Remote Data Service Installation on a Grid-enabled Java Platform

Eduardo Blanco, Yudith Cardinale, Carlos Figueira, Emilio Hernández
Universidad Simón Bolívar, Departamento de Computación y T. I.,
Apartado 89000, Caracas 1080-A, Venezuela
{eduardo,yudith,figueira,emilio}@ldc.usb.ve

Robinson Rivas, Marta Rukoz
Universidad Central de Venezuela, Centro de Computación Paralela y Distribuida
Apartado 29065, Caracas 1040-A, Venezuela
{rrivas,mrukoz}@ccpd.ucv.ve

Abstract

We define a framework for remote service installation, called SIMG (Service Installation Metaservice on Grids). Its main goal is to allow grid users to define and safely install their own data filters on remote data servers. Once installed, these filters may be used like the predefined services and may be available for a broader community of users. SIMG is implemented in suma/g, a Globus-enabled platform for remote execution of Java bytecode. We also present our experiences using SIMG for defining and remotely installing two complementary services in a Content-Based Image Retrieval system.

1. Introduction

Emerging applications which process large data collections promise dramatic progress in scientific discovery. Some projects produce and analyze very large data sets that may span millions of files or data objects [7]. Data intensive applications are increasing in scope and sophistication to perform scientific data collection and analysis on a scale never before achievable.

Such amounts of data can not be downloaded for local processing; instead, they are processed either on the server side or in a distributed platform. A trend in large-scale scientific data analysis is to exploit computational, storage and other resources located at multiple sites, and to make those resources accessible to the scientists through a safe and virtualized environment. Grids [5] provide a suitable platform, offering users the possibility of using large-scale controlled environments across different organizational boundaries. In particular, Data Grids [2, 12] make it possible to access many large data repositories, integrating them in a highly dynamic distributed environment. In current grid developments and applications, data providers and data processing agents are usually closely related, e.g. under the umbrella of the same project [14, 21]. However, there will likely be an evolution towards a situation in which data providers and data consumers may belong to different virtual organizations [10]. Research groups around the world will be interested in processing data generated by a few data providers owning highly sophisticated scientific equipment.

In current distributed platforms, such as the Web, data services are usually defined by the data providers. These providers define the data structure and data access methods. As a result, data services often do not adapt accurately to the necessities of multiple and multidisciplinary consumers (clients), due to the diversity of requirements. The data may be analyzed in many different ways, depending on the discipline, methodology or modeling environments, among other kinds of research perspectives. A particular client (e.g. a research group) may be interested only in a small subset of the data, which may not be available as such. The clients may find themselves in the situation that they should download the whole of the data and filter out the data they do not need. For instance, a satellite image provider may offer complete representations of a large area, with many attributes of information and many images per hour, while a particular client may be interested in a small subarea and a lower amount of attributes. It is hard to think that a particular set of data services can satisfy the requirements of all users for the resolution of different problems. Parameterized services for selecting particular data subsets may be useful but insufficient, because it is diffi-
Motivated by these considerations, we have focused our efforts on designing and producing a framework that allows clients to enhance the accessibility to large data sets. This is a Java-based framework, called SIMG (Service Installation Metaservice on Grids), whose goal is to allow clients to define and install their own data filters on the server side without the direct participation of the data providers. Once installed, these new data filters may be used like the standard, predefined services. They will be available, not only for the user who installed them but also for a larger community of users. The key of our proposal is that the data provider may not be involved in the definition of data services, except for the definition of a service for the access to raw data.

Installation of a new service may be associated to a significant amount of data processing. For instance, a big data repository may have to be swept in order to build an index of relevant segments, which in turn may be the base of a new data service. In this case, processing takes place either in the server side or in a distributed platform (e.g. a grid) the data server belongs to. With SIMG it is possible to define the preprocessing necessary when a new service is installed. SIMG was implemented on a grid platform, called SUMA/G [8], that transparently executes Java bytecode on remote machines. The services installed with SIMG are Java objects whose methods constitute the service interface.

In summary, the development of SIMG addresses three key problems:

- Data publishing and discovery: The idea is that the data providers publish a basic interface for processing raw data.
- Providing a (meta)service to allow users to install their own services in the remote data repository
- Providing a safe platform on the server side for (1) executing applications that access and process the raw data and (2) harbour the service code and the intermediate data structures that may be generated.

In this work we describe the approach used by SIMG for solving these problems, and its implementation in SUMA/G. We also describe our experiences defining Image Retrieval Services which use Content-Based Image Retrieval (CBIR) techniques.

