USING GRID COMPUTING TO ENABLE HYPERSPECTRAL IMAGING ANALYSIS

By

Carmen Liliana Carvajal-Jímenez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

University of Puerto Rico
Mayagüez Campus
2004

Approved by:

Manuel Rodríguez, Ph.D. Member, Graduate Committee	Date
Néstor Rodríguez, Ph.D. Member, Graduate Committee	Date
Wilson Rivera, Ph.D. President, Graduate Committee	Date
Ana C. González, Master Representative of Graduate Studies	Date
Isidoro Couvertier, Ph.D. Chairperson of the Department	Date

ABSTRACT

USING GRID COMPUTING TO ENABLE HYPERSPECTRAL IMAGING ANALYSIS

By

Carmen Liliana Carvajal-Jímenez

Hyperspectral imaging analysis demands large input data sets and in turn requires significant CPU time and memory capacity. Grid Computing has the potential of improving the performance of these types of data and computational intensive applications. In this thesis we describe the design and implementation of Grid-HSI, a Service Oriented Architecture-based Grid application to enable hyperspectral imaging analysis. Grid-HSI provides users with a transparent interface to access computational resources and perform remotely hyperspectral imaging analysis through a set of Grid services. The base of the system is composed by a Portal Grid Interface, a Data Broker and a set of specialized Grid services. The system is based on the Open Grid Service Architecture (OGSA), and implemented on the top of Globus Toolkit 3 (GT3). We have conducted experiments that show the suitability of Grid-HSI to perform efficiently hyperspectral analysis.

RESUMEN

EL TITULO DE LA TESIS

Por

Carmen Liliana Carvajal-Jímenez

Las imágenes hiperespectrales demandan gran tiempo de procesamiento al igual que gran requerimiento de capacidad de memoria. La computación en malla tiene el potencial de mejorar el rendimiento para el análisis de imágenes hyperespactrales y de aplicaciones de computación intensiva. En esta tesis se describe el diseño y la implementación de Grid-HSI, una aplicación en malla basada en una arquitectura orientada a servicio para habilitar el análisis de imágenes hyperespactrales. Grid-HSI provee a los usuarios una interfase transparente para acceder a recursos computacionales y ejecutar remotamente análisis de imágenes hyperespectrales a través de un conjunto de servicios grid. El sistema esta basado en Open Grid Services Architecture (OGSA), e implementado con Globos Toolkit 3.0.

To my father Salomon Carvajal Mantilla, in loving memory

ACKNOWLEDGMENTS

I would like to thank infinitely my advisor Dr. Wilson Rivera for giving me the opportunity to work with him, for his continuous support, dedication and help in this project and for his friendship during all these years. Also, I would like to thank Dr. Manuel Rodrguez, Dr. Nestor Rodrguez for serving on my committee and for reviewing my work.

I thank to my husband Idalides for the academic contributions and for all love and compression during this work. I am grateful to my mother and my brothers for all their love, understanding, advice, and for always believing in me. Too, i am also grateful to all my friends who helped directly or indirectly in preparing this work.

Finally, I would like to thank the Department of Electrical and Computer Engineering of the University of Puerto Rico at Mayagez, the NSF Center for Subsurface Sensing and Imaging Systems (CenSSIS) and the Hewlett-Packard Technology Center at Puerto Rico for the financial support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 The Problem

Researchers and scientists at the NSF Engineering Research Center for Subsurface Sensing and Imaging Systems (CENSSIS) need access to heterogeneous and distributed resources to perform hyperspectral imaging data analysis. With the rapid advances in the resolution, frame rate, and dynamic range of spectrometers, the required bandwidth has soon exceeded throughput limits inherent in store and process systems. Thus, hyperspectral imaging analysis demands large input data sets and requires significant CPU time and memory capacity. It is then expected that Grid-level resources can play a significant role in improving performance while increasing pervasity of the image processing algorithms.

This thesis addresses the problem of demonstrating the suitability of Grid Computing technologies to advance hyperspectral imaging analysis.

The remainder of this chapter is organized as follows:

Section 1.2 describes the proposed solution.

Section 1.3 presents our research objectives.

Section 1.4 Background and reviews related work.

Section 1.5 summarizes our most important contributions.

Section 1.6 presents the structure of this writing.

1.2 The Solution

We have designed Grid-HSI, a Service Oriented Architecture-based Grid application to enable hyperspectral imaging analysis. Our first Grid-HSI prototype is composed by a Portal Grid Interface, a Data Broker and a number of Grid services to enable HSI analysis. Grid-HSI provides users with a transparent interface to access computational resources and perform remotely hyperspectral imaging analysis through a set of Grid services.

Our results suggest that the use of Grid-HSI is a new source of computational power that is accessible and applicable to the remote sensing problem set.

1.3 Research Objectives

The goal of this research is the development of a Grid infrastructure suitable for hyperspectral imaging analysis. The system will provide a transparent user interface and access to computing resources at a local environment at the University of Puerto Rico (UPRM). The specific objectives of the research can be listed as follows:

- Design and develop a service oriented architecture - based Grid platform suitable to hyperspectral imaging analysis.
- Develop a study of performability in a Grid platform.

1.4 Background and Related Works

Applications such as the monitoring of coastal environments often use broad area passive sensors mounted on space or airborne platforms, which collect data over many frequency bands. Current operational sensors deployed by NASA collect data over few low spectral resolution bands. However, new sensor concepts based on imaging spectrometry or so called hyperspectral imagers (HSI) collect high spectral resolution data over a couple of hundred of wavelengths effectively producing an image where at each pixel we get the spectral response of the object(s) in the field of view of the sensor.

Hyperspectral Imaging (HSI) data contains high spectral resolution and spatial information of the object under study. HSI analysis is based on the concept of imaging spectrometry where spectral and spatial information is used to identify or detect objects, or estimate parameters of interest (Figure 1.1). As the object of interest is embedded in a

Figure 1.1: Hyperspectral Imaging.

complex media (i.e. coastal waters or skin), the measured signature is a distorted version of the original object signature (e.g. a coral reef or a blood vessel) mixed with clutter. By large hyperspectral imaging analysis concentrates on dimensionality reduction and classification algorithms. Dimensionality reduction algorithms reduce the data information, so that it can be processed efficiently. Classification of a hyperspectral image sequence, in turn, identifies which pixels contain various spectrally distinct materials. Once all pixels are classified into one of several classes or themes, the data may be used to produce thematic maps. Depending on the nature of the application, the thematic maps may be used to produce summary statistics regarding the objects in a scene or for object or target recognition purposes.

Different classification metrics have been proposed from minimum distance, such as Euclidean, Fisher Linear Discriminant, and Malahanobis, to maximum likelihood [1] to correlation matched filter-based approaches such as spectral signature matching [2]. There are two major techniques to image classification: supervised and unsupervised. In supervised classification techniques, an analyst develops quantitative descriptions of the spectral characteristics of the various classes of interest for a particular scene. These descriptions are then used as reference spectral signatures against which every pixel in an image is compared. The pixels are classified according to the spectral signature they most closely resemble. In unsupervised classification, the algorithms do not use training data as the basis for classification. Instead, the algorithms examine the unknown pixels in the image and aggregate them into various classes according to the clusters found in the spectral space that contains the image.

Next, we discuss relevant work related to the parallelization and distributing of hyperspectral imaging codes.

Lugo [3] developed a parallelization and distribution of dimensionality reduction algorithms (Feedback Iterative Method and Principal Component Analysis) and classification algorithms (Eu-

clidean Distance Classifier and Maximum Likelihood Classifier). This was performed using the C programming language Message Passing Interface (MPI). The Parallelized Linear Algebra PACKage (PLAPACK) based on MPI was used to leverage complex matrix manipulations operations. This hyperspectral parallelized algorithm suite provides researchers with a powerful tool that allow flexibility and increase considerably new results by being able to combine algorithms and reduce dramatically the time of execution.

Rivera [4] implemented a system nonsupervised iterative system of reduction of bands and classification of pixels that integrates these two stages through a closed knot. It considers the number of pixels classes as entrance parameter to carry out the reduction of the dimension of the image. The objective of the system is to try to choose a good subset of bands in which the distance is maximized among the classes or enter its centroids. The system was implemented using MATLAB.

A MATLAB Toolbox for Hyperspectral Image Analysis [5] focuses on the optimization and integration of the unsupervised and supervised HSI classification algorithms developed at the UPRM Laboratory for Applied Remote Sensing and Image Processing (LARSIP). The toolbox contains algorithms to load images of several file formats, routines for dimensionality reduction, and both supervised and unsupervised classification algorithms.

The Parallel Computational Environment for Imaging Science (PiCEIS)[6], developed at the Pacific Northwest National Laboratory, is an image processing software designed for efficient execution on massively parallel computers. During processing and visualization an image is fully distributed in contrast to many of the current master-slave model for image processing. The user has the choices of either displaying the output to their monitors or to the IBM Scalable Graphics Engine. The heart of the communication strategy is based on the Global Array/ ARMCI Toolkit, an existing portable NUMA one-way globally addressable "shared memory" model for distributed SMP parallel computers. The Global Array model extends the current models of parallel computing using shared memory, pthread and message passing style of one sided get/put model in MPI-2. While PiCEIS is focused on distributed SMP parallel computers, Grid-HSI is platform independent.

