An Extensible Resource Discovery Mechanism for Grid Computing Environments[1]

Tania Gomes Ramos and Alba Cristina Magalhaes Alves de Melo
Department of Computer Science, University of Brasilia (UnB), Brazil
Campus Universitário – ICC Norte – Subsolo – Cx Postal 4466
{tania, albamm}@cic.unb.br

Abstract

Grid computing is emerging as a new infrastructure to provide collaborative and secure resource sharing over multiple geographically distributed organizations. In this scenario, resource discovery is a very important component, since it is responsible to retrieve information about the resources that compose the grid. Traditionally, the kind of data to be retrieved by resource discovery mechanisms is statically defined. In a highly heterogeneous and dynamic environment such as a grid, statically defined searches are usually inappropriate. In this paper, we propose and evaluate an extensible resource discovery mechanism for grid systems, where the basic resource information retrieval can be extended to include user-defined specific resource searches. Our experimental results show that the proposed mechanism is able to incorporate new searches to our grid resource discovery service in reasonable time.

1. Introduction

The popularity of the internet made possible the interconnection of millions of powerful machines in a global scale. Several measures were made which state that, most of the time, the majority of these interconnected machines remain idle. This led to the idea of metacomputing [2], which proposes the creation of a supercomputer by taking advantage of the idle cycles of the machines connected to the internet.

Grid computing can be considered as an evolution of metacomputing where not only the computing power is shared, but also several other types of resources such as data, software and specific hardware [6]. The main goal of grid computing is to enable collaborative and secure resource sharing over multiple organizations which are geographically distributed.

In this scenario, resource management plays a fundamental role since it is responsible to discover resources, allocate them to tasks and monitor the task execution, among others [11]. In this paper, we will deal only with resource discovery.

Resource discovery mechanisms are responsible to retrieve information about the resources that compose the grid. Traditionally, the data to be gathered by the resource discovery mechanism is statically defined. In a highly heterogeneous and dynamic environment such as a grid, statically defined searches are usually inappropriate.

In this paper, we propose and evaluate an extensible resource discovery mechanism for grids, where the basic information retrieval can be extended to include user-defined specific searches. The proposed mechanism was designed as a Globus 3 [12] grid service.

Our experimental results show that the proposed mechanism is able to incorporate new searches to our grid resource discovery service in reasonable time.

The remainder of this paper is organized as follows. Section 2 briefly describes grid computing. Resource discovery in grids is discussed in section 3. Section 4 presents the design of our extensible resource discovery mechanism. Some experimental results are presented and discussed in section 5. Finally, section 6 concludes the paper and presents future work.

2. Grid Computing

The term Grid Computing was conceived in the mid-1990s to denote a new infrastructure of distributed computing for the scientists and engineers in a more advanced scope. This name was inspired by the electrical power energy because of its pervasiveness, ease of use and reliability [6].

Grid Computing technologies and concepts were initially developed in order to enable resource sharing between scientific institutions with common projects, who needed to share data, software and computational power. In developing applications for the grid, it is essential to have an unified middleware to provide a transparent interface to the underlying protocols.

The Globus Toolkit [5] emerged in 1997 as an open source project and quickly became a *de facto* standard for

[1] This work was partially supported by FINATEC/UnB/Brazil

grid computing infrastructure. Globus defines and implements a set of protocols, APIs and services used by hundreds of grid applications all over the world. Moreover, it worked as a pioneer in interoperable grid systems development.

In 2002, the Open Grid Services Architecture (OGSA) [11] was introduced by the Global Grid Forum (GGF) to expand standardization. The OGSA provided a new architecture for grid applications based on Web Services in order to achieve interoperability using industry standards.

Many OGSA architecture implementations were developed, including one for Globus. Beyond the definition of a set of standardized interfaces, the OGSA architecture provides a framework for portable and interoperable service definition and thus provides a basis for grid development. The work carried out in this paper is deployed on a grid based on Globus (GT3).

To be an attractive choice, grid computing must provide adequate mechanisms for resource management to huge heterogeneous environments spread over multiple organizations that are geographically distributed [11]. One of the most important components of a resource management system for grids is resource discovery.

