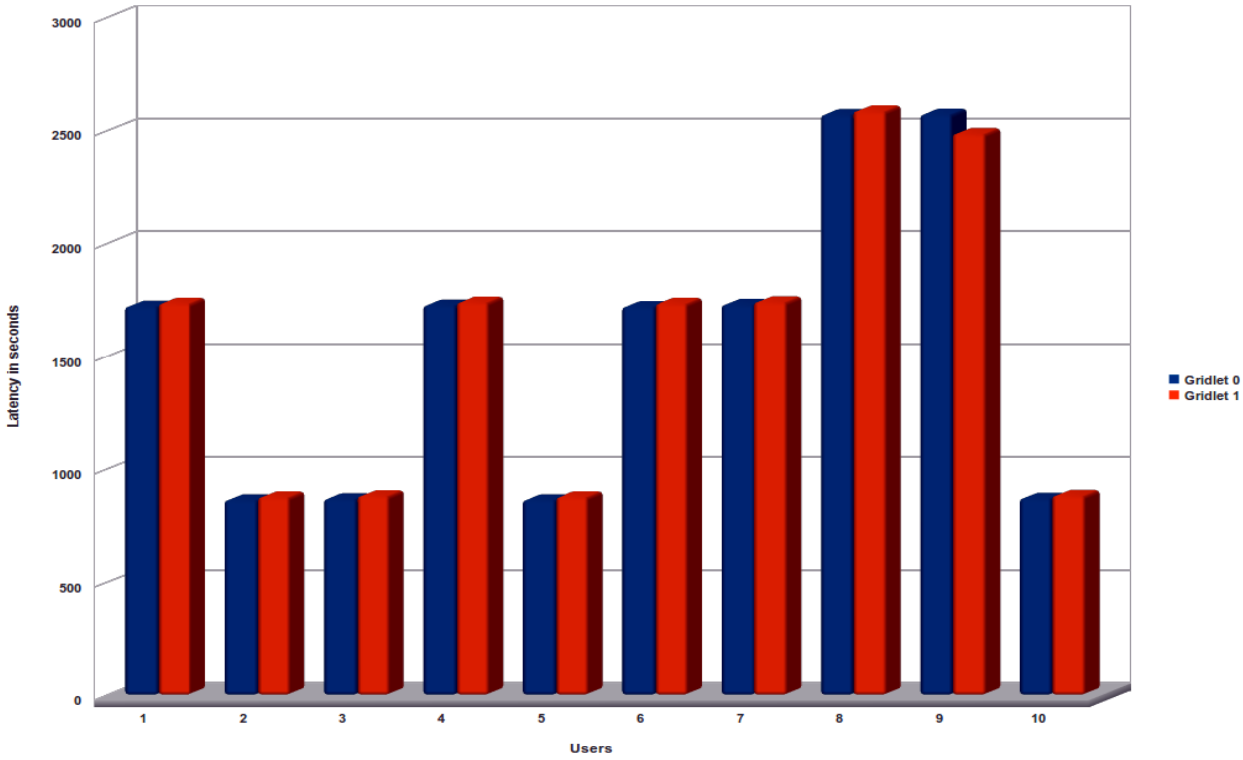


Figure 5.1: Test 1 - Performance of Latency - 10 Users

For the next 72 users, the destination of the first gridlet was changed to *TORINO* instead of *NORDUGRID*. But the essential difference is that now users are located in different resources and then the path to reach the resource *TORINO* can change by resource.
As we can see in Figure 5.2, the latency varies according to the starting point of gridlet. The latency of the *Gridlet_1* is low between the first and among the last members of the graph, this is because their routers are geographically close (especially the last six users who reside in the same router resource *TORINO*).
Instead, in the central part of the graph, we can see a gradual increase in both latencies due to a shift away from gridlet target.

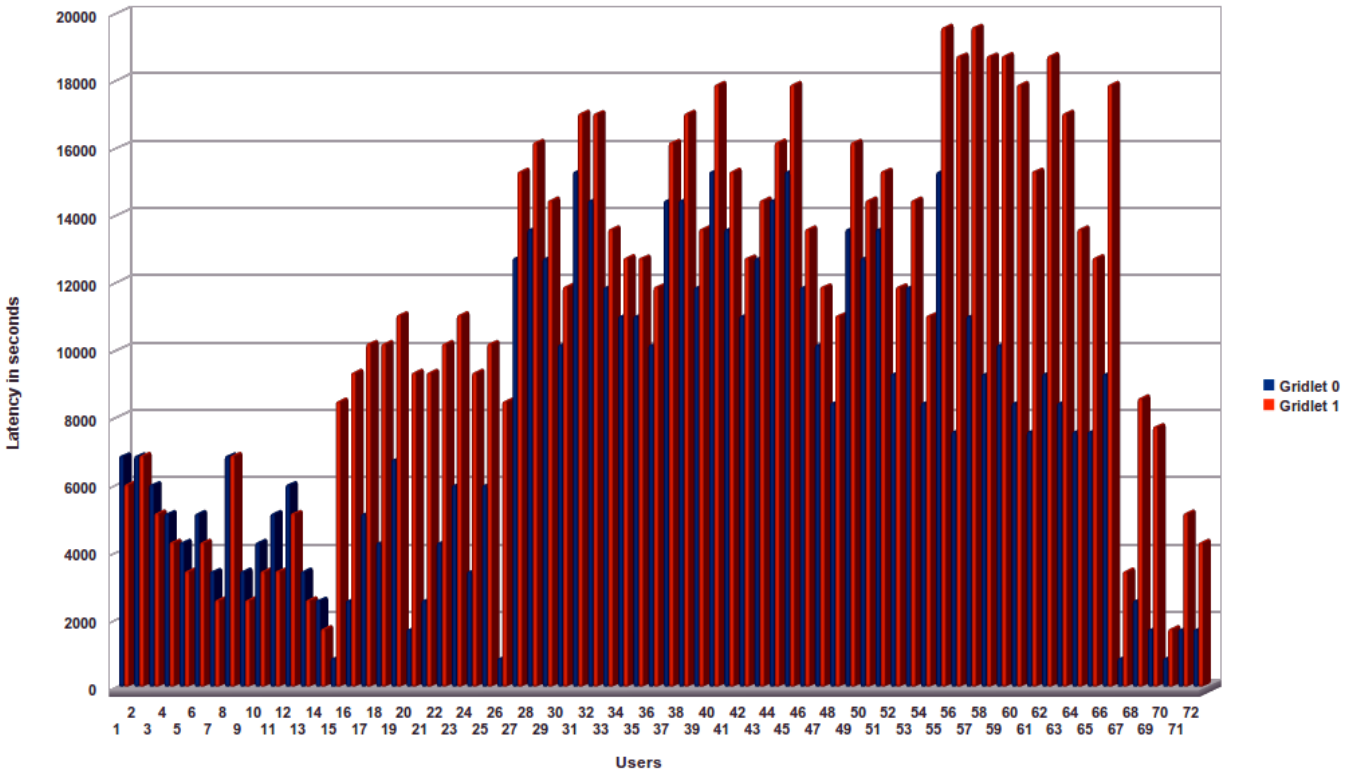


Figure 5.2: Test 1 - Performance of Latency - 72 Users

The last 18 users show us a scenario that we expected, in fact the destination of the first gridlet is now *PADOVA* and all the last users reside in the same router of this resource. So the latency of the *Gridlet_0* results low, while the value of the latency of the *Gridlet_1* tends to increase.
The Figure 5.3 shown these results.

5.2 Test Two - Introduction of different baud rate

With the second test, we have simulated the behaviour of the following case: only two gridlets (only two jobs) per every resource. But we have changed the values of the baud rates.
In Table 5.1 shows the baud rates that were indicated in the Scenario EU DataGrid TestBed 1.
Instead, the Table 5.2, represents the baud rates between routers and users.
Also in this test we assumed that the number of machines per each resource is one. To see the behavior of the network, and to make it as close as possible to reality, we have divided the destinations of the two gridlets.

The first gridlet, *Gridlet_0*, was sent to different resources, while the second gridlet, *Gridlet_1*, was always sent to the same resource.

- From *User_0* to *User_9* the *Gridlet_0* was sent to resource *CERN*.
- From *User_10* to *User_81* the *Gridlet_0* was sent to resource *NIKHEF*.

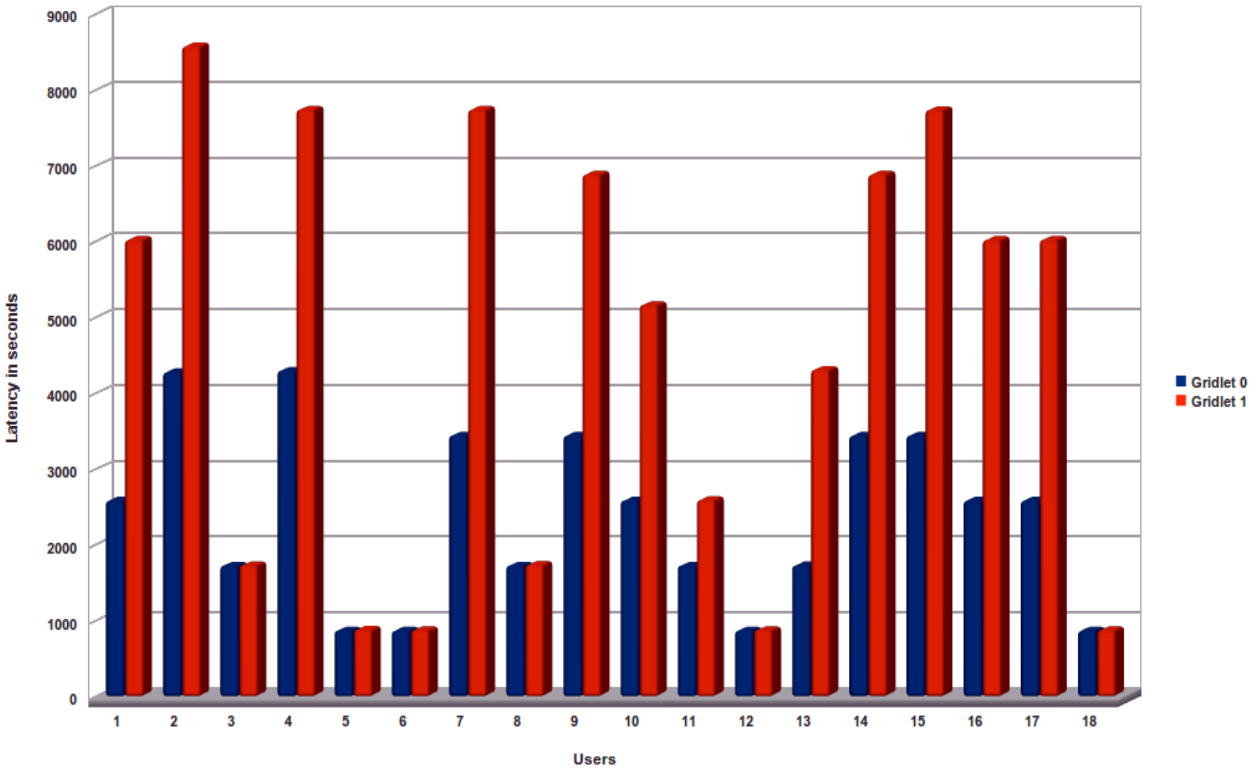


Figure 5.3: Test 1 - Performance of Latency - 18 Users

- From *User_82* to *User_99* the *Gridlet_0* was sent to resource *PADOVA*.
- From *User_0* to *User_99* the *Gridlet_1* was sent to resource *LYON*.

The Figure 5.4 shows the results from *User_0* to *User_9*.
All these users are part of the resource *CERN*, this means that the performance of the chart reflects the initial hypothesis (namely that the latency tends to increase with the number of gridelts that join the network).
In addition, in this graph, the *Gridlet_0* was always sent to resource *CERN*, and *Gridlet_1* was always sent to resource *LYON*.
Indeed, as we can see on Figure 4.1, the resource *CERN* and the resource *LYON* reside on two different routers (respectively *R0* and *R1*) and a greater latency in the case of a *Gridlet_1* is motivated by the fact that the *Gridlet_1* has a larger path to accomplish than *Gridlet_0*.

If we consider a greater number of users on multiple resources, the result is what appears in Figure 5.5. In this second part we analyzed users from *User_10* to *User_81*.
In general, both the latencies (of the *Gridlet_0* and *Gridlet_1*) tend to increase with an increase of users who join the network. But in this case the *Gridlet_0* was sent to different resource, unlike *Gridlet_1*, which is always sent to *LYON*. So the resource *LYON* has a high amount of work to do then the other resources and this justifies a higher latency.

Table 5.1: Router Baud Rates specifications EU DataGrid TestBed 1

First Router	Second Router	Baud Rate (Gb/s)
$Router_0$	$Router_1$	0.15
$Router_1$	$Router_2$	2.5
$Router_2$	$Router_3$	10
$Router_2$	$Router_8$	10
$Router_3$	$Router_4$	10
$Router_3$	$Router_5$	2.5
$Router_4$	$Router_5$	0.15
$Router_4$	$Router_6$	10
$Router_5$	$Router_6$	2.5
$Router_6$	$Router_7$	10
$Router_7$	$Router_8$	10
$Router_8$	$Router_1$	1

Table 5.2: Resource Baud Rates specifications EU DataGrid TestBed 1

Router	Resource	Baud Rate (Gb/s)
$Router_0$	CERN	1
$Router_1$	Lyon	2.5
$Router_3$	RAL	2.5
$Router_3$	Imperial College	2.5
$Router_4$	NorduGrid	0.15
$Router_5$	NIKHEF	0.62
$Router_7$	Milano	0.10
$Router_7$	Torino	0.05
$Router_7$	Roma	0.10
$Router_7$	Bologna	0.15
$Router_7$	Padova	0.05

The only few cases in which the latency decreases occur when the resource is sent and returned through the routers with a series of very high baud rates.