The rest of the paper is structured as follows. Section 3 describes SIMG’s functionality and implementation on SUMA/G. We present an example of using SIMG in section 4. Section 5 discusses some related works. The last section summarizes the conclusions and future work.

2. SUMA/G overview

SUMA (Scientific Ubiquitous Metacomputing Architecture)\(^1\) [7] is a grid platform that transparently executes Java bytecode on remote machines. SUMA middleware was originally built on top of commodity software and communication technologies, including Java and CORBA. A recent reimplementation called SUMA/G [8] is based on Grid Services (Globus services) [19]. By reimplementing SUMA components on top of Grid Services we can connect to deployed grids, while keeping SUMA execution model unchanged and independent from the evolution of the Grid Services. Additional functionalities are inherited from Globus, such as the I/O services.

As an execution platform, SUMA/G offers the Java execution model, and mpiJava for parallel programs. Classes and local files are sent on demand from the local machine (where the client is located) to the actual execution platform (where an Execution Agent is running) chosen by the SUMA/G core.

2.1. Implementation of SUMA/G on Globus

Globus is distributed as a toolkit that includes basic services and libraries for resource monitoring and management, plus security, file management, job submission, and communication. There has been increasing interest in supporting Java within the grid [20, 18]. Its GT3 version is the first full-scale implementation of new Open Grid Services Architecture (OGSA).

Current implementation of SUMA/G on top of Globus services is based on the Java CoG Kit. The Java CoG Kit defines and implements a set of general components that map grid functionality into a commodity environment. Not only it enables access to the Globus services, but also gives the benefit of using the Java framework as the programming model. According to the Java CoG Kit components categorization, SUMA/G mainly uses the Low-Level Grid Interface Components, which supply mappings to commonly used Grid Services (information services, resource management services, data access services, etc).

\(^1\) http://suma.ldc.ush.ve
The key Globus services, from the point of view of suma/g are:

- The Grid Security Infrastructure (GSI): it defines a common credential format based on X.509 certificates and a common protocol based on TLS/SSL. GSI policy allows a user to authenticate just once per computation, at which time a credential is generated that allows processes created on behalf of the user to acquire resources without additional user intervention. The GSI is used to enhance the user authentication and authorization in suma/g, as well as a mechanism for including all suma/g components in the grid security space.

- The Monitoring and Directory System (MDS): it enables grid application developers and users to publish their services. It is used by suma/g to register its execution platforms (active Execution Agents), and its services. The suma/g Scheduler queries the MDS to find a suitable platform when it receives a request for a particular job execution. With this service, resources that belong to a Globus grid can be added to the suma/g resources. suma/g has its own mechanism for submission and allocation of resources, but we expect to achieve a greater integration of suma/g with GRAM (Globus Resource Allocation Manager).

- The Globus Heartbeat Monitor (HBM): it consists of a simple, uniform and reliable mechanism for obtaining real-time information about the grid structure and status. This mechanism allows suma/g Execution Agents to post information about their state, and suma/g core to receive that information in order to make scheduling decisions.

The suma/g architecture is depicted in Figure 1.

3. SIMG on SUMA/G

In this section we describe SIMG’s functionality and implementation on suma/g. Our final goal is to provide support for evolved and customizable access to data on grid platforms. We have developed mechanisms that allow users to define, implement, and remotely install their own services according to particular requirements on data grid-enabled repositories. SIMG allows for a smooth transition from service development to deployment, since services can be locally implemented and tested as local packages, for a later seamless deployment. For instance, if class “service” is invoked by class “application”, the execution of local command “java application” will load the class “service” on demand, as usual in Java. If class “application” is executed on the Grid (using shell command “suma execute service”), class “service” is loaded on demand from the local directory to the worker node where “application” is running. Once tested, class “service” can be finally deployed on the Grid (using “suma install service”). Note that these mechanisms can be exploited in many other ways different from data filtering, such as for remote installation of numerical programs. However, in this work we focus on its application on providing enhanced data services. We define below services and outline access control handling in SIMG.

Services are specified by:

- a service name
- a list of services that the new service depends on (requirements)
- an API
- a documentation (in current implementation, a javadoc file)
- an implementation, i.e., a set of methods that actually implements the new service

Data filter installation often requires the execution of a constructor method, which may build additional data structures, such as indexes that point to the original raw data. These data structures will reside on a server in the grid, probably the same data server, which means that the data service not only consists of giving access to the raw data but also providing space for hosting intermediate data structures.