The WebDedip [7] explores object oriented modeling technique in the web domain. The WebDedip has a three-tier architecture composed by GUI, DedipServer and Agents. Java distributed

object architecture is used along with the object serialization for network communication among the components. The GUI is the web enabled graphical user interface to make the entire user interaction truly system independent. It has a back-end DedipServer running on the web site. When the GUI submits the request to the DedipServer, it reads the application configuration information from the configuration file. The DedipServer initiates the execution of the first process in the interdependency chart. It informs the agent(s) on the target node to start the execution of the process. The agent sends the status information back to the DedipServer when the process is completed. While WebDesip is based on Java distributed object architecture, Grid HSI is fully based on OGSA architecture providing a functional Grid based application system.

JSIM[8], the Java-based SAR image analysis tool environment, is a tool environment for image analysis of synthetic aperture radar data. The tool environment was designed and developed using Java programming language. JSIM use the client-server approach in which multiple clients from a local or a remote machine can access the same application at any time. The tool runs on various platforms and can be used through the Internet. Similarly to WebDedip, JSIM does not provide Grid computing facilities.

1.5 Summary of Contributions

In summary, this research makes the following contributions:

- A demonstration of the suitability of Grid computing technologies to enable hyperspectral imaging analysis.
- A complete description of the methodology and technical issues when developing service oriented Architecture - based Grid applications.

1.6 Thesis Structure

The reminder of this thesis is organized as follows: Chapter 2 presents an overview of Grid computing technologies. In particular, the definition, specification and implementation of the Open Grid Services Architecture (OGSA). Chapter 3 presents the design and implementation of the

proposed solution (Grid-HSI), it also discusses the esperimental results. Chapter 4 presents a study of performability issues in Grid platforms. And finally, chapter 5 lists our conclusions and suggests some areas for future work.

CHAPTER 2

Overview of Grid Computing

2.1 Introduction

In this chapter, we present an overview of Grid Computing. Grid computing involves coordination and networking of resources across dynamic and geographically dispersed organizations in a transparent way for users[9]. Grid technologies emphasize effective operation in large scale, wide area environments, including access to remote computation, information services, high speed data transfers, special protocols (e.g., multicast), and gateways to local authentication schemes. The Open Grid Services Architecture (OGSA) and its associated implementation, the Globus Toolkit 3.0, are becoming a standard platform for Grid services and application development, based upon Web services protocols and open standards.

The remainder of this chapter is organized as follows:

Section 2.2 provides a high level description of the Grid architecture.

Section 2.3 describes the Open Grid Services Architecture (OGSA)

Section 2.4 describes the Globus Toolkit 3.0

Section 2.5 summarizes the issues discussed in this chapter.

2.2 Architecture of Grid

In this section we present Grid architecture as it is described in the Globus project [10].Grid architecture (see Figure 2.1) represents a conceptualization of the main principles and requirements in Grid environments. The motivation for building this architecture is the need for a new model describing sharing of heterogeneous resources. This architecture identifies the basic components of a Grid systems, defines the purpose of such components, and finally indicates how these components interact each other.

The architecture of the Grid is described in terms of layers, each providing a specific function. In general, higher layers are focussed on the user (user-centric), whereas lower layers are more focussed on computers and networks (hardware-centric). The different layers and functionalities are

Figure 2.1: Architecture of Grid, Source: The Anatomy of the Grid [10]

described as follows.

Fabric layer: Interfaces to local control, of physical and logical resources. The fabric layer is composed by computational resources, storage systems, catalogs, distributed file systems, network resources, and sensors to be share.

Connectivity layer: Defines core communication and authentication protocols supporting Grid-specific network transactions.

The authentication protocols are governed by the following principles:

- *Single sign on*, applies to enabling the user to have multiple access to the resources from the Fabric layer during the same login, once the authenticity has been established. That is, once sign on is performed, the user is authenticated for the entire Grid.
- *Delegation* designates the ability to provide a program with the appropriate rights such that it could behave on user's behalf and further access those resources to which the user has permissions.
- *Integration* with various local security solutions addresses the issue of allowing communication with the local security solutions by providing mapping to the local environment. For instance, Grid security should be able to cooperate with Kerberos and Unix security which could be implemented by the providers of sites or resources.
- *User-based trust relationships* is concerned with directing the security constrains from the user to the intended resources and not further among their providers.

Resource layer: Allows the sharing of a single resource. This layer includes protocols for control and management of individual resources.

Two primary classes of resource layer protocols can be distinguished:

- *Information protocols* are used to obtain information about the structure and state of a resource, for example, its configuration, current load, and usage policy.
- *Management protocols* are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operation(s) to be performed, such as process creation or data access. Management protocols are responsible for instantiating sharing relationships, ensuring that the requested protocol

operations are consistent with the policy under which the resource is to be shared. Issues that must be considered include accounting and payment. A protocol may also support monitoring the status of an operation and controlling the operation.

Collective layer: Allows resources to be viewed as collections. This layer includes all the services that allow us to manage several resources.

Examples of services are:

- Directory services enabling the discovery of resources. A directory service supports queries for resources by name or by attributes such as type, availability, or load.
- Monitoring and diagnostics services enabling fault detection, such as overload, failure, intrusion.
- Grid-enabled programming systems thus extending their functionality by augmentation.
- Data replication services, dealing with storage capabilities to overcome issues such as data access performance measured in terms of response time, reliability and cost.
- Software discovery services envisaging the discovery and selection of the best software and platform for solving a selected problem.

Resource and connectivity protocols handle all Grid specific network transactions between different computers and other resources on the Grid.

Application layer: Uses the appropriate components of each layer to support the application. Applications Grid access to the infrastructure. According to the requirements of the application, it can be necessary to happen through all the layers or to connect themselves directly to the infrastructure.

In the next sections we go deeply in the anatomy of grid architecture to understand aspects of resource sharing with maximum interoperability of the Grid Architecture, referred to as Open Grid Services Architecture (OGSA); the specification of the archicteture, referred as Open Grid Service Infrastructure (OGSI); and one implementation of the OGSI specification, the Globus Toolkit 3.0 (GT3).

2.3 Open Grid System Architecture (OGSA)

The Open Grid Services Architecture (OGSA) defines Grid Services [10] as extensions of Web services. Thus, Grid services are basically Web services with improved characteristics to make them adequate for Grid-based applications.

The two principal elements of the OGSA Platform (See Figure 2.2)are Open Grid Services Infrastructure (OGSI), and OGSA Platform Services.

Figure 2.2: OGSA Platform

Open Grid Services Infrastructure (OGSI) defines mechanisms for creating, managing, and exchanging information among entities called Grid services. Platform Services provides a set of interfaces that support the negotiation of policies, service level agreements, reservations, and in addition maps the related agreements to Grid services. Moreover, it provides the basic functionality

to manage data in a Grid environment. This element defines the Common Management Model (CMM) that provides the manageability infrastructure for resources in OGSA. CMM defines the base behavioral model for all resources and resource managers in the Grid, plus management functionality like relationships and lifecycle management.

2.3.1 Open Grid Service Infrastructure (OGSI)

OGSI [11] is a Grid software infrastructure standardization effort based on the emerging Web service standards to provide maximum interoperability among OGSA software components. *Open Grid Service Infrastructure* OGSI addresses detailed specifications of the interfaces and conventions that a service must implement in order to fit into the OGSA framework.

Next the different interfaces and conventions are detailed.

Factory. The Grid services, which implement this interface, provide a way to create new grid services. Factories can create provisional instances of limited function, such as a scheduler creating a service to represent the execution of a particular job. Factories also may create longer-lived services such as a local replica of a frequently used data set.

Life cycle. Because grid services may be transient, grid service instances are created with a specific lifetime. The lifetime of any particular service instance can be negotiated and extended as required.

State management. OGSI specifies a framework for representing state called Service Data and a mechanism for inspecting or modifying that state named Find/SetServiceData. Moreover, OGSI requires a minimal amount of state in Service Data Elements that every grid service must support, and requires that all services implement the Find/SetServiceData portType.

Service groups. Service groups are collections of grid services that are indexed using Service Data. These services may be members of a group for a specific reason, as part of a federated service, or they may have no specific relationship such as the services contained in an index or registry operated for discovery purposes.

Notification. The state information (Service Data) that is modeled for grid services changes as the system runs. Many interactions between grid services require dynamic monitoring of changing state. The notification framework allows for asynchronous, one-way delivery of messages from a source to a subscribed sink. Grid services support an interface (NotificationSource) to permit other grid services (NotificationSink) to subscribe to changes.