The final part is proof of the above, that the second gridlet has a higher amount of work

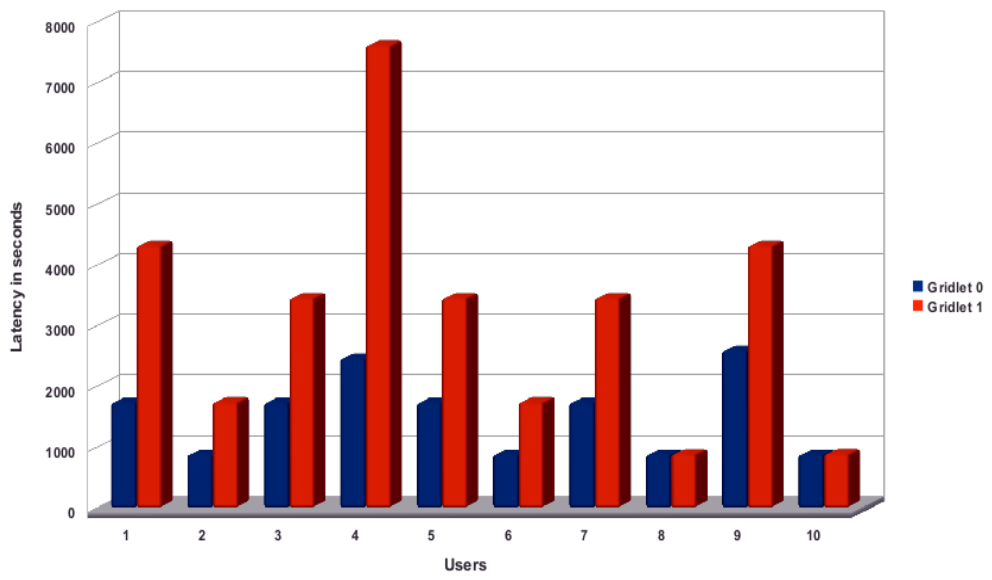


Figure 5.4: Test 2 -Performance of Latency - 10 Users

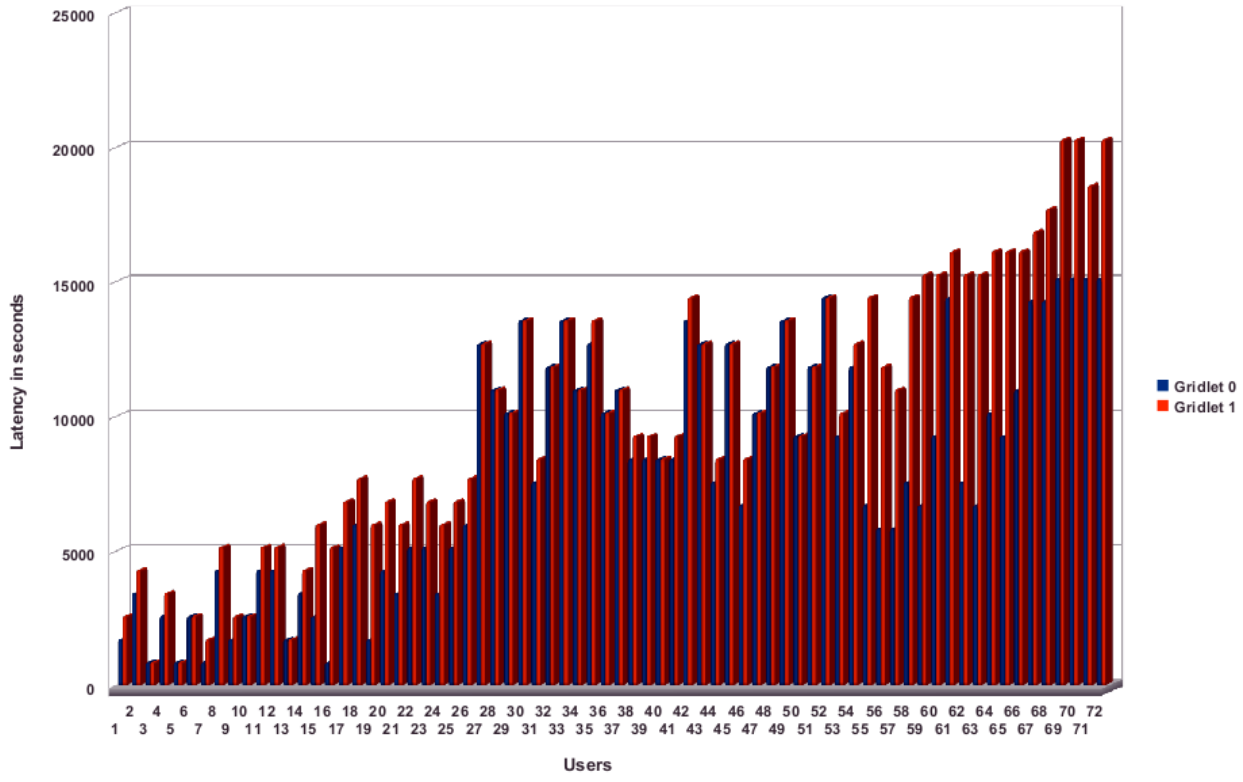


Figure 5.5: Test 2 - Performance of Latency - 72 Users

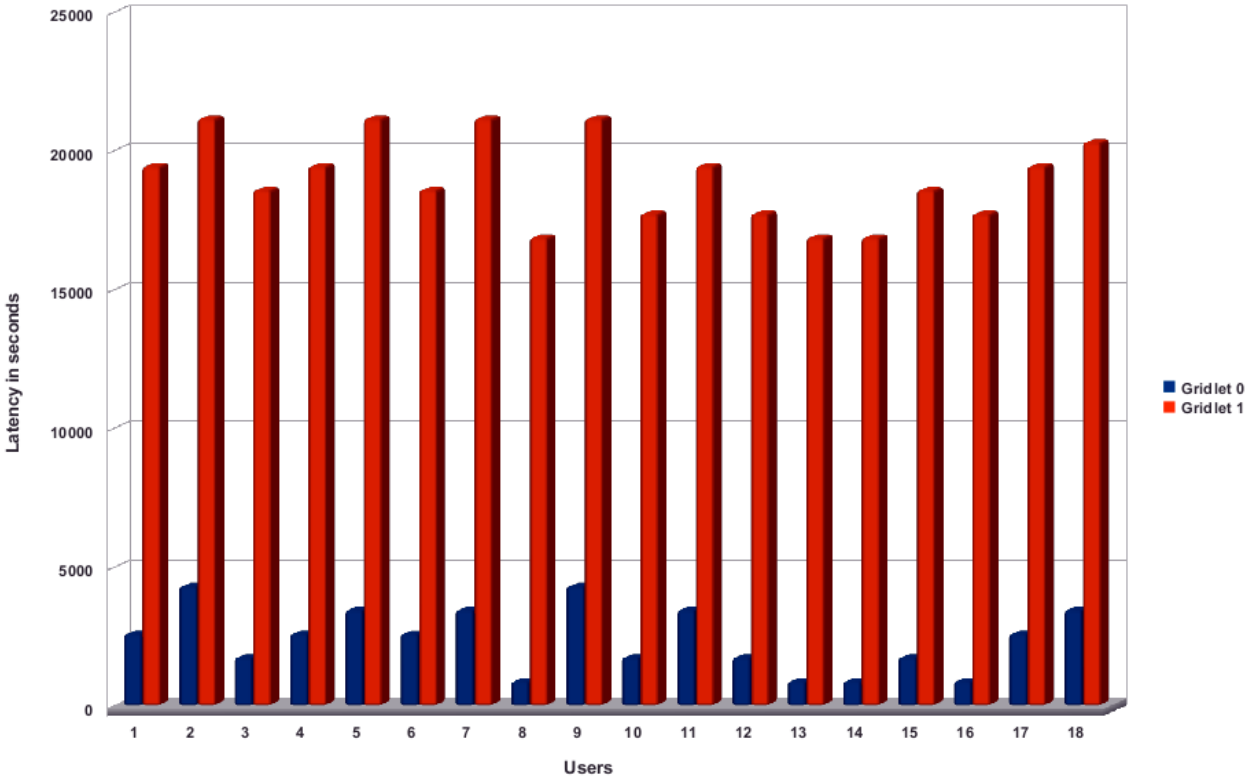


Figure 5.6: Test 2 - Performance of Latency - 18 Users

to do than the other resources, and in part it is already visible in Figure 5.4 with the final users. Indeed the Figure 5.6 shows the behavior of the last 18 users and we can see the huge difference between the latency of the first gridlet and the latency of the second gridlet.

5.2.1 Notes about the first two examples

In the first two examples we have given the network behavior in simple cases, with the aim to understand the evolution of the standard simulation. There would be other tests to be carried forward, based on the increase or decrease the baud rate between routers and between routers and resources.
As we shall see, in general, we found that with the decrease of the baud rate there is an increase in the latency, especially with regard to the second gridlet, while the increase in the baud rate causes a slight decrease in the level of latency.
These differences are in hundreds of seconds.
In these first two examples the number of gridlets is equal to two, and that is why we have analyzed the results of the first gridlet with different target resources and the second gridlet on a single target resource.
This allows us to get a visual of the results of the first gridlet in different circumstances, and a view of the second gridlet constant, and then compare them.

5.3 Test Three - The new settings

In this scenario we increase the values of two fundamental data:

- the number of the gridlets, from 2 to 5;
- the number of machines per resource, from 1 to 5.

Moreover gridlets have been redirected in the following order:

- From *User_0* to *User_24* the *Gridlet_0*, *Gridlet_1* and *Gridlet_2* were sent to resource *NIKHEF*.
- From *User_0* to *User_24* the *Gridlet_3*, *Gridlet_4* were sent to resource *PADOVA*.
- From *User_24* to *User_49* the *Gridlet_0*, *Gridlet_1* and *Gridlet_2* were sent to resource *RAL*.
- From *User_24* to *User_49* the *Gridlet_3*, *Gridlet_4* were sent to resource *TORINO*.
- From *User_50* to *User_74* the *Gridlet_0*, *Gridlet_1* and *Gridlet_2* were sent to resource *CERN*.
- From *User_50* to *User_74* the *Gridlet_3*, *Gridlet_4* were sent to resource *BOLOGNA*.
- From *User_75* to *User_99* the *Gridlet_0*, *Gridlet_1* and *Gridlet_2* were sent to resource *IMPERIAL COLLEGE*.
- From *User_75* to *User_99* the *Gridlet_3*, *Gridlet_4* were sent to resource *ROMA*.

Thanks to a substantial increase in the number of machines available (we have gone from 100 machines in the example two to 500 machines), the value of latency is drastically reduced. As shown in List of Figures 5.1, the latency of the *User_0* in the simulation with one machine ranges from about 2000 to about 10000 seconds.

Listing 5.1: Results of User 0 with one machine

```
    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    0       Success      38      2574     858   2574.02055733335
    ==================================================


    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    1       Success      43      2574     858   10297.8443013333
    ==================================================
```

Instead, the latency of the *User_0* in the simulation with five machines (and five gridlets), decreases due to the presence of a higher number of machines.
As shown in List of Figures 5.2, the latency of the *User_0* in the simultion with five machines ranges from about 800 to about 2000 seconds.

Listing 5.2: Results of User 0 with five machines

```
    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    0       Success      63      2574     858    869.41017999999
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    3       Success      88      2574     858    901.240719733294
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    1       Success      63      2574     858    1726.27018000013
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    4       Success      88      2574     858    1757.65071973346
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_0 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    2       Success      63      2574     858    2582.18022030973
    ==========================================================
```

In this context, we highlight a range of important information.

The gridlets are not executed in sequence, but based on the actual availability of the resource to which they are sent. In fact, if we check the List of Figures 5.2 after *Gridlet_0* is sent to the resource *NIKHEF* (resource id number 63), *Gridlet_3* comes into play (and not the *Gridlet_1*) but it is sent to the resource *PADOVA* (resource id number 88), because that is the first gridlet that can be sent to this resource.
This means that the resource *NIKHEF*, at the moment when it is requested by the *User_0*, is not available because it is probably occupied by other users.
In fact, the resource *NIKHEF* has five machines with which it performs the gridlet received. But there are 25 possible users that may require the execution of a gridlet. And so it's possible that the occurrence of a temporary unavailability of the resource cause of a queue of gridlets.
The large number of machines available for each resource allows the same resource to dispose of the work in short time, thus maintaining a level of very low latency (as you can see from the two previous listings).

In addition, this fact occurs with some regularity. We always take into account the List of Figures 5.2.
The third gridlet to be executed is *Gridlet_1* that is sent to the resource *NIKHEF*. Then the *Gridlet_4* to *PADOVA* and finally the *Gridlet_2* to *NIKHEF* again.
Even in these a queue of gridlets that has been created changes the gridlets' order of *User_0*.