Access control. suma/g uses GSI support for user authentication and authorization. This is achieved through the creation of a time-stamped proxy, based on the user’s private key, to interact with suma/g, which has the same access restrictions that
any other Globus service has. In Globus, users are granted access if they are mapped to a registered virtual organization. Within a virtual organization, which is handled at the Globus level, there are three types of user groups, handled internally by suma/g:

- **Data owners**, who are allowed to register new data resources
- **Service installers**, who are allowed to register and allocate new services
- **Users**, who are allowed to access registered services, on a per-service basis.

Thus, the same certificate is used for virtual organization mapping and for user type mapping within suma/g.

### 3.1. Service installation

The sequence of steps to install a service is depicted in figure 2. To assist users, SIMG delivers a basic support that includes: users authentication and authorization; obtaining information about services like, for instance, currently registered and installed services, available services, documentation; obtaining a service's API; eliminating services.

The main steps to install a service are described below.

1. **Data publishing** At the starting point, there is a data repository (e.g., a satellite images database), whose owner wishes to make available to other users on the grid, including letting them add new data filters, based on image processing methods. An agreed upon data interface (for instance, a subset of SQL commands, tailored to the knowledge domain) is installed at the repository server. The common data interface makes it possible to build services independently of the raw database format. Finally, the data owner registers the resource within MDS. As a result, the following registers appear in MDS:

- A new data service, including its specification and an API.
- A resource (with an active Execution Agent) providing both the new service and the capability of accepting installation of new services, possibly relying on already installed services.

2. **Service registration** An authorized user (service installer) can add new services to the grid, based upon others already registered. When a service installer submits a new service, it is registered in MDS; the packages, API and documentation of the service are stored in the core of suma/g.

3. **Service allocation** A service allocation request implies querying MDS to find all resources that meet the service requirements (e.g., basic services needed by the submitted service). If available, an appropriate resource is selected; the packages are uploaded to the resource from the core, then installed. Finally, the service is started. Decoupling registration from allocation makes it possible to install the service on other resources after registration.

### 3.2. Using the services

Once a data access service has been installed, it is ready to be used. The users can query SIMG about all available API’s and services already allocated. The users could develop their Java applications based on the service API and execute them in the grid; suma/g locates the appropriate resource offering the requested service.

4. **An Experience: CBIR Services on SIMG**

For this work, we have developed a Java based Image Retrieval Service, which uses Content-Based Image Retrieval (CBIR) [9, 17] techniques. It is part of the HUYA CBIR system [16] developed at the UCV 2, and offers users the possibility of image indexing for

2 http://ccpd.ciens.ucv.ve/~huya
Further retrieval based on contents, usually color and texture similarity. Each image is indexed by extracting a set of numeric values (called feature vector); similarity comparison between images is done by calculating an appropriate distance on their respective vectors. The images database could be indexed by different CBIR techniques, hence there could be several indexes over the same database.

The Image Retrieval Service (IRS) is divided in two main modules: the first one allows to extract the feature vector of each image in the database and to construct an appropriate index according to a chosen CBIR technique. The second one allows to Search for Image Similarity (SIS) based on a given example image.

In the following sections, we describe how two SIS services, called SIS-9R and SIS-CR, are installed in SUMA/G using SIMG. Each service is based on a different feature extraction method: Nine-regions for SIS-9R, and Circular-region for SIS-CR. In both methods, the images are divided in regions, each one represented by its average color vector in the three-dimensional space (R, G, B). The two methods differs in the way they define the regions. In Nine-region, the image is divided in a matrix of 3 × 3 rectangles of the same area. In Circular-region a circular region is defined in the center of the image, with a 5th of the total area, and four additional areas in an “X” shape around the circle. Figure 3 shows an example of these two methods. Note that for Nine-region method there are 27 values on the feature vector (9 regions × 3 colors), while Circular-region has 15 values (5 regions × 3 colors).

**Figure 3. Examples of Nine-region and Circular-region methods**

### 4.2. Service registration

For each of SIS-9R and SIS-CR, a services installer provided a service specification (i.e., name, requirements, API, documentation, and implementation) to SUMA/G. For example, the following interfaces define the basic API for the SIS-9R service, which are obtained for SUMA/G during the “Service registration”:

```java
// Interfaces definition for SIS using Nine Regions method.
interface SIS-9RinitInterface {
    public init(URL database) throws InvalidURLException;
}
interface SIS-9RsearchInterface {
    public Vector searchNearestNeighbor(URL baseImage, int n) throws InvalidURLException;
}
```

### 4.3. Service allocation

For each of SIS-9R and SIS-CR, a services installer asks SUMA/G for a resource to allocate the service which, in our testbed, is the EA at UCV. The implementation packages are downloaded to the EA, and the service is initiated, fetching the images and constructing the respective index. When both allocation processes finish, SIS-9R and SIS-CR services become available for users in SUMA/G. This is an example of the initiation of SIS-9R at the EA:

```java
// When allocation is done, the service should be started. The init method should be called by the EA // during the "Service Allocation". In this moment, // an index file is created.
...
SIS-9RinitInterface service = (SIS-9RinitInterface) new ImageService(); try { service.init("http://..../images_database"); } ...
```