HandleMap. When factories are used to create a new instance of a grid service, the factory returns the identity of the newly instantiated service. This identity is composed of two parts, a Grid Service Handle (GSH) and a Grid Service Reference (GSR). A GSH is a standard Universal Resource Identifier (URI) - it indicates how to locate the Instance, but not how to communicate with it. Before the GSH can be used, it must be resolved into a Grid Service Reference. GSR is a mechanism to convey capabilities of a service to a client. The HandleMap interface provides a way to obtain a GSR given a GSH.

OGSI creates a new extension model for WSDL called GWSDL. There are two core requirements for describing Web services based on OGSI:

- The capacity to describe interface inheritance, which is a core concept with most of the distributed object systems.
- The capability to describe additional information elements with the interface definitions. One such information element is called service data and is discussed later.

Several software frameworks available today are based on the OGSI specification. The Globus GT3 is the most prominent and practical implementation of this specification.

2.3.2 Web Services

Web services is a subset of a service-oriented architecture [12]. A web service is an application or application component accessible on the web and intended to be used by another application, a client application [13]. The concept relies heavily on XML and its family of technologies.

A web service is a specific kind of service that is identified by a Uniform Resource Identifier (URI)[14] and exhibits the following characteristics:

- It exposes its features programmatically over the Internet using standard Internet languages and protocols, and
- It can be implemented via a self-describing interface based on open Internet standards (e.g., XML interfaces which are published in a network-based repositories).

The Web Services are oriented to clients with not a previous knowledge of the service until this is invoked to the user, meaning, exists a strong undocking between client and servant of the service. One main feature of Web Service it is that they do not handle services with state information (stateful), that is to say, "do not remember" values from a call to another one.

Web services provide a set of XML-based protocols and processes for service discovery (UDDI), description (WSDL), and interaction (SOAP) (See Figure 2.3).

Figure 2.3: Web Services Platform

Universal Description, Discovery, and Integration (UDDI) [15] provides a platform-independent way of describing and discovering Web services and Web service providers. The UDDI data structures provide a framework for the description of basic service information, and an extensible mechanism to specify detailed service access information using any standard description language through a centralized registry of services.

UDDI provides two basic specifications that describe a service registry's structure and operation:

- A definition of the information to provide on each service, and how to encode it. UDDI encodes three types of information about Web services:
  - "white pages" information includes name and contact details.
  - "yellow pages" information provides a categorization based on business and service types
  - "green pages" information includes technical data about the services.
- A query and update API for the registry that defines how this information can be accessed and updated.

UDDI defines a SOAP-based API for querying centralized Web Service repositories. UDDI makes possible to discover the technical details of a Web Service (WSDL) as well as other business-oriented information and classifications.

Web Services Description Language (WSDL) [16] is the de-facto XML-based standard for describing Web services, provides documents to achieve self-describing, discoverable services and interoperable protocols, standard mechanisms for creating, naming, and discovering transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. These WSDL documents drive the communication between the parties in the grid, and are identical across all platforms, tools, and programming languages implementing OGSA.WSDL complements the UDDI standard by providing a uniform way of describing the abstract interface and protocol bindings of arbitrary network services. In OGSA, services adhere to specified Grid service interfaces and behaviors defined in terms of WSDL interfaces and conventions and mechanisms for creating and composing sophisticated distributed systems.

The main structure of a WSDL document looks like this:

```
<definitions>
   <types>
       definition of types...
   </types>
   <message>
       definition of a message...
```

```
    </message>
    <portType>
        definition of a port...
    </portType>
    <binding>
        definition of a binding...
    </binding>
</definitions>
```

The basic elements of a WSDL document include data descriptions, abstract interface definitions, and bindings to actual implementations. They are embodied in a set of XML constructs that build upon each other. The first part of a document consists of a *types* section that acts as a container for data type definitions using XML schemas. The types section is followed by a *message* section that defines abstract definitions of data to be communicated based on the types already defined. Messages define exchanges between the service provider and requestor,in turn and are abstractly described and later on bounded to a concrete network protocol and a message format. The next XML section defines one or more *portTypes*, which consist of a collection of *operations*, where each operation defines an action supported by the service. Each action is defined in terms of an exchange of the messages already defined in the previous XML section. A *serviceType* section defines a collection of portTypes provided by the service. Finally, mappings of these abstract definitions to concrete implementations occurs using ports and services, where a service is an implementation of a serviceType and consists of a collections of network endpoints or ports, where a port is an implementation of a portType, and is defined by a *binding* that contains a concrete protocol and data format spec for a particular portType.

Simple Object Access Protocol (SOAP) [17] is a language standardized by the consort W3C, that defines a simple and extensible XML messaging framework that can be used over multiple protocols with a variety of different programming models. SOAP also defines a complete processing model that outlines how messages are processed as they travel through a path. Overall, SOAP provides a rich and flexible framework for defining higher-level application protocols that offer increased interoperability in distributed, heterogeneous environments.

The architects of SOAP based it on two common protocols: HTTP [18] and XML [19]. SOAP treats XML as an encoding scheme for request and response parameters of the method calls and uses HTTP as a transport layer.At its core, a SOAP message has a very simple structure: an XML element with two child elements, one of which contains the header and the other the body. The header contents and body elements are themselves arbitrary XML. In addition to the basic message structure, the SOAP specification defines a model that dictates how recipients should process SOAP messages. The message model also includes *actors*, which indicate who should process the message.

The SOAP structure looks like this:

```
<soap:Envelope
 xmlns:soap="...">
  <soap:Header>
    <!-- extensible headers -->
  </soap:Header>
  <soap:Body>
    <!-- payload -->
  </soap:Body>
</soap:Envelope>
```

2.3.3 Grid Services

OGSA [20] specifies three conditions that a web service must have before it qualifies as a Grid Services. First it must be an instance of a service implementation of some service type as described above. Second, it must have a Grid Services Handle (GSH), which is a type of Grid URI for the service instance. The GSH is not a direct link to the service instance, but rather it is bound to a Grid Service Reference (GSR). The GSR might be (the OGSA allows for other representations) the WSDL document for the service instance with the required "instanceOf" and other OGSA extensions. The idea is that GSH provides a constant way to locate the current GSR for the service instance, because the GSR may change if the service instance changes or is upgraded.

A Grid Service is a locatable instance and, potentially with state (stateful), that implements one or more interfaces described by way of portTypes of WSDL. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability. These additional mechanisms can be grouped in four areas:

- *Naming* It assures the existence a unique name for each instance on Grid Service and allows the location of Grid services (discovering) by means of name
- *Service Data*. It manages the data sets associated to the execution of a Grid Service. It is useful to index Grid Services according to their characteristics and capabilities
- *Notification* These are the mechanisms used for the communication between the components of a Grid application. The notifications are very closely related to service data. Because when any change occurs in any Service Data the notification source will notify to all notifications sinks.

  There are two types of notification:

  - Push - The data is sent to the client as soon as it is received at the service end. This gives the client no control over what it receives but it is very efficient.
  - Pull - A message is sent to the client saying data is available for pickup, the client then has to request the data from the service in a separate call. The purpose of this implementation is that it gives far more control to the client end to select what data it receives. However, it does however increase the overheads.
- *Lifecycle management¿* Mechanisms for the creation and destruction of instances of Grid services.

A Grid service implements one or more interfaces, where each interface defines a set of method operations that is invoked by constructing a method call through method signature adaptation using SOAP. Grid service interfaces correspond to portTypes in WSDL used in current Web services solutions. The standard interface of a Grid service includes multiple bindings and implementations (such as the Java and C# languages).

A web service can only have a single port type - the interface exposed by the web service in WSDL. A grid service however can define a new port type as an extension of another port type, an example of this is: a service always extends the GridService port type, which is defined by the

OGSA. Other port types that can be extended to provide extra functionality to a grid service are defined by the OGSA, for example the NotificationSource port type that is extended if a service is to act as a notification source. An important point is that the NotificationSource port type will extend the GridService port type [21]. Figure 2.4 illustrates the concepts surrounding OGSI, and its relation to Web Services.

Figure 2.4: Web service vs Grid service

Table 2.1 summarizes the OGSA Grid Services interfaces.

Grid services can be deployed on different hosting environments – even different operating systems. OGSA also provides a Grid security mechanism to ensure that all the communications between services are secure. All the services (persistent or transient) are built on the Globus Toolkit.

Port Type	**Operation**	**Description**
GridService: all Grid services implements this interface and provides these operations and behaviors.	FindServiceData	Allows a client to discover information about the service's state, execution environment, and additional semantic details not available in the GSR.
	SetTerminationTime	Set and get termination time for Grid service instance.
	Destroy	An operation to explicitly destroy an instance.
Notification-Source: allows interested parties to subscribe to service data elements and receive notification events when their value is modified. (Optional)	SubscribeTo NotificationTopic	Subscribe to be notified of subsequent changes to the target instances service data.
Notification-Sink:enables a Grid Service instance to receive notification messages based in a subscription. (Optional)	DeliverNotification	Used to send a given message to all subscribers to a particular topic.
Registry: maintains a collection of Grid Service Handles, with policies associated with that collection (Optional)	RegisterService	Add or atomically update a Grid Service Handle to the registry.
	UnregisterService	Remove a Grid Service Handle from the registry.
Factory: provides a standard WSDL operation for creation of Grid service instances. (Optional)	CreateService	Create a new Grid service instance
HandleMap: defines a option for mapping one GSH to one GSR	FindByHandle	Returns a Grid Service Reference for a Grid Service Handle.