Table 5.3: User 0 simulation times (in seconds)

Gridlet ID	Resource Target	Submission	Entry Queue	Exit Queue	Success
0	NIKHEF	557.361	/	/	1414.591
1	NIKHEF	571.711	571.711	1414.591	2271.841
2	NIKHEF	595.241	595.241	2271.841	3129.371
3	PADOVA	596.421	/	/	1453.801
4	PADOVA	610.571	610.571	1453.801	2311.071

Further evidence is given by the values of various times. *Gridlet_0* comes into action at second 557.361, the resource *NYKHEF* is free and completes the execution at time 1414.591. When *Gridlet_1* enters in execution time at second 571.711 the resource *NYKHEF* is occupied and it has to wait for the first gridlet to be executed (in this case *Gridlet_0* at second 1414.591).
Indeed *Gridlet_1* enters the queue at the same time as *Gridlet_0* when it comes into action (571.711), and it exits at second 1414.591 just when the *Gridlet_0* has finished its cycle, and it completes the execution at second 2271.841.
Similarly, this also happens to *Gridlet_2*. It enters at time 595.241 seconds, but the resource is still occupied and it enters the queue. It exits from the queue at time 2271.841 seconds just when the *Gridlet_1* has been executed and it completes its execution at time 3129.371. The *Gridlet_3* and the *Gridlet_4* behave the same way.
Gridlet_3 enters the execution time 596.421 when the resource *PADOVA* is free and it finishes its cycle at 1453.801.
Gridlet_4 comes into action at second 610.571 but the resource *PADOVA* results occupied ant it waits until the first gridlet in the queue of *PADOVA* has termined its execution (in this case *Gridlet_3* at second 1453.801).
We can also analyze this aspect by the values of latencies. *Gridlet_0* has a latency of about 870 seconds. *Gridlet_1* has a latency of about 1730 seconds and *Gridlet_2* has a latency of about 2580 seconds. This means that among the three gridlets that are sent to resource *NIKHEF*, there is a difference between latencies of about 850 seconds.
And if we consider the values of latency of *Gridlet_3*, about 900 seconds, that was performed after *Gridlet_0*, and *Gridlet_4*, about 1750 seconds, that was performed after *Gridlet_1*, it is clear that we have created a queue of gridlets:
User_0 after sending and receiving *Gridlet_0* from *NYKHEF*, tries to send *Gridlet_1* again to *NYKHEF* that results occupied. So, it sends the first gridlet available (*Gridlet_3*) for the resource *PADOVA*. After that it tries to send the second gridlet to *PADOVA*, but also in this resource there is a code queue of gridlets. Then it sends and receives *Gridlet_1* from *NYKHEF* because of the queue, 830 seconds pass after the notification from the notification of the previous gridlet. As one can see from the Table 5.3. As can be seen from the table, the same process occurs until the last gridlet.

Another example of the queue of gridlets is the *User_75*. As we can see from the List of Figures 5.3, the first gridlets processed are *Gridlet_3* and *Gridlet_4*.

This means that at first the *User_75* tried to send *Gridlet_0* to resource *IMPERIAL COLLEGE* (resource ID number 53), but it was occupied. So it sent and received the first gridlet available for the resource *ROMA* (resource id number 78), *Gridlet_3*, and after that the *Gridlet_4*. Finally he tried again to send the *Gridlet_0* and once confirmation is received, *Gridlet_1* and *Gridlet_2*.
Indeed if we check the value of the latency, we can see that the queue in *IMPERIAL COLLEGE* affects the value of latency.

Listing 5.3: Results of User 75 with five machines

```
    ========== OUTPUT for EUDataGrid_User_75 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    3       Success      78      2574     858   2218.06010359999
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_75 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    4       Success      78      2574     858   2217.79997880002
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_75 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    0       Success      53      2574     858   3319.76009879997
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_75 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    1       Success      53      2574     858   3314.98956066664
    ==========================================================

    ========== OUTPUT for EUDataGrid_User_75 ==========
Gridlet_ID  STATUS  Resource ID  Cost  CPU Time  Latency
    2       Success      53      2574     858   3309.16956066666
    ==========================================================
```

5.4 Test Four - Different processing elements

In this test we maintained the same characteristics as Test 3 (5.3) with the exception of the number of PEs. In fact, we decided to decrease the number of Processing Elements for each resource from four to one.

The Processing Elements (PEs) are very important because one or more PEs are put together to create a machine. So in this case the power of each machine is decreased to one fourth of its initial value.

What we expect is a fairly substantial increase in the value of latencies. We always take into account the *User_0*, as reported in the List of Figures 5.4.

Listing 5.4: Results of User 0 with five machines and one PE

```
    ================== OUTPUT for EUDataGrid_User_0 ==================
Gridlet_ID   STATUS   Resource ID   Cost   CPU Time      Latency
    3        Success       88        2574      858    8594.13641306668
    ==================================================================

    ================== OUTPUT for EUDataGrid_User_0 ==================
Gridlet_ID   STATUS   Resource ID   Cost   CPU Time      Latency
    4        Success       88        2574      858    8593.96486640003
    ==================================================================

    ================== OUTPUT for EUDataGrid_User_0 ==================
Gridlet_ID   STATUS   Resource ID   Cost   CPU Time      Latency
    0        Success       63        2574      858    12844.519593643
    ==================================================================

    ================== OUTPUT for EUDataGrid_User_0 ==================
Gridlet_ID   STATUS   Resource ID   Cost   CPU Time      Latency
    1        Success       63        2574      858    12844.5589989248
    ==================================================================

    ================== OUTPUT for EUDataGrid_User_0 ==================
Gridlet_ID   STATUS   Resource ID   Cost   CPU Time      Latency
    2        Success       63        2574      858    12844.7274522581
    ==================================================================
```

The differences between this Test and the results reported in List of Figures 5.2 are huge. The latency of the *User_0* in the simultion with one PE per machine, ranges from about 8000 to about 13000 seconds. In some girdlets, nearly ten times more.
This allows us to understand the importance of each machine and its components.
In addition, to reconnect to the considerations contained in Test 3, 5.3, we have another example of queue of gridlets.
Since there is only one PE for each machine, the response time of each resource is intended to increase compared to the previous examples and, consequently, the values of the latencies increase. If the response time of each resource increases, then it increases the possibility of queues of gridlets forming and also the size of the queues themselves.
The *User_0*, in the last example, finds a queue in the resource *NIKHEF* and it decides to send as first gridelet the *Gridlet_3* to the resource *PADOVA*, because compared to *NIKHEF* it was released first.

As a further test, we analyzed the network with a number of PEs equal to ten. The results are shown in Figure 5.7

In this figure the results of the latency (for the *User_0*) with 10 PEs per machine are put in comparison with the other two cases.

In the first case we show the trend of latency with a number of PEs per machine equal to four.

In the second case we consider the trend of latency with a number of PEs per machine

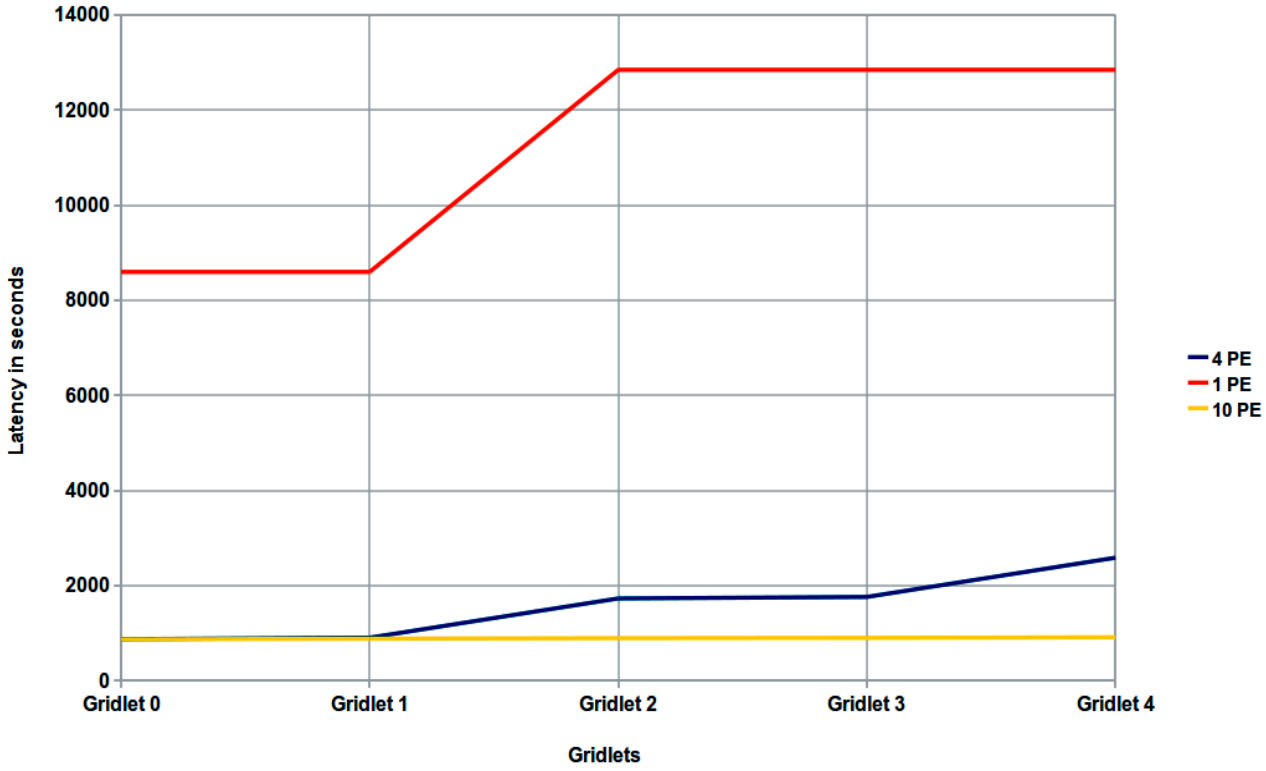


Figure 5.7: Test 4 - Performance of Latencies of User 0 with different PEs

equal to one.

And the last case is the new trend with a number of PEs per machine equal to ten.
With four PEs we have a trend that tends to increase with the number of gridlets that are processed. We observe that the value of latency is still quite low, this means that with four PEs, each machine can do its job fast enough.
As the number of gridlets that are entered by users on the network increases, there is a gradual increase in the value of latency.
With one PE the scenario of values is much higher, and again tends to increase with the number of gridlets. Obviously in the two previous cases, the value peaks based on the presence or absence of queues of gridlets.
Totally different is the result obtained with ten PEs. As shown in the Figure 5.7, the value of the latency remains constant, linear, for all gridlets that join the network. This means that with ten PEs in each machine, the latencies remain low and constant and the possibility of queue formation is decreased.
Of course the formation of queues can always happen, even with ten PEs, but with a very low percentage of probability.

Even for the *User_75*, that we have partially analyzed earlier, it behaves very similarly with ten PEs.
In the Figure 5.8 reports the behavior of the *User_75* with four, one and ten PEs, and the results are very similar to those of *User_0*.

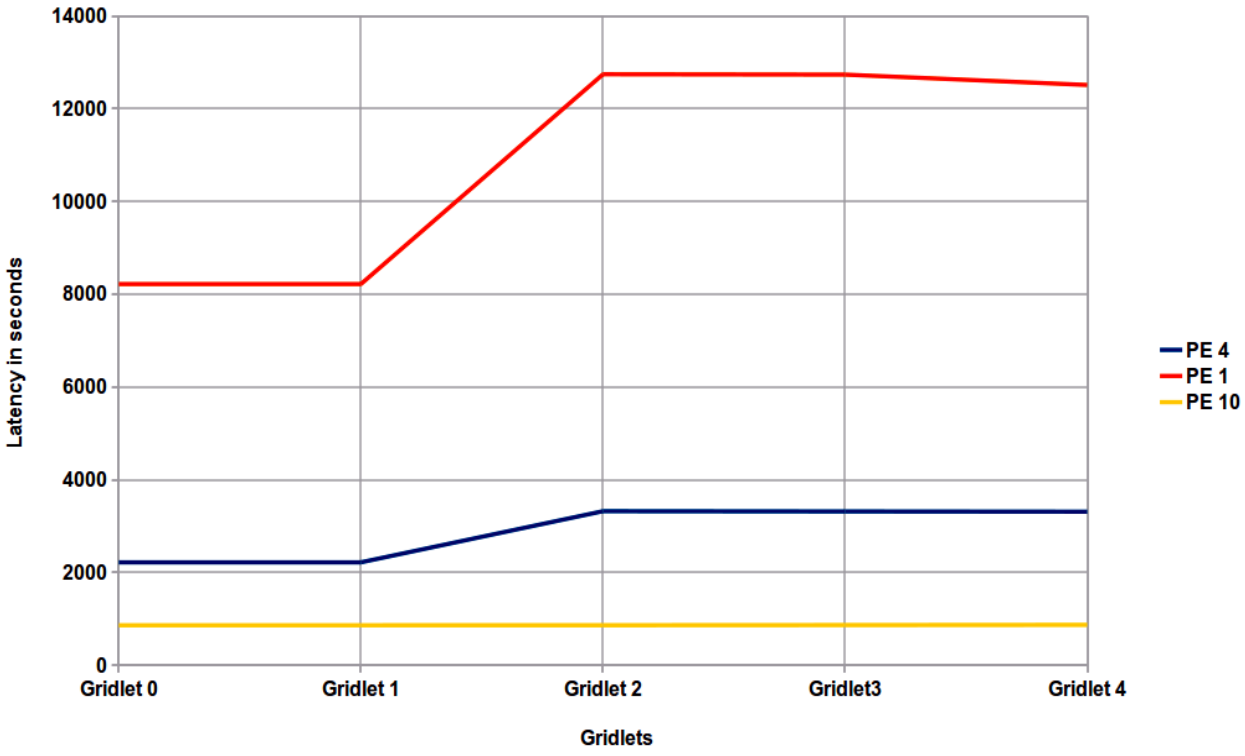


Figure 5.8: Test 4 - Performance of Latencies of User 75 with different PEs

5.5 Test Five - Different lenghts for the gridlets

In this Test we have modified the lenght of the gridlets.
The lenght is one of the important details of a gridlet and it is expressed in MI (Millions Instruction).
The standard value of the length of a gridlet, in fact is that which we have used in previous tests, is 42000000 MI. The goal of this test is to observe the behaviour of the simulation when increasing and decreasing this value.
For this test we analyze the value of the *User_53* with a length of the gridlets first equal to 42000000 MI, then increased by a factor of ten (420000000 MI) and finally decreased by a factor of ten (4200000).

In the Figure 5.9 shows the state of latency for the three different cases.

With a lenght equal to 42000000 MI we have a standard situation: with increasing number of gridlets, the latency also increases accordingly.