### 4.4. Using the services

Once the services have been allocated, users can search for similar images using the indexes already created. This operation is allowed for every user with execution permissions in SUMA/G and is done following the SUMA/G execution model. It means that users can invoke the execution of an application using preinstalled services just by typing “sumag application”, in HTTP servers, and accepting remote installation of services. In our testbed, the EA was started on a cluster of PCs at the Universidad Central de Venezuela (UCV) campus, while the SUMA/G core ran on a cluster of PCs at Universidad Simón Bolívar (USB) campus.
the same way they execute that class locally (by typing “java application”).

An example of Java source code for the services invocation in a user application is included below.

```java
//Finally, a user application can get all similar //images by giving an image example. The application //will be executed with a simple command //like: sumag Execute application_name
...
SIS-9RsearchInterface service =
   (SIS-9RsearchInterface) new ImageService();
try { Vector results =
   service.searchNearestNeighbor(myImageURL,20);}
...
```

Figure 4 shows the results of using both SIS-9R and SIS-CR services with the same query image. Note that "similarity" between base image and result images depends exclusively on the kind of feature extracted from images and the distance formula used. New feature extraction algorithms shall produce different result sets.

Using SIMG, another user located at a different site, could register and allocate another SIS service that uses a different indexing method or a more sophisticated service that uses the already allocated services, making it available to other users. An important consequence is that users can invoke the services closer to their needs, with just a single interface.

5. Related work

Standard component frameworks, such as J2EE, are not designed for the deployment of services by third parties, nor are they usually grid enabled.

A number of tools have been proposed to help handling services and component deployment on grids. Some of these tools are intended for grid administrators. For instance, the IBM Grid Toolbox [6] enhances a Globus system by offering a Web-based management system for common administration tasks, including services building, packaging and deployment, monitoring, etc. Quattor [15], addresses the remote fabric installation and setup, including nodes OS installation from scratch, then required basic grid services components uploading, configuration and launching. It leverages on LCFG [4], a non grid system used to install clusters and networks of computers with Linux. RMS [?] is also a grid administration tool, developed for software management in large distributed environments, which has been used in grid systems. It is based on mobile agents, an interesting technique for handling interoperability and complex, distributed components administration.

DistAnt [11] and GridAnt [3] specifically address the problem of helping users deploy their applications on the grid resources. They extend Ant [1] with grid specific tasks, which users can employ to define their appl-

plication’s deployment workflow (e.g., compiling; authentication; resource location; code and data packaging, transport and unpackaging/installation, etc.). These tools operate at a higher level than SIMG; however, they are oriented to isolated applications deployment, instead of the composable services deployment provided by SIMG. We believe these tools (DistAnt and GridAnt) can be combined with SIMG to leverage on the basic stackable services provided by SIMG, and the grid services access facilities automated by Dis-
tAnt and GridAnt.

H2O [13] shares several goals and concepts with SIMG. It is a Java based metacomputing system, with a flexible architecture built on a software backplane supporting component-based. Services can be plugged and made available to users by resource providers, third-parties developer or clients. A number of service properties (e.g., access control, priority) are set by providers. It uses its own component communication middleware (RMIX). In contrast, SIMG relies on standard component communication middleware (CORBA) and Globus services. Also, SIMG allows for a smoother transition from service development to deployment, since services can be implemented and tested as local packages, for a later seamless deployment.

6. Conclusions and Future Work

SIMG provides an adequate framework for user-customizable access to large data repositories, in the context of data grids. This framework is flexible and secure. It is flexible because it offers mechanisms that allow users to define and install their own data filter services on remote data repositories. It is safe because it relies on the grid security infrastructure, based on X.509 certificates. Additionally, SIMG is implemented in Java, which provides portability and security mechanisms for execution of foreign programs, such as the Java sandbox. Support for SIMG has been implemented in SUMA/G, and successfully tested for Content-Based Image Retrieval services. The proposed mechanism is useful in contexts in which a large amount of data needs to be processed once and then used many times. This is often the case of image databases, which additionally are usually very large and not suitable for downloading and processing at the users’ site. Different users or research groups may install their own data filters on the same data server, accessing in this way the same raw data through different perspectives or visions. Ongoing work aims at adapting SIMG to support the implementation of high level information retrieval systems, based on knowledge perspectives and ontologies.

References