Table 2.1: OGSA Grid services interfaces

2.4 Globus ToolKit 3.0

The Globus project is a multi-institutional initiative for the investigation the development of fundamental technologies for Grid computing. Globus is the standard de facto for the implementation of Grid applications. The toolkit addresses issues for resource monitoring, discovery, management, security and file management. The version Globus ToolKit 3.0 has been implemented to adjust to the OGSA requirements. Globus is based on three pillars: Resource management, information services and data management [22]. These components use the Grid Security Infrastructure (GSI) protocol for security at the connection layer (See Figure 2.5).

Figure 2.5: Globus Toolkit 3.0 Architecture Layers

The Globus Toolkit Version 3.0 (GT3) provides three components referred as GT3 Base Services.

2.4.1 Resource Management Components

The Resource Management component, involves the allocation of Grid resources. It includes packages such as the Globus Resource Allocation Manager (GRAM) and Globus Access to Secondary Storage (GASS).

Grid Resource Allocation Manager (GRAM) Reports, monitors and publishes in-

formation about the identity and state of local computations (registry). Moreover, it allows users to schedule and manage remote computations. Specifically, various classes and methods allow users to submit jobs, bind to already submitted jobs, and cancel jobs on remote computers. Other methods allow users to determine whether or not they can submit jobs to a specific resource (through a Globus gatekeeper) and to monitor the job status (pending, active, failed, done, and suspended).

A Grid may comprise more then one GRAM, each of them controlling a set of resources. By means of control or management we defer operations such as submission, monitoring, pausing or stopping. The job manager is created by the gatekeeper located on the remote computer and is responsible for starting and monitoring the job as well as for sending back to the client information regarding the changes in the job's status. A job manager exists for every client request and consists of a Common Component and a Machine-Specific Component. The later implements the internal API used by the former. The Resource Specification Language (RSL), which is a structured language for specifying the resource requirements and parameters, is also parsed by GRAM. A gatekeeper is a process running as root on the server before any requests are sent from the client machine and its tasks are:

- Mutual authentication with the client
- Mapping the remote user to a local one
- Activating a job manager on the local host as a local user
- Pass the allocation arguments to the job manager

When the job is finished, the job manager sends the status information back to the client and terminates

Global Access to Secondary Storage (GASS) simplifies the porting and running of applications that use file I/O, eliminating the need to manually log onto sites and ftp files or to install a distributed file system. Globus provides an essential subset of GASS services to support the copying of files between computers on which the Grid Services are installed.

2.4.2 Information Services Components

Information service components, implemented as the Index Service, is one of the GT3 Base Services. It can be used to index Service Data carrying state information from multiple grid service instances for use in resource discovery, selection and optimization. The Index Service uses an extensible framework for managing static and dynamic data for Grids built using GT3. It aggregates Service Data from multiple grid service instances using the subscription-notification mechanism defined by the OGSI specification. It uses a Service Data Provider program to create and manage dynamic service data and can also perform registration of the instances of Grid Services.

Service data is an interface that is easily to query and allows clients to access data elements, it has two forms:

- State information - provides information about the current service state. An example of this would be a service that performs a calculation and the service data it updated to state what the operation was and the results.
- Service metadata - provides information describing the service being provided. An example of this is a service that performs calculations. The service data describes what calculations the service can provide and at what cost.

Service data is defined in a separate Service Data Elements (SDE)which is pointed to in the GWSDL file. A single service can have multiple SDE's and each element within a single SDE can have a defined cardinality [21]. Service data is modeled in the OGSI - defined namespace attribute. This OGSI schema type for service data contains seven predefined atributes (See Table 2.2). These attributes are standart XSD types, with exception of the *mutability* attribute.

2.4.3 Data Management Components

Data Management components, involves the ability to access and manage data in a Grid environment. This involves utilities such as GridFTP and globus-url-copy, which are used to move files between grid enabled .

SDE Attributes	Description
maxOccurs	Indicates the maximum number of values that a SDE can have. The value of this attribute can be unbounded, which indicates an array with no size limit.
minOccurs	Indicates the minimum number of values that a SDE can have
modifiable	Specifies if the value of a SDE can be changed by a client. Default value = false (all SDEs are by default "read only")
nilable	Specifies if the value of this SDE can be NULL. Default value = false
mutability	Indicates of whether and how the values of a SDE can change. This attribute can have the following values: * static: The value of the SDE is provided in the GWSDL description. * constant: The value of the SDE is set when the Grid Service is created, but remains constant after that. * extendable: New elements can be added to the SDE, but not removed. * mutable: New elements can be added and removed.

Table 2.2: Service Data Element Attributes

The Reliable File Transfer (RFT) is an OGSA based service that provides interfaces for controlling and monitoring 3rd party file transfers using GridFTP servers. GridFTP is based on the FTP protocol [RFC 959] and provides a file transfer service with linked with grid security mechanisms. GridFTP Is the protocol proposed for all data transfers on the Grid. It extends the standard FTP protocol with facilities such as multistreamed transfer, autotuning and globus based security. GridFTP must support Grid Security Infrastructure (GSI) and Kerberos authentication, with user controlled setting of various levels of data integrity and/or confidentiality.

2.4.4 Grid Security Intrastructure (GSI)

GSI delivers a secure method of accessing remote resources. It enables a secure, single sign-on capability, while preserving site control over access control policies and local security infrastructure. GSI uses X.509 certificates, a widely employed standard for PKI certificates, as the basis for user authentication. This service Support robust and flexible authentication, integrity, and confidentiality features are critical when transferring or accessing files. GSI is a hierarchical infrastructure with distributed services allowing for scalability and fault tolerance (a resource failure does not affect other resources). GSI provides the following two protocols:

- Enquiry protocol: Grid Resource Inquiry Protocol (GRIP)
- Registration protocol: Grid Resource Registration Protocol (GRRP)

The Grid imposes two types of servers from which information is stored. These components interact with each other and higher-level services (or users) using two basic protocols: a soft-state registration protocol (GRRP) for identifying entities participating in the information service, and an inquiry protocol (GRIP) for retrieval of information about those entities. In brief, a GRIS uses the registration protocol to notify a GIIS (or other higher-level service) of its existence. A GIIS uses the inquiry protocol to obtain information from the known to that provider, which merges into an aggregate view.

2.5 Web Services -Resource Framework

WSRF defines a new approach for accessing stateful resources which supports the grid computing infrastructure and also provides for stateful Web services in general - in other words, the special-case solution for grid computing (in OGSI) was generalized to the benefit of all Web services (through WSRF). With the WSRF, new standard interfaces for Grid services has been introduced which makes it necessary to change the implementations using OGSI, if they should comply with the new upcoming standard.

2.5.1 Globus ToolKit 4.0

Globus Toolkit 4 Based on the new standard WSRF interfaces, a new version of Globus, the Globus Toolkit 4.0 has been announced for the final of 2004. This new version will still have the same native interfaces like Globus Toolkit 2.4. The Globus Toolkit 3.0 interfaces will be dropped and replaced by new WSRF compatible interfaces. GT4 will be incompatible with GT3.

2.6 Summary

This chapter provided a high-level overview of Grid computing. It describes the component structure of OGSA and its associated implementation, the Globus Toolkit (GT3). The Globus GT3 is the most prominent and practical implementation of this specification. OGSA is the critical component to making Grids work. OGSA authors propose serviceData elements and an operation FindServiceData to provide introspection in OGSA Web Services.

A major difference between web services and the grid is that web services lack many services that are required by the potential users of Grid (such as those explained above). Web services however, are catching up with grid and adding services which will eventually lead to a convergence of the technologies. Grid based applications, contrary to client-server approaches, provides the capabilities of persistence and potential transient process on the web. Thus, we can perform a chain of operations and we would have to get the result of one operation and send it as a parameter to the next operation.

CHAPTER 3

GRID - HSI: A Grid Service Oriented Application

3.1 Introduction

This chapter focuses on the description of the architecture, design and implementation of Grid-HSI, a hyperspectral imaging analysis tool immersed into a grid platform which supports remote analysis and visualization . The system is based on Open Grid Service Architecture (OGSA) and implemented on the top of Globus Toolkit 3.0 (GT3). Grid-HSI is composed by a Portal Grid Interface, a Data Broker and a set of specialized Grid services. Experimental results show the suitability of the prototype system to perform efficiently hyperspectral analysis.

The remainder of this chapter is organized as follows:

Section 3.2 describes the design of Grid-HSI

Section 3.3 presents our implementation.

Section 3.4 presents experimental results

Section 3.5 summarizes the issues discussed in this chapter.