If we increase the length ten times (420000000 MI) we get a substantial increase in latency due to the difficulty of the resource to process the gridlet quickly.
The value of the latency ranges from about 8000 to about 25000 seconds, a remarkable difference. But if we compare this range of values with that of the standard situation, we note that the value of the latency is increased by a factor of ten, as well as the length value. Indeed the value of latency in the standard situation ranges from 800 to 2500 seconds.
In fact if you look closely at the trend of the graph, we note that the behavior in both cases

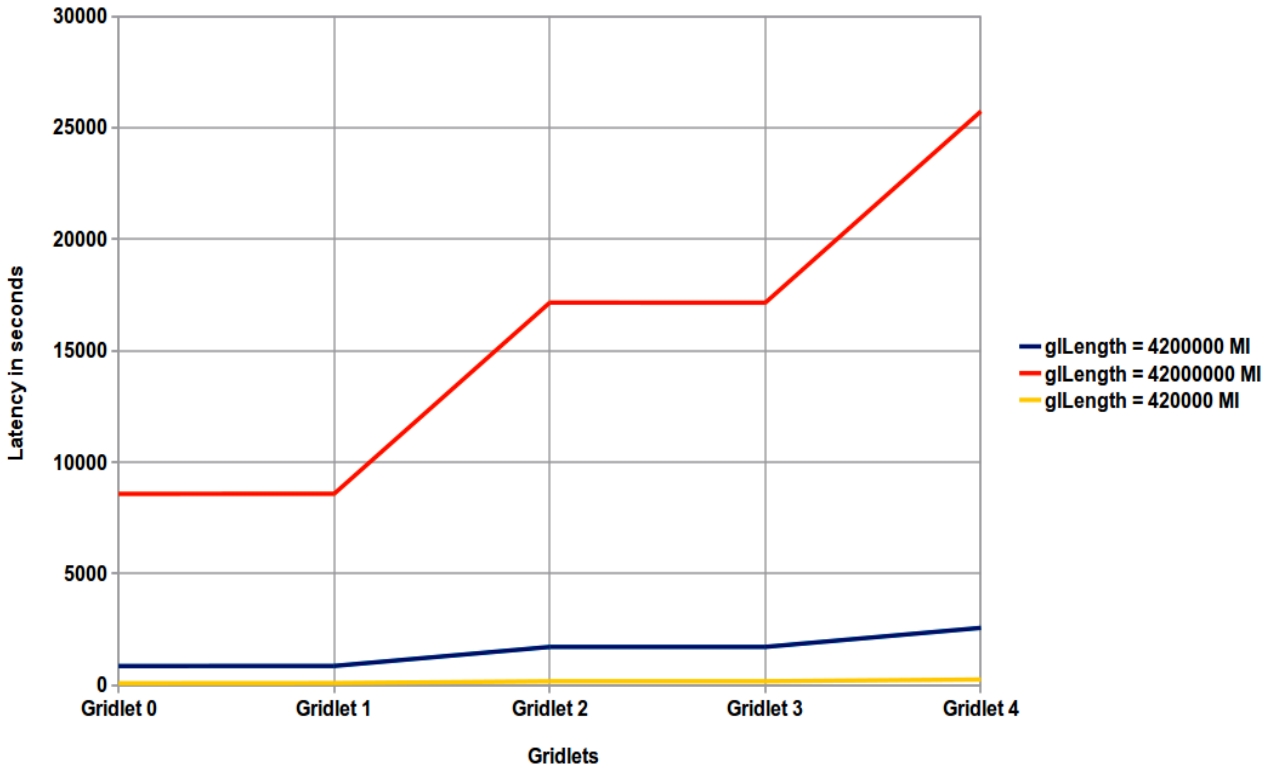


Figure 5.9: Test 5 - Performance of Latencies of User 53 with different lenghts of gridlets

is similar, only more pronounced when the length was increased.
Finally we see a very similar behavior when the length of gridelt was decreased by a factor of ten. Also in this case the value of latency range from 80 to 250, thus ten times less than the standard cases.
And also in this situation the graph shown the trend, but it is much less visible than the previous case because of the scale of the graph needed to show all three scenarios.

5.6 Comments of the results

The results obtained in this first scenario are the results that we could expect.
Indeed, we have prepared a toolkit to make it less casual and more programmable, and according to this by repeating the tests with different settings, we could get different results but find comparisons between them.
In particular:

- Test one: with the standard settings we have achieved results in line with expectations. In general, the level of latency tends to increase with the number of gridlets placed in the network.
  Having made the baud rate constant for all nodes, we can see that the geographical position of the various users and the allocation of gridlets onto the different resources play basic roles in determining the values of latency.

- Test two: the results are similar to those of test one, with some variations based on the various baud rate of the nodes.
  In fact there are now connections faster than others and this fact affects the latency, especially for the second gridlet which is always sent to the same resource.

- Test three: with an almost total change of the settings we get an almost total change in the levels of latency.
  More machines available means more efficiency in executing and disposing gridelts which means a lower level of latency.
  We can also observe that certain peak latencies are also due to the formation of traffic queues.

- Test four: these results allow us to understand the real contribution of the primary source of GridSim, i.e. computing power.
  Increasing the number of machines available may not be enough.
  In fact, by varying the number of processing elements we can observe how the power of every machine is relevant for an efficient network topology.

- Test five: in this test the results show us how important the length of the gridlet is that the user enters into the network.
  Entering a gridlet too long into the network slows the implementation process, as compared to a gridlet of standard length.
  When a gridlet is too short, is has the advantage of a faster execution but a disadvantage that it keeps part of the network busy just to perform a simple task.

6 Results of Thesis - Scenario Artificial EU

In this chapter we insert the most important results of some tests, of the scenario *Artificial EU*.
For this network topology, five tests have been implemented.

In the first test will use this scenario with the standard level of baud rate but with all our settings for the rest of the parameters.

Instead in the second test we will introduce a variant that will change the values of the different baud rates with our settings, and also will change the value of Maximum Transmission Unit (MTU).

Test number three will change the values of Processing Elements (PEs) and simultaneously the number of machines, in order to simulate the behavior of network topology with different computing powers.

Test number four will subject to changes in the length of gridlets, to study how GridSim behaves in different series.

In the fifth and final test, we will compare two different situations: the first with all the standard settings of our network, the second with all gridlets sent to the resource *Munich*.

The map of this scenario is shown in Figure 6.1.

6.1 Test One - Standard baud rate

In the first test of this scenario we focus our attention immediately on the large-scale example with, unlike the previous scenario, the following simulation data:

- the number of total machines for each resource is 6;
- the number of total gridlets for each user user is 10.

And the gridlets have been redirected in the following order for each users.
From *User_0* to *User_19*:

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *PARIS*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *WARSAW*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *MADRID*.

From *User_20* to *User_39*:

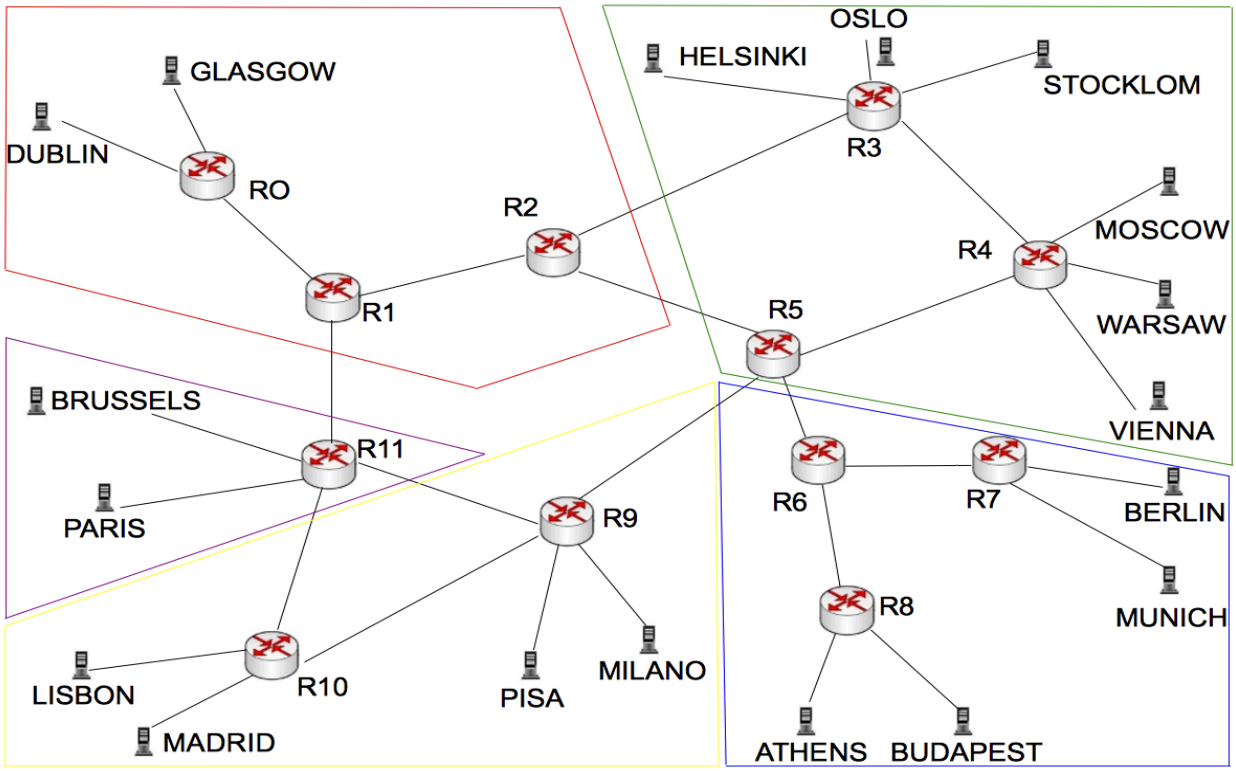


Figure 6.1: The simulated topology of EU Artificial

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *MUNICH*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *MOSCOW*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *BRUSSELS*.

From *User_40* to *User_59*:

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *BERLIN*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *STOCKHLOM*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *MILANO*.

From *User_60* to *User_79*:

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *VIENNA*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *OSLO*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *ATHENS*.

From *User_80* to *User_99*:

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *GLASGOW*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *LISBON*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *DUBLIN*.

From *User_100* to *User_119*:

- *Gridlet_0*, *Gridlet_1*, *Gridlet_2* and *Gridlet_3* were sent to resource *HELSINKI*.
- *Gridlet_4*, *Gridlet_5* and *Gridlet_6* were sent to resource *BUDAPEST*.
- *Gridlet_7*, *Gridlet_8* and *Gridlet_9* were sent to resource *PISA*.

We have implemented the network in which the baud rate between routers and between router and resource doesn't change.
The value of the baud rate between routers is 1 Gb/s, instead the value of the baud rate between routers and resources is 3,5 Gb/s.
The Figure 6.2 represents the behaviour of the network for the first four users. As shown in the graph, the results are influenced by the formation of queues of gridlets.

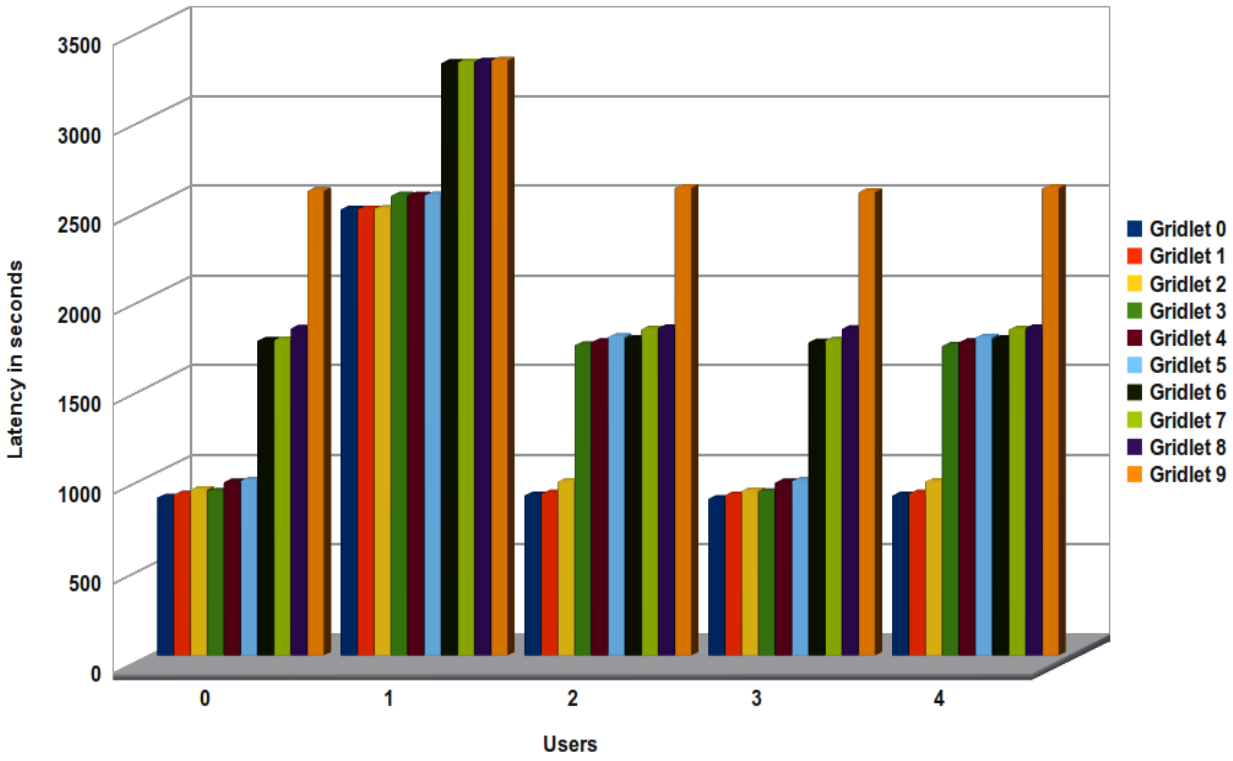


Figure 6.2: Test 1 - Performance of Latencies of four users

User_0 has a constant rate for the first four gridlets, gridlets because it sent *Gridlet_0* as the first gridlet to the resource *PARIS*, but after that this resource resulted occupied and *Gridlet_1* is left in a queue.
Thus *Gridlet_4* (the first gridlet for the resource *WARSAW*) starts its extension, but then, like before, also *WARSAW* resulted occupied and *Gridlet_5* is also left in a queue.