3. Resource Discovery in Grids

Resource Discovery can be defined as a directory service directed to the spontaneous network's environment [3]. In these networks, resources can enter or leave at any time and this mechanism's proposal is to provide information about the resources available in a specific moment.

In a grid environment, resource discovery is a very complex task basically for two reasons. First, the resources that are potentially interconnected to a grid are not only computers, but also softwares, instruments and data, among others. This adds a very high degree of heterogeneity that should be taken into account. Second, a grid can have a very huge number of resources, spread over multiple administrative domains that are geographically distributed. In this scenario, scalability issues must be taken into account.

Many resource discovery mechanisms have been proposed in the literature for grid environments. Some of them, such as MDS2 *(Metacomputing Directory Service*) [4] and Data Grid Resource Discovery [9], are specific resource discovery services. However, the great majority of them are contained in more general grid proposals such as MDS3 (*Monitoring and Discovery Service*) [12], NWS (*Network Weather Service*) [16], VIRD (*VEGA infrastructure for Resource Discovery*) [8], GLOPERF [10] and NimRod/G [1].

MDS2 [4] and MDS3 [12] were proposed in the context of the Globus Toolkit project [5]. Although they have the same acronym, there are great differences between them since MDS2 is a LDAP (*Lightweight Directory Access Protocol*)-based implementation whereas MDS3 is based on Web services. This led to an extensive change on the structure of MDS, that made MDS3 be part of the GT3 Information Services Component.

NWS [16] is a performance monitoring and prediction service that is based on the concept of performance sensors which gather information about the available resources and use them to predict performance. The Data Grid system [9] is a resource discovery mechanism specific to Oracle databases integrated to the Sun Grid Engine [14]. VIRD [8] is a three-level decentralized infrastructure for resource discovery on grids. GLOPERF [10] is in fact a monitoring tool for network performance in grids which is integrated to Globus Toolkit 2. Nimrod/G[1] is a resource management system for grid environments that uses the concept of grid economy to negotiate resource allocation and integrates resource discovery capabilities.

The main characteristics of these systems are summarized in Table 1.

Table 1. Resource Discovery Mechanism characteristics

System	Topology	Extensible	Data retrieved	Type of retrieval	Output format
MDS2[4]	Hierarchical	No	Operating system, CPU, memory, file system, others	information providers	LDIF
MDS3[12]	Decentralized	No	Operating System, CPU, memory, file system, others	service providers	XML
NWS[16]	Decentralized	No	Network throughput and latency, CPU, memory, others	passive and active sensors	LDIF
Data Grid[9]	Hierarchical	No	Tables, stored procedures, database load, others	daemon	specific metadata format
VIRD[8]	Hierarchical	No	Operating system, CPU, memory, others	name servers	BNF
Gloperf[10]	Decentralized	No	Network throughput and link activity	daemons acting as sensors	LDIF
Nimrod/G[1]	Hierarchical	No	Operating System, CPU, memory, file system, network, others	MDS2 [4]	LDIF

As can be seen in table 1, the resource discovery mechanisms proposed for grids are either hierarchical or decentralized. Most of these mechanisms retrieve data related to the machines that compose the grid (operating system used, CPU load, memory occupation, among others). Some of them [1][16][10] also retrieve network-related data. An example of a discovery mechanism for databases in grids [9] is also presented. The entity used to retrieve the information about the resources varies a lot (information and service providers, sensors and daemons) and also does the output format.

None of the resource discovery mechanisms analyzed were able to execute extensible searches. An extensible search allows the incorporation of new types of data to be retrieved to the resource discovery mechanism while it is already active. In other words, an extensible search allows the dynamic and transparent creation of a new resource search. This feature is particularly interesting in a grid environment where new types of resources can be added at any time and new needs of information arise constantly.

4. Design of the Extensible Resource Discovery Mechanism

In this section, we propose a resource discovery mechanism for grid environments that is able to incorporate new resource searches on-the-fly. As far as we know, this is the first time that an extensible resource discovery mechanism is proposed for grids.

As [1][4][8][9], we propose an hierarchical topology. The Grid is divided into different virtual organizations (VOs), as illustrated in Figure 1. Each virtual organization is structured following the Master x Slave paradigm to perform data retrieval of the available resources. Although in figure 1 we have represented only one master at each virtual organization, an hierarchical structure composed by many masters can be defined for a given VO.