3.2 Design of Grid - HSI

3.2.1 Grid Computing Environment

A Grid Computing Environment (GCE) is essentially a set of tools and technologies that provides user-friendly interfaces to access various grid resources and services while hiding the complexities. Such Grid services and resources are accessed through Grid clients. The Commodity Grid toolkit (COG) provides a programmatic interface to standalone Grid clients written in programmatic languages such as Java and Python. Thus, web centric applications use CoG application program interfaces to provide friendly interfaces and make Grid clients accessible from a browser. In this project a GCE is deployed as a Computational Web Portal built on a multi-tier architecture as shows in figure 3.1.

Among the major features of a Computational Web Portal we have that it:

- Extends the user desktop by providing a seamless access to remote computational resources (hardware, software, and data).
- Provides users with capabilities to resolve complex problems, allocate resources, and analyze results.
- Hides from users the complexity of heterogeneous,distributed, high performance back end.

3.2.2 Multi-tier architecture

The system architecture is logically divided into three tiers: front-end, middle-tier and back-end.

- The front-end provides easy-to-use graphical user interfaces and programmatic interfaces to access the back-end resources.
- The Middle -tier provides a common service layer to support these multiple front-end clients and diverse back-end resources.

Figure 3.1: Grid Computing Environment

- The back-end consists of both Grid-based and non Grid based resources.

In a multi-tier architecture the front-end clients present graphical user interfaces (GUIs) in which user requests are accepted and forwarded to the correct middle-tier services for processing. The results of processing the user request are also displayed in the front-end client. These interfaces hide all the complexities involved in performing the needed actions in the middle-tier and back-end.

Web browser-based clients are ideal when middle-tier services are to be accessible from anywhere on the Internet without requering for any special software. Since all the processing occurs at the server side, there is sufficient latency involved in getting responses back from the server and the response time also depends on the networks characteristics.

Grid-HSI portal is a multi-tier system that allows integrating existing commodity components into a single, user friendly, web accessible system without compromising security of the computational resources.

3.2.2.1 Front-end

The front-end tier accepts user requests, forwards the request to appropriate services in the middle -tier, and displays the corresponding responses to user requests based on the results provided by the middle-tier. User requests are typically accepted through wizards based Graphical User Interfaces (GUIs). These interfaces hide all complexities involved in processing the request, performing necessary operations to satisfy the request, and generating results. Javascript is used for providing client-side processing while dynamic content is generated using servlets [23] and Java Server Pages JSP [24].

Portal Grid Interface: These interfaces allows users to enter the required input parameters (See Figure 3.2) for executing Grid services associated to each HSI algorithm implemented in Grid-HSI. The portal Grid Interface is implemented with servlets, which provides web-based GUI design for user interfaces. The Portal Grid Interface uses Java servlets hosted within a Tomcat servlet container environment. All requests to the Portal Grid Interface go through an Apache server, which forwards requests to the Tomcat using Apache JServ Protocol (AJP).

3.2.2.2 Middle tier

The Middle tier processes the user requests forwarded by the front-end, performing the necessary operations including accessing back-end resources and middle-tier services.

Data Broker: This component is a link between the Portal Grid Interface and the HSI Grid services. The Data Broker manages data related to the Grid services available on each resource so a match between local resources and user requests is met. When the Data Broker receives a user request it sends the request with information about the selected node to a Grid Service Client.

3.2.2.3 Back-end

The back-end consists of Grid-based and non-Grid-based resources, which includes services for data transfer, job submission, job monitoring, and specific Grid Service Clients.

Figure 3.2: Interface of Grid-HSI

HSI Grid Services: These services implement the HSI algorithms. For each service a grid service client is implemented so continuous access to Grid services will be performed through the associated client stub. Jobs are submitted by users through the Portal Grid Interface. A Master Manager Job Factory Server (MMJFS), implemented in GT3, executes the task in the remote resources indicated by the Data Broker, examines results of submitted jobs, evaluates information of resources, and so on. After the client stub receives the request from the Data Broker it proceeds to send the request to the node specified by the Data Broker. Table 3.1 summarizes the main HSI Grid Services.

Name	Description
CmeansClassifier	Generates the classification vector (.txt) using the C Means algorithm according to selected parameters.
FimClassifier	Generates the classification vector (.txt) using the feedback iterative algorithm according to selected parameters.
PcaReduction	Generates a new matrix of reduced dimensionality (.txt) using the Principal Component Analysis (PCA) according to selected parameters.
TxtPng	Converts a ClassificationVector.txt file that contains the result membership for each pixel on the image to a Portable Network Graphics (PNG) format file.

Table 3.1: HSI Grid Services

Figure 3.3 depicts the complete Grid-HSI architecture as described above. The Grid Infrastructure includes the local resources, the HSI Grid services and the associated clients stubs. Each HSI Grid service has a Grid Service Client associated to provide access through the Data Broker.

3.2.3 Functional Design of Grid-HSI

The Functional design of Grid-HSI system as depicted in Figure 3.4 is constituted by a set of components interconnected logically to accomplish the system objective.

The functional process is described as follows:

1. Initially the Portal Grid user using a web Browser sends a service request with classifier

Figure 3.3: Grid-HSI architecture

parameters through a HTML Form.

2. The Data Broker Servlet reads these parameters, defines which resource to use and sends such parameters to the Classifier Grid Client.
3. The Classifier Client sends these parameters to Classifier grid service in the selected node.
4. The Classifier Grid Service invokes its local Classifier Algorithm. This algorithm yields a result txt File and sends to the Classifier Grid Service an algorithm report.
5. The Classifier Grid Service sends back to the Classifier Grid Client the algorithm report and the result txt File Id.
6. The Classifier Client receives the algorithm report and the result File id, it proceeds to send these parameters to the Data Broker.
7. The Data Broker sends to the Server Result Displayer the algorithm report and sends to the Txt-Png Converter Grid Client the Result File Id.
8. Txt-Png Converter Grid Client sends Result File Id to Txt-Png Converter Grid Service.
9. Txt-Png Converter Grid Service invokes its Txt-Png Converter with Result File Id as parameter.
10. Txt-Png Converter reads from Hard disk Result Txt File wrote by the Classifier algorithm and processes it to get it in a Png file.
11. The Png File Id is sent back to the Data Broker although Txt-Png Converter Grid Service and Txt-Png Converter Grid Client.
12. The Data broker with this Png File Id invokes a Transfer File Client which, using a TCP Socket and performs the transference from the resource Hard Drive to Client Hard Drisk.
13. Finally, the Servlet Data Displayer receives Png File id from the Data Broker, reads the Png File from its hard Disk and sends the Png file to Web user.

Figure 3.4: Grid-HSI Functional Design

3.2.4 Infrastructure of Grid-HSI

The Infrastructure of the system is divided in three components (see Figure 3.5).

Figure 3.5: Infrastructure of Grid-HSI

Web Server: Provides access to end users through a web browser. The portal uses java servlets hosted within a Tomcat servlet container environment. All requests to the grid portal pass through an Apache server. Apache server forwards requests to the Tomcat using AJP protocol.

Infrastructure Server: Contains a database with information about all the available nodes for the execution of tasks in a given instant of time and about on the tasks that are already being executed in the Grid. The server takes charge of the administration of nodes that continually are added or eliminated on the Grid. This server finally, communicates with the portal grid to provide the available resources and to inform of possible changes.

Nodes: Resources available potentially as part of the grid for the execution of works. Each resources contains an activity demon (DA) that notifies to the Infrastructure Server the readiness of that node for the execution of remote tasks.

3.3 Implementation

3.3.1 Setting up the environment

The necessary steps for installing and setting up the environment are described in this section. The Globus Toolkit is a free software that can be obtained from [25]. We have worked with Globus Toolkit 3.0.2. The proper bundle has to be chosen according to the Operating System. The prerequisite software also needs to be installed.

- The Certificate Authority (CA) in the infrastructure is independent from the grid installation. For our grid environment we chose to have our own local grid CA, with self-signed certificates. The authentication of a grid-service request consists of two separate parts: host-authentication and user-authentication. In orden to run a remote grid service, the user must be accepted by the remote machines, and the grid user must be mapped to a local user on the remote machine. Therefore, two certificates are necessary: one for the user, and one for the host. The user certificate is placed in the ***.globus/*** directory of the user's home directory while the root-owned host certificate is located at ***/etc/grid-security/***.

  The following actions are necessary to create and sign the certificates.

  1. *Request the user certificate for the end user*: A user certificate is requested by issuing the grid-cert-request command.

     Next we present the output of the command:

     ```
     [root@x1 root]# su - itso
     [itso@x1 itso]$ grid-cert-request
     Enter your name, e.g., John Smith: griduser01
     Enter PEM pass phrase: <passphrase>
     ...(author omits lines)
     ```

     After issuing the grid-cert-request command, a file named usercert_request.pem is stored in the /home/griduser01/.globus directory.

  2. *Request the host certificate for the machines*: A host certificate is requested by issuing the grid-cert-request command with the -service and -host parameters. The host certificate

is requested in a similar fashion to the way the user certificate was requested.