Listing 6.1: Timing of Gridlet received by User 0

```
ArtificialEU_User_0: Receiving Gridlet #0 with status Success at
   time = 1431.5901839999767 from resource ArtificialEU__Paris
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #4 with status Success at
   time = 1461.4202319999495 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #1 with status Success at
   time = 1478.2001828570772 from resource ArtificialEU__Paris
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #5 with status Success at
   time = 1480.5002239999321 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #7 with status Success at
   time = 1540.6002091427347 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #8 with status Success at
   time = 1551.320209142725 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #2 with status Success at
   time = 2311.780227999988 from resource ArtificialEU__Paris
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #6 with status Success at
   time = 2328.790227999957 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #9 with status Success at
   time = 2403.9602091429283 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_0: Receiving Gridlet #3 with status Success at
   time = 3150.43018399999 from resource ArtificialEU__Paris
```

In this case, both the queues are released quickly and *Gridlet_1* and *Gridlet_5* have been performed. This explains the similar values of the first four gridlets.
But given the large number of gridlets on the net, both the first two resources are returned as occupied and the two gridlets that are to be carried out subsequently are the *Gridlet_7* and *Gridlet_8*, namely the first two gridlets of the resource *MADRID*, and in the graph they are represented from the subsequent two values slightly higher.
The next to be performed are the gridlets that were previously queued in each resource:

Gridlet_2, *Gridlet_6* and *Gridlet_9*, and therefore have higher latency values than previous gridlets.
The last to be performed is *Gridlet_3* which has undergone a queue longer than all previous gridlets.
The List of Figures 6.1 shows these results.

On the contrary, *User_1* has a completely different pattern of latency. This is because the gridlets immediately encounter a series of queues of gridlets, which tend to increase their latencies.
In the Figure 6.2 we can see that the first three gridlets have almost the same (and high) latency. This is because they are performed in the sequence: *Gridlet_4*, *Gridlet_5* and *Gridlet_6*, all of them addressed in *WARSAW*, because in resource *PARIS* there is a queue that prevents the execution of the first four gridlets.

Listing 6.2: Timing of Gridlet received by User 1

```
ArtificialEU_User_1: Receiving Gridlet #4 with status Success at
   time = 3177.4202319999767 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #5 with status Success at
   time = 3178.940233142867 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #6 with status Success at
   time = 3181.2902279999616 from resource ArtificialEU__Warsaw
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #7 with status Success at
   time = 3256.6002091428113 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #8 with status Success at
   time = 3257.9402091428406 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #9 with status Success at
   time = 3259.28020914287 from resource ArtificialEU__Madrid
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #0 with status Success at
   time = 3989.320228000048 from resource ArtificialEU__Paris
-------------------------------------------------------------------

ArtificialEU_User_1: Receiving Gridlet #1 with status Success at
   time = 3993.990184000003 from resource ArtificialEU__Paris
-------------------------------------------------------------------
```

```
ArtificialEU_User_1: Receiving Gridlet #2 with status Success at
   time = 3999.970183999997 from resource ArtificialEU__Paris
```

```
ArtificialEU_User_1: Receiving Gridlet #3 with status Success at
   time = 4005.5901839999924 from resource ArtificialEU__Paris
```

Indeed as we can foresee from List of Figures 6.2, after these, the three resources of *MADRID* are performed, and lastly, after the queue has been disposed, the four resources of *PARIS* are performed.
This is a clear example of how the formation of code causes a total change in the performance of various gridlets.

The *User_2* has a different behavior than the two previous users, but always linked to the formation of queues.
The first three gridlets performed are the first gridlets for each resource, namely *Gridlet_0*, *Gridlet_4*, *Gridlet_7*.
It is now quite clear that this order is related to the formation of queues: before in the resource *PARIS*, then in *WARSAW* and finally in *MADRID*.
After that, based upon various queues in the various resources, six gridlets are performed in series, and lastly, the *Gridlet_7* from the resource *PARIS*.

It is easy to see from the Figure 6.2, the *User_3* has a behavior very similar to the *User_0*, while the *User_4* has a pattern which mirrors the *User_2*.
In this test the behavior of other users is very close to those of the first four users that we analyzed.

6.2 Test Two - Different MTUs

In this test we have changed the value of MTU. The MTU is the maximum transmission unit and it is expressed in bytes.
We also have different values of the baud rate, between routers and between routers and resources. In Table 6.1 shows the baud rates that were indicated in the Scenario EU Artificial. Instead, in the Table 6.2, represents the baud rates between routers and users.

The values of the different baud rates between routers were chosen based on geographic location: the closer the routers, the higher the transmission rate.
Instead the values of the different baud rates between routers and resources have been determined on the basis of our assessment: some of the most important centers such as Munich, Milano, Dublin, Berlin, Pisa, Madrid and Glasgow have baud rates higher than other resources.

We decided to analyze the situation of the *User_10*. This is because all users in this test have a similar behavior. Indeed the behavior of the ten gridlets is not affected by the specific terms of latency.
The Figure 6.3 shows that the levels of latencies are very similar and this is due to an important special feature of the toolkit GridSim: the packets which are larger than the

Table 6.1: Router Baud Rate specifications EU Artificial

First Router	Second Router	Baud Rate (Gb/s)
$Router_0$	$Router_1$	5
$Router_1$	$Router_2$	1
$Router_2$	$Router_3$	2.5
$Router_2$	$Router_5$	2.5
$Router_3$	$Router_4$	7
$Router_4$	$Router_5$	0.55
$Router_5$	$Router_6$	2
$Router_5$	$Router_9$	0.5
$Router_6$	$Router_7$	1.5
$Router_6$	$Router_8$	1
$Router_9$	$Router_{10}$	1.25
$Router_9$	$Router_{11}$	5.5
$Router_{10}$	$Router_{11}$	7.5
$Router_{11}$	$Router_1$	1.5

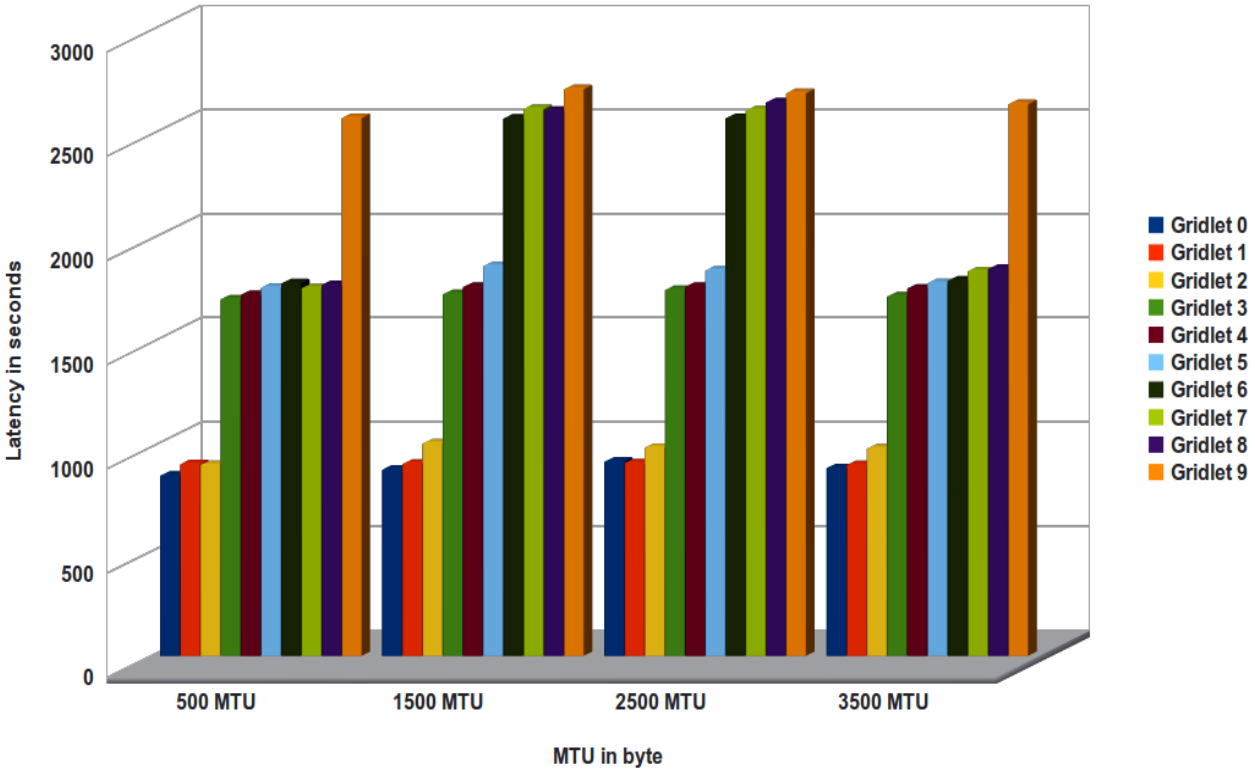


Figure 6.3: Test 2 - Performance of Latencies of Users 10 with different MTU's levels

MTU should be split up into MTU size units. When GridSim entities want to send a data over the network, each of them has Input and Output (I/O) entities attached to it.

The Output entity is responsible for splitting the data into MTU sized packets, whereas the Input entity is accountable to collate the different packets in a stream all together, and send them a piece of data to the GridSim entity.
In addition, these I/O entities act as a buffer to hold the packets until a link is free.
A network packet in GridSim is represented as an interface class *Packet*. Currently, there are two classes that belong to this category, i.e. *NetPacket* and *InfoPacket*.
A *NetPacket* class is used to encapsulate data passing through the network, whereas class *InfoPacket* is devoted to gathering network information during runtime which is equivalent to Internet Control Message Protocol (ICMP) in physical networks.
A *Packet Scheduler* is responsible for deciding the order in which one or more packets will be sent downlink.
The ability to split data into packets avoids overloading the network.

6.3 Test Three - Change the calculation power

The third test of this scenario allows us to evaluate the progress of gridlets varying simultaneously the number of machines for each resource and the number of processing elements for each machine.
Our goal is to compare five different cases:

Table 6.2: Resource Baud Rate specifications EU Artificial

Router	Resource	Baud Rate (Gb/s)
$Router_0$	Dublin	1
$Router_0$	Glasgow	2.5
$Router_3$	Helsinki	0.5
$Router_3$	Oslo	0.6
$Router_3$	Stockhlom	0.15
$Router_4$	Moscow	0.5
$Router_4$	Warsaw	0.10
$Router_4$	Vienna	0.05
$Router_7$	Berlin	1.10
$Router_7$	Munich	2.15
$Router_8$	Budapest	0.05
$Router_8$	Athens	0.15
$Router_9$	Milano	1.25
$Router_9$	Pisa	1.25
$Router_{10}$	Madrid	0.95
$Router_{10}$	Lisbon	0.55
$Router_{11}$	Paris	0.05
$Router_{12}$	Brussels	0.05

- The standard case: *Number of Machines* = 6 and *Number of PEs* = 4;
- The first case: *Number of Machines* = 12 and *Number of PEs* = 8;
- The second case: *Number of Machines* = 12 and *Number of PEs* = 2;
- The third case: *Number of Machines* = 3 and *Number of PEs* = 8;
- The fourth case: *Number of Machines* = 3 and *Number of PEs* = 2;

Given the large amount of data, we decided to focus our analysis on a single user and on three particular gridlets. We will evaluate the behavior of the *User_75* on *Gridlet_0*, on *Gridlet_5* and on *Gridlet_9*.

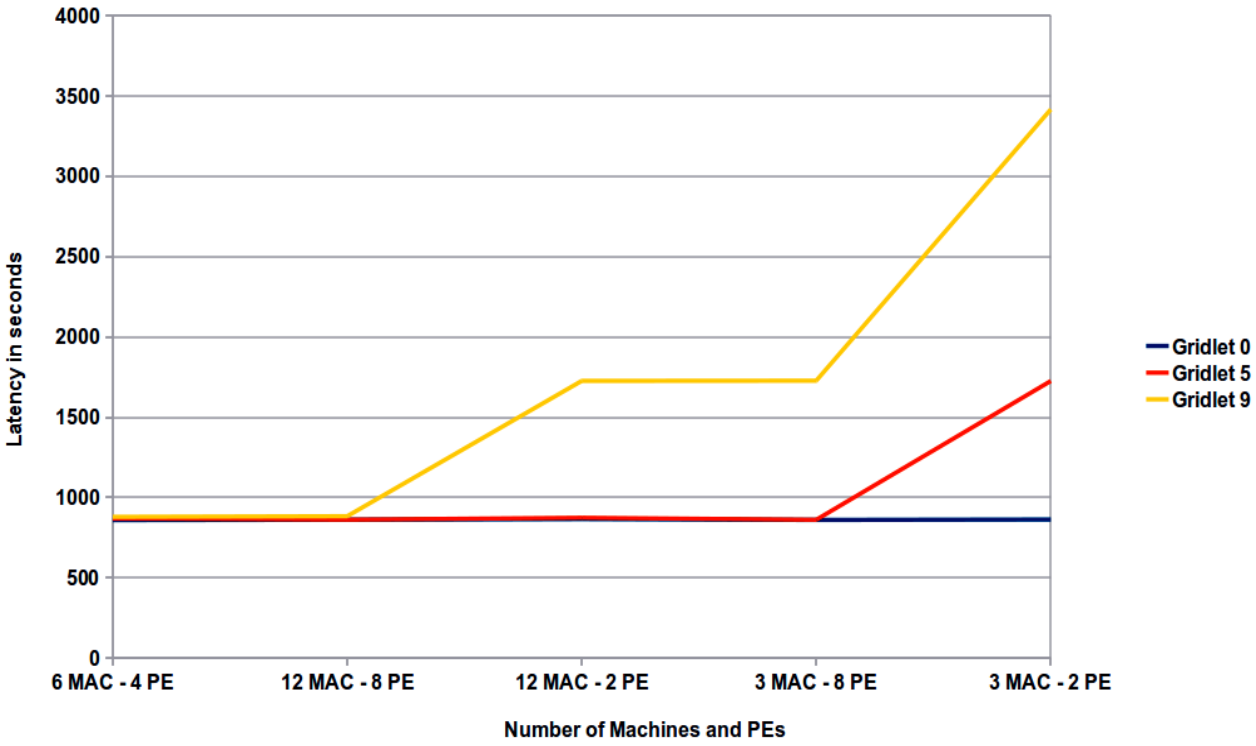


Figure 6.4: Test 3 - Performance of Latencies of Users 75 with different number of machines and PEs

The first consideration is the latency of the three gridlets in the five cases described above.