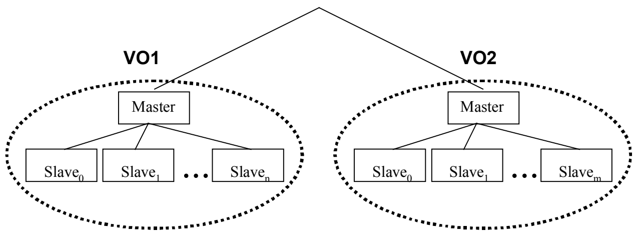

Figure 1. Hierarchical topology of the resource discovery mechanism

4.1 Basic Resource Discovery

Like the mechanisms mentioned in section 3, our mechanism is able to perform basic resource discovery. In our case, a master/slave architecture is used where the master is responsible to update the resource database and the slaves are responsible to actually retrieve information from each machine that composes the grid. This basic mechanism is shown in Figure 8.

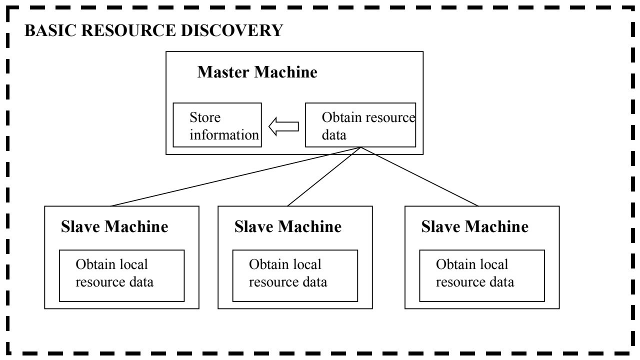

Figure 2. Basic resource discovery in our mechanism

Both master and slave activities are performed by two grid services defined in the context of Globus Toolkit 3[12].

The definition of both services is made in a WSDD (*Web Service Data Definition*) file that may be visualized in Figure 3.

```
<?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="br/unb/cic/infosys/searchslave/SearchResourceService"
      provider="java:RPC">
    <parameter name="allowedMethods" value="*"/>
    <parameter name="className"
     value="br.unb.cic.infosys.searchresource.impl.SearchResourceImpl"/>
 </service>
 <service name="br/unb/cic/searchmaster/SearchMasterService"
    provider="java:RPC">
   <parameter name="allowedMethods" value="*"/>
   <parameter name="className"
    value="br.unb.cic.infosys.searchmaster.impl.SearchMasterImpl"/>
  </service>
</deployment>
```

Figure 3 . Services definition in WSDD file

The master service (*SearchMasterService)* is responsible for looking up the slaves services (*SearchResourceService)* which are available in each of the associated slave machines and invoking their resource information retrieval.

The lookup for slave machines is started by reading a configuration file that contains the slave machines' description. It means that each master machine has its own file in XML format that contains the IPs and names of all the slave machines that are under its control and responsibility.

An example of a configuration file is illustrated at figure 4. In this case, there are three slave machines (pos-06, pos-09 and pos-10) that will have their information retrieved in the resource discovery mechanism.

```
<System xmlns xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\xsd\informationHost.xsd">
  <Host>
      <MachineName>pos-06</MachineName>
      <IP> 164.41.14.86 </IP>
  </Host>
  <Host>
      <MachineName>pos-09</MachineName>
      <IP> 164.41.14.89 </IP>
  </Host>
   <Host>
      <MachineName>pos-10</MachineName>
      <IP> 164.41.14.90 </IP>
  </Host>
</System>
```