3. *Send the certificates to the CA*: The certificate request is made by copying the unsigned certificate to the /CA/IN directory of the CA machines
4. *Have the CA sign the certificates*: In order to sign the certificates in the server we use the program "openssl", this is executed in the bash with the flags to sign the user and server certificates. This step helps to create an automatic script that is being developed. Next we present a example of the command:

```
openssl ca -config /CA/openssl.cnf
           -batch
           -policy policy\_anything
           -keyfile /CA/demoCA/private/cakey.pem
           -out /CA/OUT/node02hostcert\_cert.pem
           -in /CA/IN/node02hostcert\_request.pem
```

   Where the following flags mean:

   -config: Configuration file

   -batch: does not ask questions

   -policy: The CA 'policy' to support

   -keyfile: Name and path of the private key

   -out: Place and name to put the signed requested

5. *Retrieve the signed certificates from the CA*: The signed certificate is stored in the /CA/OUT directory of the CA machine, copy the approved request from CA to the nodes and users.

- Two security files need to be created. These files essentially contain a list of users that are to be given access to the Globus environment and located in the /etc/grid-security directory. These files are grid-map file and grim-port-type.xml.

  The grid-map file is allocated in /usr/grid security/. This file has a list of authorized users has to be updated.

  The grim-port-type.xml file is an XML document that maps the user ID to the Grid service name. Next, we show as example of /etc/grid-security/grid-mapfile

```
grid-mapfile
"/O=UPRM/OU=CISE/CN=griduser01" griduser01
"/O=UPRM/OU=CISE/CN=globus\_test" globus\_test
```

The lines in such a file are composed of the subject from the X.509 certificate identifying the user, between quotations, followed by user's common/local name separated between each other by a space.

- Testing can be performed using globus-run command once a gatekeeper has been initiated on a server machine and as soon as a proxy has been created on the local machine. This command is used for submitting jobs to the remote machine which is viewed as a resource in the grid environment.

3.3.2 Implementation of Grid Service

Writing and deployment a grid services implicate:

1. *Define the interface of service*: the operations that allow the service are defined through of GWSDL document, called Port Type. This dococument contains data about the interface, semantic and management of a request to the Web Service. A WSDL document has the following parts:

   <*definitions*>, in most of the cases, a file WSDL defines a service only.

   <*message*> and <*portType*>, describe what operations provide the service.

   <*binding*>, describes how the operations are invoked.

   <*service*>, describes where is the service and

   <*documentation*>, presents other information to the user.

   Next, we present the sections of the GWSDL document that defines the interfaces of the TxtPng Grid Service.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="TxtPngService"
    targetNamespace="http://www.globus.org/namespaces/2004/02/GridServices/
    TxtPngService"
```

```
    xmlns:tns="http://www.globus.org/namespaces/2004/02/GridServices/
    TxtPngService"
    xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
    xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/
    gridWSDLExtensions"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
```

The definition tag contains two important attributes:

- *name*: name of the GWSDL file.
- *targetNamespace*: The target namespace of the GWSDL file, all the PortTypes and operations defined in this GWSDL file will belong to this namespace.

```
<gwsdl:portType name="PngPortType" extends="ogsi:GridService">
    <operation name="startService">
        <input message="tns:startServiceInputMessage"/>
        <output message="tns:startServiceOutputMessage"/>
        <fault name="Fault" message="ogsi:FaultMessage"/>
    </operation>
    ...

</gwsdl:portType>

</definitions>
```

The <gwsdl:portType> tag has two important attributes:

- name: name of PortType.
- extends: Allows defined the PortType as an extension of an existing PortType. In our case, we extends from ogsi:GridService PortType, which all Grid Services must extend from.

Inside the <gwsdl:portType> we have an <operation> tag for each method in our PortType. The <operation> tag has an <input> tag, an <output> tag and a <fault> tag. These tags have a message attribute, which specifies what message should be passed along when the operation

is invoked (input message) and when it returns successfully (output message) or when and error occurs(fault message)

2. *Implement the service*: The operations described in the GWSDL document are implemented through Java class, when has to meet certain requirements.

   This example shows the implementation for TxtPng Grid Service:

```
package org.globus.GridServices.services.core.PngService.impl;
import org.globus.ogsa.impl.ogsi.PersistentGridServiceImpl;
import org.globus.GridServices.stubs.PngService.PngPortType;

import uprm.edu.*;

public class TxtPngImpl extends GridServiceImpl implements
TxtPngPortType {
    ...
    public void startService(String name) {
        this.imageName = name;
        this.height = 145;
        this.width = 145;
        System.out.println("Started...");
    }

    public void saveImage(String outputFile){
        try {...
            }
         catch (Exception e) {
            ...}
    }

    ...
 }
```

TxtPngImpl is a child class of GridServiceImpl. All Grid Services must be extended from the base class GridServiceImpl. This is a what is usually called the skeleton class, because it contains the basic functionality to all Grid Services.

We implement the methods specified in the service interface: startService, newImage, processImage and saveImage.

3. Configuring the deployment in Web Service Deployment Descriptor (WSDD): Put the Port Type and the service implementation together, and make them available through a Grid Services-enabled web server. The WSDD file contains information to be used by the server on how to publish the grid service. A WSDD file has the following elements:

   <service name>, describes where is the service.

   <parameter name="className"> indicates the java class that will be activated like Web-Service (TxtPngPortType).

   <parameter name="baseClassName">, indicates the class that provides the implementation of a grid service.

   <parameter name="schemaPath">, indicates the grid services container where the WSDL file for this service can be found.

   This example shows the WSDD file for a TxtPng Grid Service:

```
<?xml version="1.0"?> <deployment name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

    <service name="GridServices/core/TxtPngService/TxtPngService"
    provider="Handler" style="wrapped">
        <parameter name="name" value="TxtPngService"/>
        <parameter name="baseClassName"
        value="org.globus.GridServices.services.core.TxtPngService.
                impl.TxtPngImpl"/>
        <parameter name="className"
```