As shown in the Figure 6.4, the *Gridlet_0* has a linear trend in each of five cases. This is beacause the first gridlet doesn't meet traffic and then, regardless of the number of machines and PEs, should not be an impact on latency.

Gridlet_5 has a similar pattern to *Gridlet_0* until the fourth case, with three machines and eight PEs. In fact, while in the first four cases the *Gridlet_5* encounters little traffic, in the fifth case, a probable formation of a code makes it impossible to have a quick execution

of the gridlet, in a situation of limited computing power.

With *Gridlet_9* we have a clear view of the situation, because being the last gridlet, it will surely be subject to delays due to traffic.
The level of latency remains constant for the first two cases. This is because the simulation of this network is optimal with six machines and eight PEs, and then doubling the computing power does not affect the latency.
But as the number of machines and/or PEs begins to decrease, we can see that the level of latency increases, because of the traffic on the network, and consequently, increases the difficulty for the resources to perform a gridlet, especially when the number of machines and PEs is idle.
We can also note that doubling the number of machines and halve the number of PEs, leads to the same level of latency compared and to halving the number of machines and doubling the number of PEs.

The second consideration is the time when a gridlet is sent running by a user. The behavior of the three gridlets is similar to the previous case.

As we can see from the Figure 6.5, *Gridlet_0* always has a linear trend. This is because the time that it comes into action, as well as the latency, is not affected by the change in computing power in the event of low traffic or no traffic.

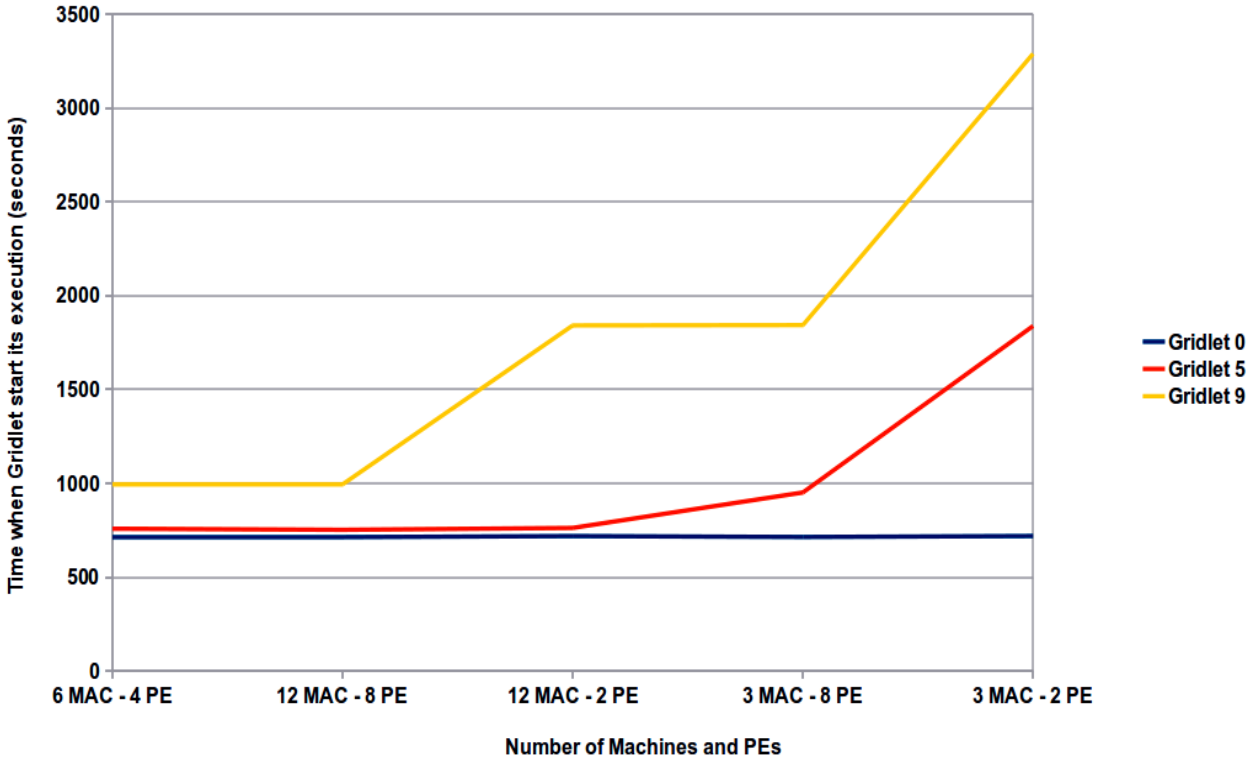


Figure 6.5: Test 3 - Performance of Execution Time of Users 75 with different number of machines and PEs

With *Gridlet_5*, unlike what happened with the latency, the level of this time remains constant until the third case, then increased slightly in the fourth case, and more prominently in the fifth.
This happens because the resource has fewer machines available (from six/twelve to three) and several more requests to perform than it did with the *Gridlet_0* and thus takes longer to initialize the process, especially when the number of PEs decreases.
As mentioned above, the last gridlet is particularly affected by traffic on the network, and then at the slightest change in computing power, time starts to increase and reaches its highest peak when computing power is at a minimum.
In *Gridlet_9*, with regard to latency, we can note two important aspects. The first one is that the level of latency remains constant for the first two cases. The simulation of this network results optimally with six machines and eight PEs, and then doubling the computing power does not affect the time.
The second one regards the third and the fourth case: halving the number of PEs leads to the same amount of time as to halving the number of machines and doubling the number of PEs.

The third and last consideration that we have taken into account, affects the time when the gridlet finishes its execution with success.
The variation of the chart for all three gridlets is similar. The only thing that changes is obviously the range of time.

The Figure 6.6 highlights this fact.
For the *Gridlet_0* we see a more linear variation, while for the time when the gridlets came running, the range was around 700 seconds, the time when the gridlet ends its execution is around 1500 seconds.
This means that for all the five cases, the gridlet takes about 800 seconds to be executed.
For *Gridlet_5*, as we have already seen before, the time starts to increase when the number of machine decreases. We pass from a range of between 700 and 1800 seconds in the event that the gridlet comes into action, to a range of between 1600 and 2600 seconds in cases where the gridlet ends its successful execution.
This means that in the cases with the gridlet comes into action the difference is around 1100 seconds, while in the cases when the gridlet ends its successful execution the difference is around 1000 seconds.
The reduction of the difference in the range of time can be explained by a partial decrease in traffic when the resource is about to end the execution of *Gridlet_5*.
Being the fifth gridelt, i.e. the middle gridlet, the outgoing traffic from a resource begins to decline slowly.

In fact, the *Gridlet_9*, the last gridlet, testifies to this. We pass from a range of between 900 and 3200 seconds in the event when the gridlet comes into action to a range of between 2000 and 4000 seconds in the case when the gridlet ends its successful execution.
In the case when the gridlet comes into action the difference is around 2300 seconds, while in the case with the gridlet ends its successful execution the difference is around 2000 seconds.
A witness to what has been said before, the outgoing traffic is decreasing and therefore the difference in the time of order execution is less than the difference in time of entry.

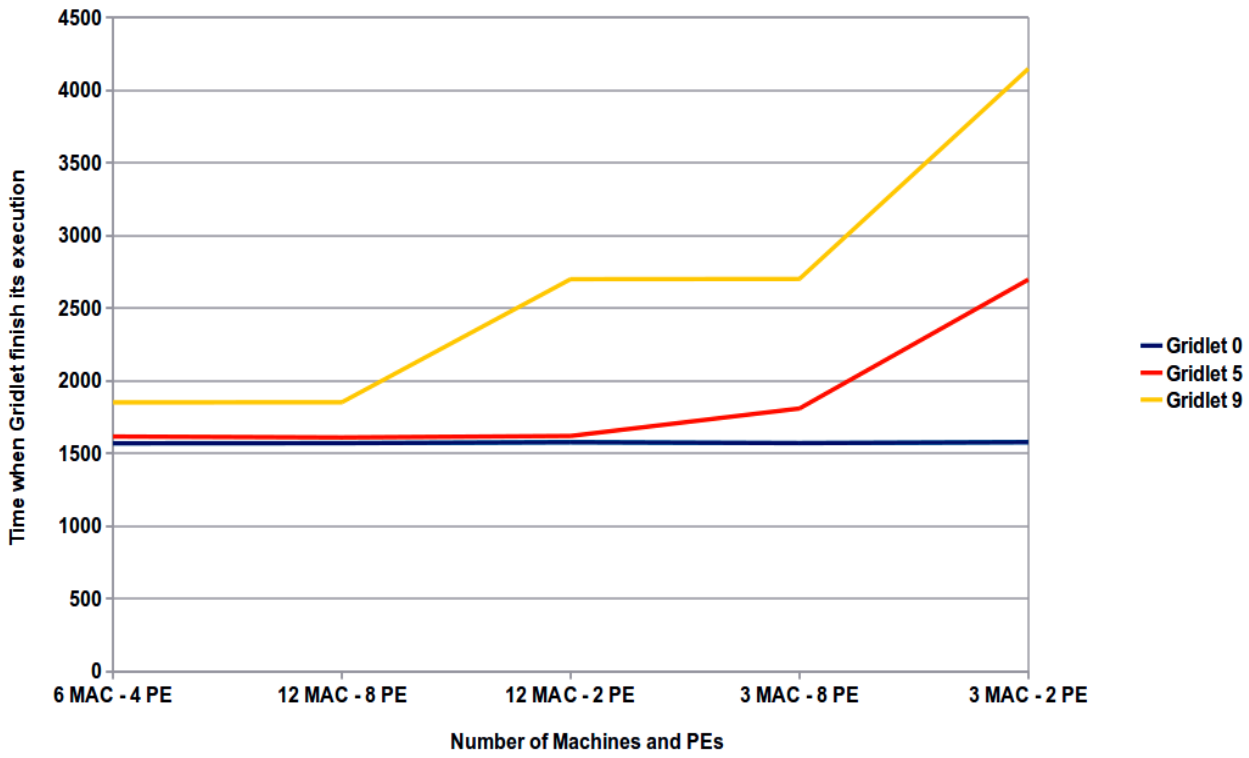


Figure 6.6: Test 3 - Performance of Success Time of Users 75 with different number of machines and PEs

6.4 Test Four - Variation in the length of gridelt

The fourth test allows us to verify the behavior of this network by varying the length of gridelts. In particular we will consider the *User_90* and evaluate the behavior of the first two and last two gridelts.
The standard length of gridelt we have used so far is 42000000 million instructions (MI). We tried to change this standard lenght with two different cases: in the first one we decrese the length by a factor of ten and in the second one we increase the length by a factor of ten.
The gridlets that we take into consideration are *Gridlet_0*, *Gridlet_1*, *Gridlet_8* and *Gridlet_9*.
The Table 6.3 shows us the *Gridlet_0*.

Table 6.3: Gridlet 0 of User 90 with different lenghts

Gridlet_0 Lenght	Execution Time	Success Time	CPU Time	Cost	Latency
4200000 MI	750.751	836.511	85.76	257.28	238.969
42000000 MI	741.53	1599.19	857.66	2572.98	866.540
420000000 MI	748.151	9319.661	8571.51	25714.53	8585.680

The execution time is similar for all three lengths.
Being the first gridelet of the *User_90*, it was less affected by the extent of the change in length as the traffic is low at the moment when comes into action.
Obviously, the time when the gridlet ends its execution is greatly influenced by the size of the gridlet.
CPU Time and cost have an almost exponential trend. When the length decreases by ten times their value also decreases by ten times. Similarly, if the length increases by ten times their value also increases tenfold.
The behavior of latency follows that of the two previous data when the length increases tenfold, while it decreases only three times when the length decreases tenfold.
This is because the value of latency is also influenced by time of success. And in this case time success with the length equal to 4200000 MI is very close to that with the length equal to 42000000 MI. This explains the higher latency.

Table 6.4: Gridlet 1 of User 90 with different lenghts

Gridlet_1 Lenght	Execution Time	Success Time	CPU Time	Cost	Latency
4200000 MI	788.301	874.301	86	258	274.259
42000000 MI	817.411	1675.411	858	2574	963.870
420000000 MI	9316.151	17887.621	8571.47	25714.41	16941.860

The *Gridlet_1* is a clear example that with increasing traffic, the length of gridlet has the greatest impact.
As we can see from the Table 6.4, it is quite clear that with increased length of gridelt there is a significant increase of all values.
In particular, we see this difference for the execution time, success time and latency. When we increase the length of gridelt, the values of the three data increased very much, and when we decrease the length of gridelt, the values decrease slightly.

Table 6.5: Gridlet 8 of User 90 with different lenghts

Gridlet_8 Lenght	Execution Time	Success Time	CPU Time	Cost	Latency
4200000 MI	1059.301	1145.301	86	258	174.681
42000000 MI	2546.181	3404.181	858	2574	2677.960
420000000 MI	17944.031	26516.031	8572	25716	25541.760

The Table 6.5 and the Table 6.6 confirm what was said previously.
The increase in traffic over the past two gridlets of each resource is much higher, and increasing the length of gridlet has a decisive effect on the values of the execution time, success time and latency.

Table 6.6: Gridlet 9 of User 90 with different lenghts

Gridlet_9 Lenght	Execution Time	Success Time	CPU Time	Cost	Latency
4200000 MI	1119.821	1205.821	86	258	231.23
42000000 MI	2592.251	3450.251	858	2574	2476.200
420000000 MI	26468.321	35040.321	8572	25716	34312.250

6.5 Test Five - Comparison between two cases

In the fifth and final test we compare two different situations.
In the first we leave the network with the same data, while in the second we will send all the gridlets to the resource *Munich*.
In this way we want to make clear what is the difference between a well-balanced network and a network totally busted.
The Figure 6.7 shows a graph with the levels of latency of both the cases for the *User_0*.
The difference is remarkable.

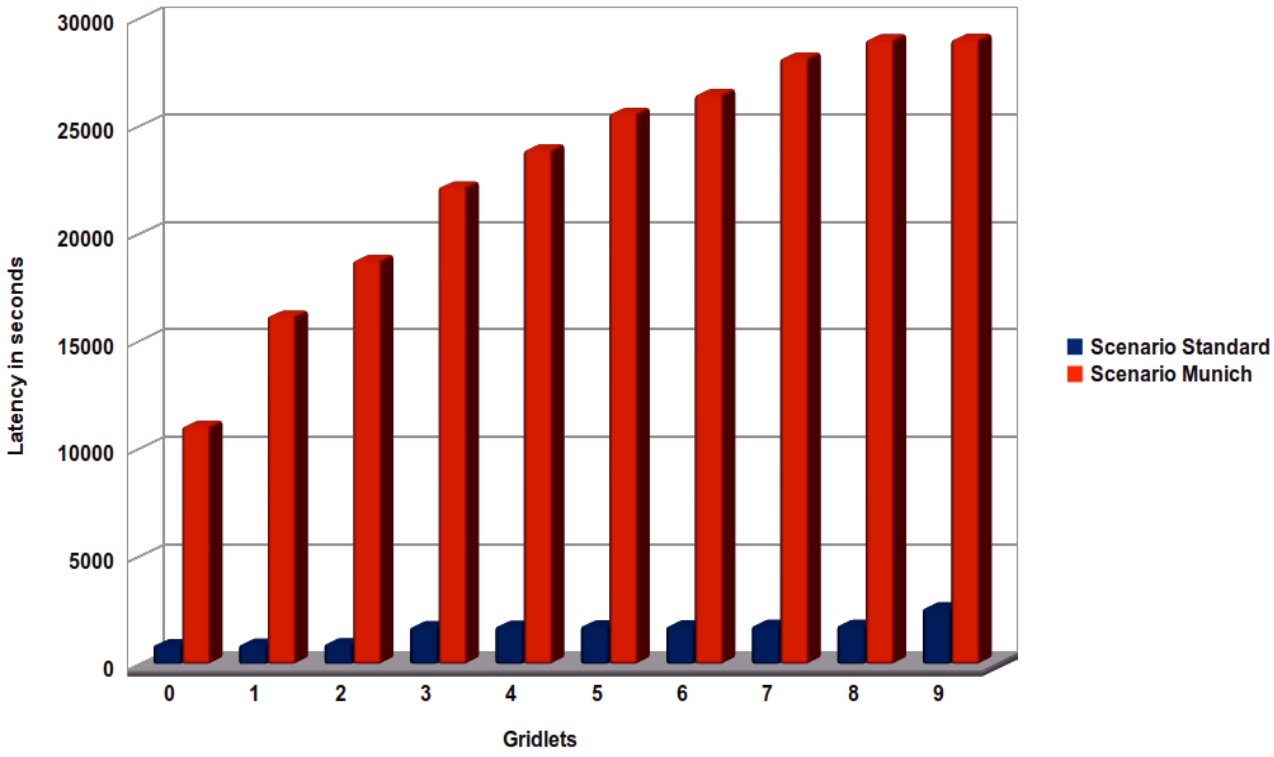


Figure 6.7: Test 5 - Performance of Latencies of User 0 in two different scenarios

In the Standard case the latency ranges from about 800 to about 2500 seconds. This result is in line with the previous tests, and that is what we expect with the default settings. In

the case where all gridlets are sent to Munich, the latency ranges from about 11000 to about 30000 seconds.
This is about twelve times higher. This huge difference can be explained simply.
All users send their six gridelts to resource *Munich*.
Thus 6 gridelts for 120 users are 720 gridelts received from the resource. What logs can be summarized as follows:

- The traffic is all directed towards *Munich*, then to the router number seven.
- This causes a blockage of the network, especially in Regional GIS number two.
- This means that each time a user sends a new gridlet, it is difficult for it to reach the resource.
- All this leads to the creation of code that switches the traffic.
- But since all traffic is facing *Munich*, queues created will be large.
- Therefore the resource will take a long time to run all gridlets.
- And all this goes to affect the latency.

The graph shows the $User\_0$ that is only part of the network.
All other users have similar differences between the standard case and that of Munich.
A major or a minor difference between the two cases may depend on such factors as the route to the resource (distance and baud rate connections), the exact moment when a gridlet reaches the resource (if there is a lot of a little bit of traffic at that moment), or how many gridlets are in the queue that has formed in the resource.

6.6 Comments of the results

In the second scenario we wanted to apply a different type of tests that still remain comparable with the first scenario.
The results obtained in this second scenario are still the results that we could expect.
Especially going in detail:

- Test one: we immediately implemented our settings, the only difference consists to observe the behavior of the network topology with the constant values of baud rates. If we take into account the traffic network that is significantly higher than the previous scenario, we can see that the results are in line with expectations: as more gridlets become part of the network, the greater the delay in executing.
- Test two: this test allows us to better understand the operation of one of the many features of GridSim, the ability to divide data packets into smaller units if they exceeded the value set as the MTU.
  The results show that the level of latency does not vary greatly from case to case, thanks to the feature described above.

- Test three: these results show how when varying the total computing power, i.e. number of machines and their PEs, the latency undergoes significant changes.
  In addition, this test also allows us to see the progress of two other times, when gridlet comes into action and when the gridlet finishes executing.
  As expected the results of these last two data are in line with the latency, the lower the power the greater the delay in responding to all three cases.

- Test four: these results indicate that varying the length of gridlet changes the various times taken into consideration.
  Again the results are in line with expectations, as the length increases, the time increase and vice versa.
  Only the time where gridlet come running does not change much, because a different size gridelt not affect the start time running.

- Test five: in this test we wanted to present that a well balanced network topology increases performance in the execution of gridlet, while a poorly balanced network has a negative impact on performance.

7 Final results

The results of these simulations give us an important view on the behavior of the toolkit in various situations. This allowed us to model the toolkit as we liked to get a network topology that reflects one hundred percent of our initial idea.
In order to obtain more significant results, we decided to implement two different scenarios. The first is based on EU DataGrid TestBed 1, built on a real network topology of a few years ago.
The second scenario is based instead on our own idea, bigger than the previous, so as to simulate an environment even more extensive.
This double selection allows us to develop our scenario with the huge advantage of being able to compare with a scenario that exists in reality.

$$Latency = Receiving\ Gridlet\ Time - Sending\ Gridlet\ Time \tag{7.1}$$

Latency has been our main unit of measurement in the various tests, in both scenarios. This is because we assumed scenarios without resources failures. Latency is simply the delay of each gridlet (7.1), the difference between the time when the gridlet is sent by user to the resource and time when the user returns gridlet with the work performed.

It is therefore clear that optimum performance of the network depends on these two values. Figure 7.1 shows the behavior of the latency.
If the difference between the two values is marked, then it increases the level of latency.
And as we have already fully explained, this difference depends on:

- number of gridlets per each user;
- traffic encountered in the network and resulting in the formation of queues;
- the path to reach the resource;
- the speed of each link (baud rate);
- the lenght of the gridlet;
- number of machines and number of processing elements;
- ...

Latency is also an excellent tool for comparing the two scenarios.

7.1 Comparison of results

In the first scenario we focused the first two tests on simple examples that have allowed us to know the best features of the toolkit.

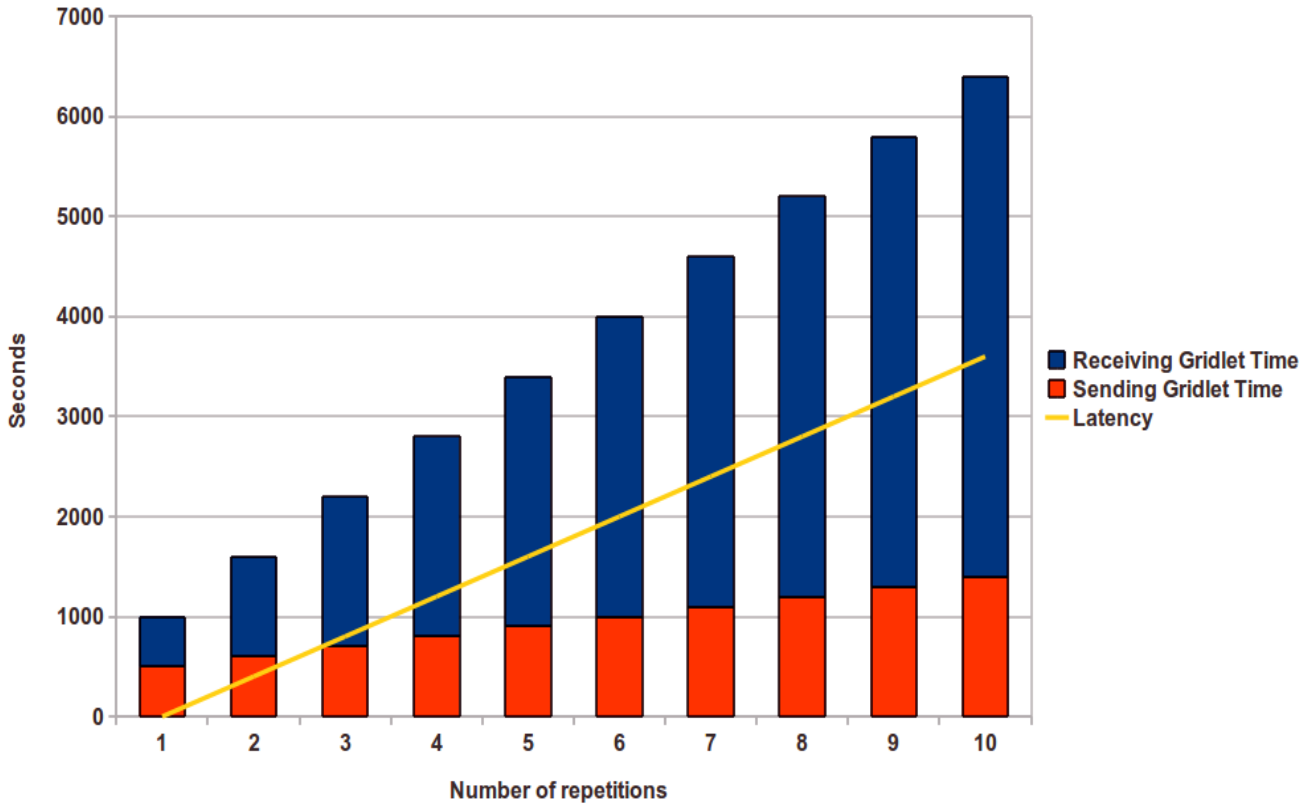


Figure 7.1: The behavior of the latency

In essence, we simulated a static network, i.e. with the same baud rate between routers and between routers and resources, with few gridlets and the same gridlets sent to their specific resources to better understand the operation of the network.

The results are in line with our expectations: if gridelts are sent to a resource nearby, then their latency will be low, but if they are dispatched to resources spread throughout the network, the latencies of gridlets undergo changes based on the paths they take.

If we consider a network with different baud rates, then this becomes an influential figure for the value of latency.
In general latency tends to increase when more gridlets join the network.
So it is quite obvious that the latency has a fluctuating behavior based on these characteristics.

The third example of the Scenario EU DataGrid TestBed 1 can be compared with the second example of the Scenario Artificial EU. In fact, these two examples are very similar to each other, except for some details.
Firstly, in the first scenario, the gridlets are redirected differently than the first two tests, so as to achieve a more balanced network.
Just as we set in the second test of the second scenario.
In the second scenario we have also changed the number of MTU. But for this comparison we will consider only the case standard with 1500 MTUs and the first five gridlets.

Table 7.1: Comparison of latency (in seconds) between two tests of the User 0

Gridlet	Scenario EU DataGrid TestBed 1	Scenario Artificial EU
0	869.410	865.821
1	901.410	898.301
2	1726.270	923.550
3	1757.650	1715.721
4	2582.180	1723.691

In Table 7.1 we can see the comparison.
For the first two gridlets the situation is similar, the value of latency is around 800/900.
The situation began to change from the third gridlet onwards, where the difference between the two scenarios starts to to become more pronounced nearly reaching 1000 seconds of difference in the latter case.
All this is explained by the increased power available to Scenario Artificial EU.
When the number of gridlets increases, the additional machine of the second scenario simplifies the work of the resource, but finds more difficulty performing the gridlet in the first scenario.

Another parallel can be drawn by test number four in the Scenario EU DataGrid TestBed 1 and test number three in the Scenario Artificial EU.
In both these examples we have changed the number of processing elements, while only in the scenario did we also vary the number of machines for each resource.
The result of this comparison is shown in the Table 7.2.

Table 7.2: Comparison of latency (approximate value in seconds) between two tests with different power

State of *Gridlet_0*	*User_0*	*User_75*
Scenario EU DataGrid TestBed 1 - 5MAC/4PE	2000	3000
Scenario Artificial EU - 6MAC/4PE	800	900
Scenario EU DataGrid TestBed 1 - 1MAC/4PE	8000	20000
Scenario Artificial EU - 3MAC/2PE	800	5000

The table shows us an interesting fact.
Moving from *User_0* to *User_75*, we have an minimal increase in traffic in the case of the EU DataGrid Testbed Scenario 1 with five machines and four PEs, while in the case of Scenario Artificial EU with six machines and four PEs PEs, there is almost no increase.
This is because of the additional machine which allows, in the latter case, greater efficiency in the execution of gridlet.