```
            value="org.globus.GridServices.stubs.TxtPngService.TxtPngPortType"/>
            <parameter name="schemaPath"
            value="schema/GridServices/TxtPngService/TxtPng_service.wsdl"/>
            ...
        </service>
    </deployment>
```

4. *Create a GAR file with Ant*: This GAR file is a single file that includes all the files and information that the grid services container need to deploy the service and make it available to the whole world. This is made with Ant tool [26]. A GAR target is available to archive all created components into an archive file that serves as the unit of deployment. The GAR target depends on several other targets that perform the following steps. 1) Converting the GWSDL file to WSDL file 2) Creating the stubs classes grom the WSDL. 3) Compiling the stubs classes, and 4) Compiling the service implementation.
5. *Deploy the service into a Grid services container*: This is also performed with the Ant tool. It unpacks the GAR file and copies the files into (WSDL, compiled stubs, compiled implementation, WSDD) key locations in the GT3 directory tree. It uses the deploy target from build.xml to deploy the generated GAR package into a Grid service hosting environment.

Figure 3.6 depicts the complete steps for implementing and deploying Grid Services.

3.4 Experiments and Analysis

For experimental purposes our local resources consists of a low cost commodity PC cluster consisting of eight nodes connected using a 100 Mbps Ethernet switch. Each node is an Intel P3-650 Mhz with 256 MB of memory running RedHat Linux 3.2.2-5.

Successfully every local resource can accomplish the requests sent by users with not regard to the sources of the request, and users can submit jobs to several nodes at the same time. The results obtained from the test cases for Grid HSI services match with those of the earlier C++ algorithms. Table 2 and table 3 show the results of the C-Means method with Euclidean distance and the Principal Component Analysis implemented using C++ language, respectively.

Figure 3.6: Steps for writing Grid Services

Table 3.2. Results C-Means method with Euclidean distance.

Number of Classes	Iterations	Bands Used	Execution Time (s)
5	4	220	41.75
5	5	220	50.593
5	6	220	59.13

Table 3.2: Results C-Means method with Euclidean distance

Table 3.3. Results Principal Component Analysis.

Number Components	Percent amount Energy	Bands Used	Execution Time (s)
3	90	220	9.50
5	90	220	9.74
7	90	220	10.00

Table 3.3: Results Principal Component Analysis

Our experiments aim in to evaluating the grid system accomplisment running C-means and Principal Components Analysis. In order to demonstrate the effectiveness of the proposed approach many test cases were used. We present two of these simulation cases. The first scenario the node that receives user request via internet (node02) is the same node that provided the grid service and executes the request because this node matches the available resources to the user request. In such a scenario we get that the grid system can complete the user request.

The container (node02) shows the classifier parameters and the results of execution. After the classifier service through classifier algorithm performs all the iterations a classification vector is returned. This classification vector contains the result membership for each pixel on the image. This classification vector is then stored on a directory of the container node as ClassificationVector.txt. Then, the TxtPng service through JAI API transform the ClassificationVector.txt on a Portable Network Graphics (PNG) format file. After the PNG File invokes a TCP Socket and performs the transference to Client Browser Hard Disk (See Figure 3.7).

The servlet Data Displayer receives the algorithm report and Png File from the Data

Figure 3.7: Results of container for one node scenario

Figure 3.8: Results of user interface for one node scenario.

Broker. Then, reads the Png file from its hard disk and shows the algorithm report and the png file to the web browser. The graphical output in this case shown in figure 3.8.

In the second scenario, the node that receives user request via internet (node04) is a different node (node02) that provides the grid service and executes the request because this node matches the available resources to the user request. In this scenario we also get that the grid system can complete the user request. Our container output is shown in figure 3.9 and the graphical output is illustrated in figure 3.10.

Figure 3.9: Results of container for two nodes scenario

The table 3.11 and table 3.12 show the total Execution Time of the request where the node that receives user request is a different node than the one which provides the grid service. We can see that the Execution Time obtained with a local C-Means Algorithm was a little faster than the obtained in a Portal Grid on all iterations. This is due to overtime generated by communications and servlets process.

In table 3.13 we can see a similar behavior as above. The local Principal Component

Figure 3.10: Results of user interface for two nodes scenario

Figure 3.11: Results C-Means Classifier (Euclidean distance discriminant and 5 classes)

Figure 3.12: Results C-Means Classifier (Euclidean distance discriminant and 6 classes)

Algorithm was a little faster than the obtained in a Portal Grid on several number of components; again, this is due to overtime generated by communications and servlets process.

Table 3.14 shows the time for each stage: Execution Algorithm Time, Transform Txt to Png, Connection Time by TCP to transfer PNG file, Send PNG File Time and the Total Execution Time. We can see again that the Execution Algorithm Time was just faster than the Total Execution Time.

In all cases the time overhead due the communications and servlets process is nor greater than 8.5%. This is not relevant if we take all advantage of using the proposed Architecture in terms of portability and accessability.

3.5 Summary

This chapter describes the design and implementation of Grid-HSI, a system prototype that provides users with a transparent interface to access computational resources and perform remotely hyperspectral imaging analysis through a set of Grid services.

Figure 3.13: Results principal Component Analysis

Figure 3.14: Results principal Component Analysis

This Grid based application, contrary to client/server approaches, provides the capabilities of persistence and potential transient process on the web. Thus, we can perform a chain of operations and be able to get the result of one operation and send it as a parameter to the next operation.

In all experiments the time overhead due the communications and servlets process is nor greater than 8.5%. This is not relevant if we take all advantage of using the proposed Architecture in terms of portability and accessability.

CHAPTER 4

STUDY OF PERFORMABILITY ISSUES IN GRID COMPUTING

4.1 Introduction

Usual metrics such as response time, resource usage, throughput and efficiency are not necessarily applicable to grids because grid performance is not characteristic to an application itself rather to the interaction between the application and the infrastructure. Performance tuning is more difficult due to dynamic environment, changing infrastructure, and heterogeneity of resources. While there exist practical solutions for performance evaluation of parallel and distributed systems , in most cases these techniques cannot be transferred directly to grids. In the last years the Grid Computing research communities has introduced novel approaches to performance monitoring and evaluation. In this chapter, we discuss important issues related to performability.

The remainder of this chapter is organized as follows:

Section 4.2 Surveys the most relevant performance evaluation tools for Grid environments.

Section 4.3 presents some critical issues.

Section 4.4 summarizes the issues discussed in this chapter.

4.2 A survey of Grid performance tools

A number of performance analysis tools have been developed to enable Grid Computing. Among then we can point at:

4.2.1 Network Weather Service (NWS)

The forecasting methodology used by the NWS assumes that each resource performance characteristic can be measured quantifiably [**?**]. Each resource can be described by a stream of performance measurements, and predictions of future measurement values are the quantities that are of interest. Another assumption is that performance measurements can be gathered non-intrusively. Finally, because the methods are time series based, the NWS forecasting method assume that the characteristics being measured have an instantaneous value that can be sampled at any given point in time. NWS have sensors that report the observed performance that a resource is able to deliver at the time a measurement is taken. A network link sensor reports periodic measurements of latency and bandwidth across a particular link. Measurement are taken as close to the application level as possible.

In NWS the Throughput is then calculated as the data size divided by the transfer time (4.1).

$$Throughput = data\_size/(data\_transfer\_time) \tag{4.1}$$

This resulting measure includes the overhead necessary to iniciate a TCP/IP communication stream, which can be significant. To calculate the effective Throughput rate, the latency is subtracted from the time recorded for the data transfer, and the result is used as the actual time to transfer the data (4.2).

$$Effective\_throughput = data\_size/(data\_transfer\_time - latency) \tag{4.2}$$

NWS presents the following limitations:

The model for communication time ((datasize/bw) + latency) is impractical for Grid environments. The current trend indicates that while latencies are bounded below by the speed of light, network bandwidth increases at an exponential rate. For example the TeraGrid platform has established a 40GB/sec dedicated link between SDSC and NCSA. The expected network latency is in the 100m/s range. Thus, one third of the time to transfer 1 GByte of date is due to network latency. Therefore, the latency component must be taken into considerations in the model for Grid platform performance.

4.2.2 REsource MOnitoring System (Remos)

The Remos (REsource MOnitoring System) allows network-aware applications to obtain relevant information about their execution environment. The major challenges in defining a uniform interface are network heterogeneity, diversity in traffic requirements, variability of the information, and resource sharing in the network. The Remos system serves as a foundation for a range of application specific approaches to dealing with network resources and their changes [**?**]. Remos aims to provide resource measurements across a wide range of network architectures environments and implementations. It uses a logical topology to capture the network information ready to be used by any network-aware applications as needed. Network measurements are performed by an independent service and hence simplifies the task of the application developer since the task of collecting network information has been moved to network developers.

By measuring the bandwidth and latency between sites, Remos has a component that determine the performance of the links connecting the network in a manner similar to the technique used by NWS.

The equation to determine the transmission time of a data is (4.3) :

$$Ttransmission = (datasize/bw) + latency \tag{4.3}$$

The bandwidth of an alternative path is the minimum of the two individual bandwidths and the latency is the sum of the individual latencies. For small amounts of data, as for text, the

latency is the dominant factor, whereas the bandwidth becomes more important with the increasing data size. The current values of the available bandwidth and the latency can easily be gathered using Remos.

Remos presents the following limitations

- Techniques used by Remos are attractive since the resultant logical topology could clearly shows where application flows could be competing with each other for bandwidth. However, the methods used in Remos could results in heavy usage of network resources and though much information is created it may not all be relevant to network aware applications.
- The design of Remos ignores a number of important network properties. It does not deal with multicasting, or with networks that provide guaranteed services. Both of these are important features that would be of interest to Grid applications.
- The static features of the network may have to be collected through SNMP (Simple Network Management Protocol). But dynamic bandwidth and latency information are somewhat harder to track. The mechanisms used by Ramos for colleting that information is through benchmarking. Benchmarking can be used to collect anny type of information. However, benchmarking is time consuming and introduces a significant amount of overhead in the network

4.2.3 Networked Application Logger (NetLogger)

Using NetLogger, distributed application components are adapted to produce timestamped logs of relevant events at different critical points of a distributed system. Events from each component are correlated, which allows one to characterize in detail the performance of all aspects of the system and network. [**?**]. The events are correlated with the system's behavior in order to characterize the performance of all aspects of the system and network in detail during actual operation. The monitoring is designed to facilitate identification of bottlenecks, performance tuning, and network performance research. It also allows accurate measurement of throughput and latency characteristics for distributed application codes. NetLogger includes a tool for analyzing and monitoring events based on visualization of the timestamp correlated event data. This performance analysis requires monitoring data for hosts (CPU, memory, disk), networks (bandwidth, latency, route), and the FTP

client and server programs.

The NetLogger Toolkit itself consists of four components: an API and library of functions to simplify the generation of application-level event logs, a set of tools for collecting and sorting log files, an event archive system, and a tool for visualization and analysis of the log files. In order to instrument an application to produce event logs, the application developer inserts calls to the NetLogger API at all the critical points in the code, then links the application with the NetLogger library.

NetLogger presents the following limitation

- NetLogger can dynamically provide real-time access to monitoring information with minimal system perturbation. However, the tool is intended for control and monitoring of the state of the whole grid rather than for analyzing performance of single application.

4.3 Grid Performability

Performance usually refers to quality of service, assuming the system is correct. Dependability, on other hand, refers to the ability of a system to perform appropriate when a failure of the system occurs. Thus, dependability include reliability(continuity of failure-free service), availability (readiness to serve), safely (avoidance of catastrophic failures), and security (prevention of failures due to unauthorized access). As system become complex(e.g. Grid Computing), measures of performability are needed to address issues of both performance and dependability simultaneously.

At system start all components are assumed to be operational and the system will operate at maximum performance. When a component faults, the system should reconfigurated itself and restart its activities subject to degraded performance.

Let S denote the set of possible configurations. The stochastic process $X = \{x(t), t \geq 0\}$ on S describes the structure of the system at time t. This process is defined in such a way that knowing *x(t)*, we must be able to know which components are up at time t. The performance of the system when state $i \in S$ is denoted by r_i=*r(i)*.

The real function $r : S \rightarrow \mathbb{R}$ called reward rate function on the state space S. In specific context r_i may signify a performance metrics such as throughput or efficiency. Thus, the value r_i somehow summarizes the performance of the system in the structure state i.

For $i \in S$, let Π_i denote the steady-state probability of residing in state i and $p_i(t)$ the probability of residing in i at time t. The steady-state performability is defined as

$$SSP = \sum_{i \in S} \Pi_i r_i \tag{4.4}$$

The trasient analog of this measure is

$$TP(t) = \sum_{i \in S} p_i(t) r_i \tag{4.5}$$

Thus, SSP is the expected asymptotic rewards, while $TP(t)$ is the expected rewards at time t.

If we define the cumulative performability as

$$CP(t) = \int_0^t r_{x(t)} ds \tag{4.6}$$

Which is a random variable, then the performability is defined as

$$PrCP(t) \leq y = D(t, y) \tag{4.7}$$

In classical performability analysis X is a continuous-time Markov chain. Because of the heterogeneity of Grid Computing environments. We have the following challenging problem in Grid performability analysis.

- The models usually assume that in a particular state $i \in S$ at time t, the system performs with rate r_i. However, the rates may also depend on the global time, thus having a reward rate function $r_i(t)$ for every state i. This might be the case in Grid Computing environments.
- We face the problem of deciding if a reward function is well defined for a specific Grid environment: Stochastics Active Networks have been used to define rewards function in a tradictional distributed systems. It is important investigate if this type of models apply to Grid Computing environments.

4.4 Summary

Grid computing system is different from conventional distributed computing systems by its focus on large-scale resource sharing, where processors and communication have significant influence on grid computing performance. Most previous research on conventional small-scale distributed systems ignored the communication time and processing time when studying the distributed program performance, which is not practical in the analysis of grid computing systems. The presence of communication heterogeneities reduces the accuracy from the communication models used to optimise collective communications in wide-area networks.

The issue of predicting resource performance is essential to Computational Grid research. Not only is it critical to effective program and system design, but also dynamic schedulers and fault diagnosis tools requires on-line access to prediction data as part of the Grid infrastructure.

In this chapter, we have discussed relevant tools for performance analysis of Grid environments, identifies advantages and limitations of those tools, and finally presented how the concept of performability relates to Grid environments.

CHAPTER 5

Conclusions and Future Work

In this thesis we have presented the design and development of a Service Oriented Architecture - based Grid platform suitable to hyperspectral imaging analysis. The result is Grid-HSI, a system prototype that provides users with a transparent interface to access computational resources and perform remotely hyperspectral imaging analysis through a set of Grid services. The system is based on Open Grid Service Architecture (OGSA) and implemented on the top of the Globus Toolkit 3.0 (GT3).

While the Grid-HSI system is a first prototype, our results suggest that this is a new source of computational power that is accessible and applicable to the remote sensing problem set.

Among the main features of Grid-HSI we can list:

1. Grid HSI is fully based on OGSA architecture providing a functional Grid based application system.
2. The Grid-HSI Grid Services provides the capabilities of persistence and potential transient process on the web.
3. Multiuser, make it possible for several users through a web browser to use the same Grid Service in the same node at time.

5.1 Future Work

There are several areas which might be improved by subsequent work:

- The Grid-HSI services that implement the hyperspectral analysis algorithms focus on using feature selection and feature extraction to reduce the dimensionality of the problem or seek representations with improved properties for the rapid analysis. Another approach is the application of the distributed learning techniques as a divide-and-conquer method. Distributed learning can be done by training classifiers on distinct subsets of data. The result of the combination of classifiers is referred to as ensemble of classifiers. The Grid-HSI infrastructure opens an avenue for research on ensembles of classifiers.
- It is important that Grid resources be used efficiently. Executing the application on a large scale can have as a result that a HSI grid service job has to be slit up into many subjobs. To find out how the Grid-HSI application scales using very large Grids, the efficiency has to be investigated. Further more is needed to provide appropriate performance metrics for Grid environments and mechanism to guarantee scalable systems.
- Fault-tolerance is a critical issue in grid computing. It includes fault-detection and failure recovery. As the number of machines in a Grid grows the probability that there is a fail in a particular node or network connections increases dramatically. It follows that when a service is delivered by composing several other services, possibly on several different machines, the probability that it completes decreases as the number of machines increases. In order to make Grid-HSI highly reliable it is required constant attention and care to handling failure.
- An issue that we need to investigate is the interaction between the fault tolerance mechanism and the data consistency protocols. This is a very challenging problem as the scale of Grid Computing environments continues to increase.

We believe that the deployment of Grid-HSI will facilitate the investigation of more fundamental problems related to performance, fault tolerance and extension of algorithms.

BIBLIOGRAPHY

[1] C. Kesselman I. Foster and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications*, 15(3), 2001.

[2] P. Swain and S. Davis, editors. *Remote sensing: the quantitative Approach.* McGraw-Hill, 1993.

[3] S. Mazer and M. Martin. Image processing software for imaging spectrometry data analysis. *Remote Sensing of the Environment*, 24(1):201–210, 1998.

[4] C.L. Carvajal-Jimnez W. Lugo-Beauchamp, K. Cruz and W. Rivera. Performance of Hyperspectral Imaging Algorithms using Itanium Architecture. *The IASTED International Conference on CIRCUITS, SIGNALS, AND SYSTEMS*, december 2004.

[5] D. Rivera-Gallego. Feedback in Pattern Recognition.

[6] D. Kaeli M. Velez-Reyes E. Rodriguez-Diaz L. Santos-Campis C. Santiago H. Velazques E. Arzuaga-Cruz., L. Jimenez-Rodriguez. and Alexey Castrodad. A MATLAB Toolbox for Hyperspectral Image Analysis. *IEEE International Geoscience and Remote Sensing Symposium*, 2004.

[7] B. Moon G. Fann, D. Jones. E. Jurrus and K. Perrine. A Parallel Computational Environment for Imaging Science. *47th International Symposium on Optical Science and Technology*, july 2002.

[8] V. Patel H. Bhatt and A. Aggarwal. Web enabled client-server model for development environment of distributed image processing. *7th International Conference on High Performance Computing*, december 2002.

[9] C. Huallparimachi and D. Rodriguez. Web enabled client-server model for development environment of distributed image processingDesign and Development of a Java-based Distributed System Tool for Synthetic Aperture Radar Image Analysis. *IASTED International Conference on Computer Science and Technology*, may 2003.

[10] I. Foster and C. Kesselman, editors. *The grid: blueprint for a future computing infrastructure.* Morgan Kaufmann Publishers, 1998.

[11] I. Foster J. Frey S. Graham-C. Kesselman T. Maquire T. Sandholm-D. Snelling S. Tuecke, K. Czajkowski and P. Vanderbilt. Open grid services infrastructure (ogsi) version 1.0. 2003. available at http://www.ggf.org/ogsi-wg.

[12] M.P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12, december 2003.

[13] D.J. Miller V.P. Holmes, W.R. Johnson. Integrating Web service and grid enabling technologies to provide desktop access to high-performance cluster-based components for large-scale data services. *Simulation Symposium, 36th Annual*, pages 167 – 174, march 2003.

[14] W3C/IETF URI Planning Interest Group. Uris, urls, and urns: Clarifications and recommendations 1.0. September 2001, http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/.

[15] The UDDI Community. Uddi universal description discovery and integration. The UDDI Community, http://www.uddi.org.

[16] Word wide web Consortium. Web services description language wsdl 1.1. March 2001, http://www.w3.org/TR/wsdl.

[17] Word wide web Consortium. Soap version 1.2 part 0: Primer. December 2002, http://www.w3.org/2002/ws/.

[18] Internet Engineering Task Force. Hypertext transfer protocol - http/1.1. rfc 2616. http://www.ietf.org/rfc/rfc2616.txt.

[19] Word wide web Consortium. Extensible markup language (xml) 1.0 (second edition). October 2000, http://www.w3.org/TR/REC-xml.

[20] J. Nick I. Foster, C. Kesselman and S. Tuecke. The Phisiology of the Grid: An Open Grid Serivices Architecture for Distributed Systems Integration, Technical report, Open Grid Service Infrastructure WG. *Global Grid Forum*, june 2002.

[21] Borja Sotomayor. The globus toolkit 3 programmer's tutorial. http://www.casa-sotomayor.net/gt3-tutorial/.

[22] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 1997.

[23] Word wide web Consortium. Word wide web consortium. http://www.wc3.org/.

[24] H. Bergsten, editor. *JavaServer Pages, 2nd Edition* . O'Reilly, 2002.

[25] The Globus Alliance. http://www.globus.org/.

[26] The Apache Software Foundation. The apache ant proyect. http://ant.apache.org/.