If we focus our attention in the case with a lower power the gap becomes wider.
In the first scenario, a single machine with four PEs can not be performing as three machiness with two PEs of the second scenario.
Once again, this underlines the importance of having a balanced network, especially with adequate computing power.

The test number five in the Scenario EU DataGrid TestBed 1 and test number four in the Scenario Artificial EU compare the different lengths of a gridlet.
In both experiments, the length of gridlet is both decreased and increased by a factor of ten, as is shown in Table 7.3.

Table 7.3: Comparison of latency (value in seconds) between two tests with different gridlet lengths for of the User 0

Lengths	Scenario EU DataGrid TestBed 1	Scenario Artificial EU
420000 MI	160.965	127.710
4200000 MI	867.460	861.501
42000000 MI	8598.340	8577.041

The standard adopted by the length of GridSim Toolkit is 4200000 Millions Instruction.
With this length we can see that in both cases the value of latency is always around 860 seconds, as happened in most of the tests for the first gridlet of the first user.
If we decrease the length of gridlet we get a decrease in latency, which is more evident in the second scenario.
This is because increased computing power, accompanied by a shorter length of gridlet, makes it faster to execute the gridlet by the resource.
The same result is obtained when the length of gridelt is greater than the standard case.
The increased computing power allows the second scenario to obtain an advantage over the first.
As we can see the difference between the two scenarios is not marked: this act can be explained by a potential difference that is a single machine and thus only creates a small change in the lengths.

Still to be considered are tests one and five in the Scenario Artificial EU.
The test number one with our default settings but with only the standard baud rates is helpful to understand how increasing the size of the network topology, the traffic increases.
This easy interpretation leads us to emphasize the importance of queues and queue management.
In the event that a resource has not successfully instantly execute the gridlets it is receiving, it will create queues that store incoming gridlets and then release them with the method first-come-first-served (FCFS).

While the fifth test show the enormous difference that exists between a network topology where gridelts are addressed in a balanced way, and a network where all gridlet are directed to the same resource.

The fifth test shows us that a good balance in the addressing of gridlets, allows for maximum effectiveness of the network topology.

7.2 Grid computing on large-scale examples

When we shift attention to large-scale scenarios, everything becomes more complicated because we have to take into account many variables.
We have to predict in advance all the situations that might occur.
The main problem we have often found is the formation of dangerous bottlenecks that can ruin the whole process of the grid.
Fortunately GridSim provides an automatic creation of queues that handle bottlenecks.
But all this, thought on a large scale, leads to an inevitable delay in the execution of each job.
To minimize the formation of bottlenecks and the consequent formation of queues we have to plan a balanced network in advance.
To design a well balanced network topology we must (for example) try to balance the destinations of the various jobs, trying to send fewer jobs in the resources that are less powerful and more jobs in the powerful resources.
The large-scale scenarios, however, allow us to link resources that are scattered all over the world, with different computational powers, reducing costs in both the public and the private sectors, which would be high.
The simulation is a key source for great success of the process of creating a real grid network.
The simulation in large-scale needed to be also well designed, but with the difference of being able to commit a certain number of errors and these errors can change at no cost, before creating a real grid network.
The ultimate goal is to create a final grid network that is practically perfect and ready to be implemented.

8 Conclusion

The tests carried out on the EU DataGrid Testbed Scenario 1 and Scenario Artificial EU have provided important results on the operation of the GridSim toolkit.
After our changes to the toolkit, we can say that GridSim remained efficient.
Indeed it has kept all the original features.

In several tests we have shown that it is possible to model and simulate a wide range of heterogeneous resources, such as single or multiprocessors, shared and distributed memory machines such as PCs and workstations with different capabilities and configurations.

It still supports a reservation-based mechanism for resource allocation.

In our test we can also note that the resources are geographically distributed across multiple administrative domains with their own management policies and goals.

In GridSim the broker (the scheduler of the resources) still focuses on improving performance of a specific application in such a way that its end-users' requirements are satisfied.

With our tests we can see that the toolkit allows the modeling of several regional GIS components.

Our changes have left intact the possibility of modeling and simulating all the network topology with different capabilities, configurations, and domains.

It still supports tasks for application jobs, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution.

Moreover the modified toolkit allows the simulation of workload traces taken from real supercomputers.

8.1 Related Works

The simulation has been used in an extensive way for evaluation and modeling of the real world systems, especially in buisness processes and in the computer systems design to assembly lines.
Consequently over the years, several software tools have been developed in order to make simulation a credible discipline.
We discussed grid computing and simulation in grid computing.
We now introduce the tool that we chose to perform the simulation of a grid network topology.
GridSim is an object-oriented toolkit for resource modeling and scheduling simulation.
GridSim can simulate different configurations.

It supports different application models that can be mapped to resources for execution by developing simulated application schedulers.
In Chapter 3 we have already discussed the architecture and components of the GridSim toolkit along with steps involved in creating GridSim based application-scheduling simulators.
The main feature of GridSim is the total adaptability to any situation.
We opted to work on GridSim because it has a complete set of features for simulating realistic grid testbeds.
Such features are modeling heterogeneous computational resources of variable performance, scheduling jobs, differentiated network service, and workload trace-based simulation from real supercomputers.
More importantly, GridSim allows the flexibility and extensibility to incorporate new components into its existing infrastructure.

8.1.1 Simulation Tools

As we mentioned in the previous chapters, simulations are essential for carrying out research experiments in grid systems.
A number of simulation tools for grids exist, such as GridSim [16], OptorSim[17], SimGrid[18] and MicroGrid[19].
These tools will be briefly mentioned next (and their characteristics are summarized in Table 8.1).

OptorSim has been developed as part of the EU DataGrid project[13], and it aims to study the effectiveness of data replication strategies.

SimGrid is an event driven simulator, which provides functionality to simulate infrastructures and applications based on their features.

Finally, MicroGrid provides on-line emulation of large-scale network and Grid resources. However, MicroGrid is actually an emulator, meaning that actual application code is executed on the virtual grid modeled after Globus toolkit[20].

To the best of our knowledge the above tools do not provide mechanisms to simulate computing resources failure.

The Table 8.1 indicates the key features that a complete toolkit should have.
As we can see GridSim is the only one of the main grid simulation toolkits to possess all these, and they are:

- *Data replication* is the process of sharing information so as to ensure to improve reliability in case of fault-tolerance, or accessibility.
  The data replication consists in storing data on multiple storage devices, or computation replication if the same computing task is executed many times.
  A computational task is typically replicated in space, but sometimes it could be replicated in time, if it is executed repeatedly on a single device.
  GridSim and OptorSim have this feature.

Table 8.1: Listing of functionalities and features for each grid simulator

Functionalities	GridSim	OptorSim	SimGrid	MicroGrid
data replication	yes	yes	no	no
disk I/O overheads	yes	no	no	yes
complex file filtering or data query	yes	no	no	no
scheduling user jobs	yes	no	yes	yes
CPU reservation of a fauilure	yes	no	no	no
workload trace-based simulation	yes	no	no	no
differentiated network QoS	yes	no	no	no
generate background network traffic	yes	yes	yes	yes

- *Disk I/O overheads* is the disk space required for non-data information (used for location and timing).
  GridSim and Microgrid have this feature.
- *Complex file filtering or data query* that allows the selection of file attributes in the replica catalogue.
  Only GridSim has this feature.
- *Scheduling user jobs* allows users to be able to do integrated studies of on demand replication strategies with jobs scheduling on available resources.
  GridSim, SimGrid and Microgrid have this feature.
- *CPU reservation of a fauilure*, it is possible to create, commit, activate, modify, cancel and query a reservation of a failure.
  Only GridSim has this feature.
- *Workload trace-based simulation*, the possibility to create a realistic simulation environment where the gridlets are competing with others.
  Only GridSim has this feature.
- *Differentiated network QoS*, the capability to simulate differentiated network quality of service.
  Only GridSim has this feature.
- *Generate background network traffic*, this is an important feature because in real-life, networks are shared among users and resources.
  Thus, performance may be affected by congested networks.
  All the toolkits in the table have this feature.

8.1.2 Failures

As we can analyze, the availability of resources may vary due to changes in their working conditions, in particular in the case with network congestion.

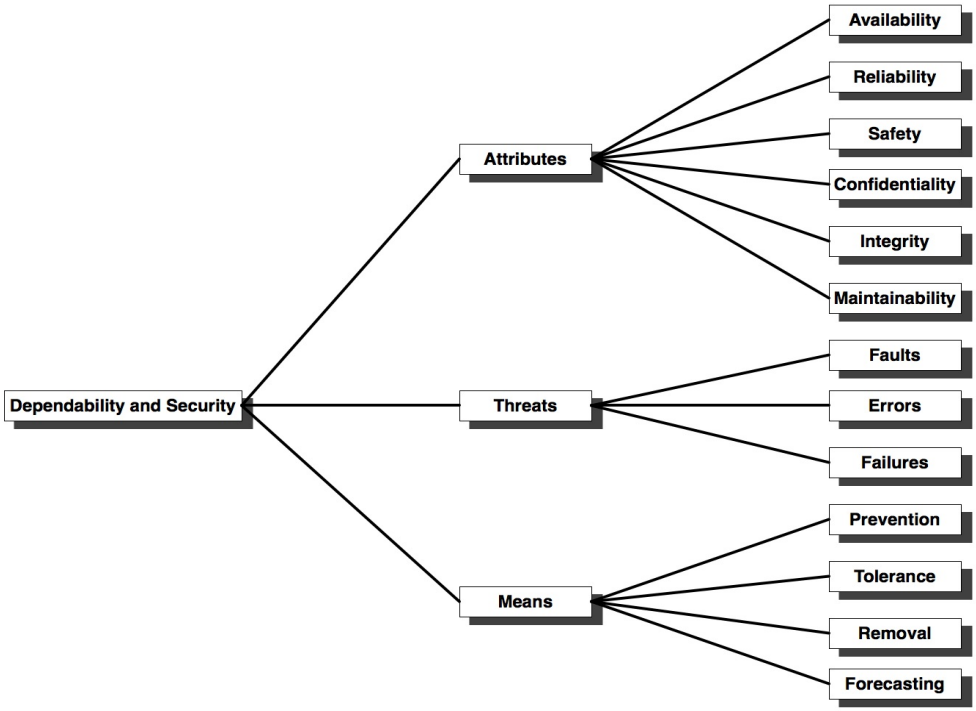


Figure 8.1: Relationship between "dependability & security"[21]

But there are also other conditions that may happen as partial failures or even the connection or disconnection of computing resources.
With many resources in a grid, the resource or network failures are the rule rather than the exception. Hence, they should be taken into account in order to provide a reliable service.
In order to cope with these challenges, from the fault tolerance point of view, the system must have failure detection and recovery schemes.

Thus in this thesis we studied some performances of the GridSim.
We did not discuss failures of the grid computing, we only introduce some characteristics.
Dependability can be thought of as being composed of three elements, as describe in Figure 8.1:

For a complete description of failure detection and recovery scheme, see [22].

List of Figures

Listings

List of Tables

Bibliography

[1] Proc. of the 1st Intl. Workshop on Grid Computing. *Data management in an international data grid project*, 2000.

[2] Rajkumar Buyya et al Anthony Sulistio. A toolkit for modelling and simulating data grids: An extension to gridsim. *Concurrency and Computation: Practice and Experience (CCPE)*, 2.02, 2002.

[3] Rajkumar Buyya. Grid computing and distributed systems laboratory and the gridbus project. 2008.

[4] What is grid computing 1, January 2011. `http://www.oppapers.com/essays/Mr/175664`.

[5] The data grid project, February 2011. `http://eu-datagrid.web.cern.ch/eu-datagrid/images/images/grid-small-prov.jpg`.

[6] What is grid computing 2, January 2011. `http://www.wisegeek.com/what-is-grid-computing.htm`.

[7] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE), Issue 13-15*, 14, 2002.

[8] Henri Casanova. Simulation for grid computing, January 2011. `http://www.nesc.ac.uk/action/esi/download.cfm?index=2653`.

[9] International Conference on Parallel and Distributed Systems. *Extending GridSim with an Architecture for Failure Detection*, 2007.

[10] Society for Computer Simulation Intl. (SCS), editor. *SimJava: A discrete event simulation library for Java*. International Conference on WebBased Modeling and Simulation, 1998.

[11] Anthony Sulistio et al. A toolkit for modelling and simulating data grids: An extension to gridsim. *Concurrency and Computation: Practice and Experience (CCPE)*, 20, 2008.

[12] Edg tutorial web site, February 2011. `http://hep-proj-grid-tutorials.web.cern.ch/hep-proj-grid-tutorials/`.

[13] Datagrid project, February 2011. `http://eu-datagrid.web.cern.ch/eu-datagrid`.

[14] Standard performance evaluation corporation (spec), February 2011. `http://www.spec.org`.

[15] Lcg computing fabric, January 2011. `http://lcg-computing-fabric.web.cern.ch`.

[16] Gridsim - a grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing, February 2011. `http://www.cloudbus.org/gridsim/`.

[17] Optorsim - a replica optimizer simulation, February 2011. `http://edg-wep2.web.cern.ch/edgwp2/optimization/optorsim.html`.

[18] Simgrid - a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments, February 2011. `http://simgrid.gforge.inria.fr`.

[19] Microgrid - emulation tools for computational grid research, February 2011. `http://www-csag.ucsd.edu/projects/grid`.

[20] Globus toolkit, February 2011. `http://www.globus.org/toolkit/`.

[21] Relationship between dependability & security, February 2011. `http://en.wikipedia.org/wiki/File:Dep-1.jpg`.

[22] Andrea Castiglioni. *Assessment of dependability scenarios with failures in large-scale grids, using GridSim toolkit.* Thesis, 2011.