Figure 4. XML configuration file with information about the slave machines

The slave service (*SearchResourceService)* is responsible for retrieving default local information about the slave machine. This information contains both static and dynamic data, as shown in figure 5.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<System>
   <Host>
     <Name> pos-03.cic.unb.br </Name>
     <IP> 164.41.14.83 </IP>
     <Memory>
         <Total> 109880.0 </Total>
         <Available> 1496.0 </Available>
     </Memory>
     <CPU>
          <Model> AMD Athlon(tm) Processor </Model>
          <Total> 896.0 </Total>
          <Available Percent> 1.36148 </AvailablePercent>
     </CPU>
     <OS>
        <Name> Linux </Name>
        <Version> 2.4.20-0 </Version>
     </OS>
     <Process>
          <Number> 78 </Number>
     </Process>
  </Host>
</System>
```

Figure 5. Information retrieved by default by our mechanism

4.2 Extensible Resource Discovery

The architecture of our resource discovery mechanism can be defined in a module-basis. The module described in section 4.1 is called basic module. Besides this one, diverse personalized modules can be coupled to it on-the-fly to perform specific searches that were not considered when the basic module was designed.

In order to define a personalized module, the user/administrator must perform only two tasks: deploy a method that retrieves personalized information and describe its characteristics in a configuration file as shown in figure 6.

```
CLASS_NAME=br.unb.cic.bioinfosys.BioFileSearch
NUM_METHODS=1
METHOD1=getFileLength
NUM_JARS=1
JAR1= BioInfosys.jar
```

Figure 6. Personalized module's configuration file

The file in figure 6 informs that a new personalized module has been deployed with the Java main class *BioFileSearch* defined in the package *br.unb.cic.bioinfosys.* In this case, there is a single method defined as *getFileLength* and one JAR (*Java Archive*) file called *BioInfosys.jar* file that will be necessary to complete the Java Virtual Machine *classpath*.

Upon detecting that a new configuration file has been generated, our extensible resource discovery mechanism incorporates automatically the new resource search to the basic resource discovery mechanism described in section 4.1. After doing that, both basic and personalized data are retrieved. Many personalized searches can be defined to extend the basic discovery mechanism.

Figure 7 illustrates the main modules in this process. Three main activities are executed concerning the new personalized module: detection, incorporation and container manipulation.

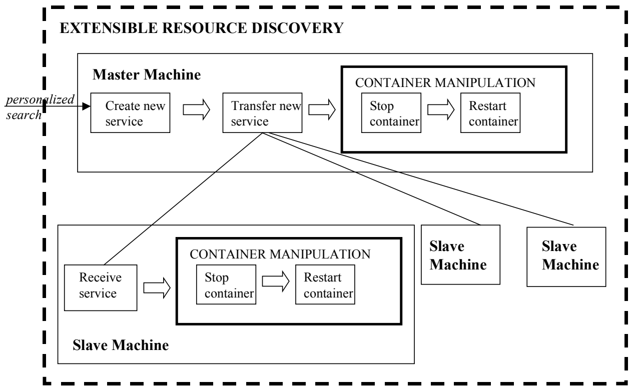

Figure 7. Extensible information search functionality

The personalized module's detection is made by the periodic verification of the existence of the configuration file shown in figure 6. When this configuration file is found, it is read by our extensible resource discovery mechanism and the file *PersonalizedServices.xml* (responsible for the personalized modules definition) is updated to indicate the name of the class and method to be invoked to perform the new search. This new module is automatically added to the personalized ones already defined in the file.

An example of the *PersonalizedServices.xml* file is shown in figure 8.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Services>
  <PersonalizedService>
     <ClassName> SoftwareSearch </ClassName>
     <Methods>
          <MethodName> isSoftwareInstalled </MethodName>
     </Methods>
  </PersonalizedService>
  <PersonalizedService>
     <ClassName> BioFileSearch </ClassName>
     <Methods>
          <MethodName> getFileLength </MethodName>
     </Methods>
  </PersonalizedService>
</Services>
```

Figure 8. Definition of personalized module in the PersonalizedServices.xml file

In this case, two personalized searches have been defined, since the tag <*PersonalizedService*> occurs twice.

The incorporation phase then begins. First, the JARs defined in the configuration file (figure 6) are added to the ones necessary for the new GAR (*Grid Archive*) file's creation. This creation represents the new resource discovery service that consists of the new personalized module added to the old defined service. Due to this operation, the system is capable of aggregating the new module to its resource discovery mechanism.

After creation, the GAR file and the *PersonalizedServices.xml* file are sent to all slave machines by the GRIDFTP mechanism [12] in a safe channel.

After being sure that the new grid service is already in the slave machines, the container manipulation phase begins. First of all, the old service is undeployed. Due to globus 3 restrictions [12], the container must be stopped. After that, the new service is deployed. When the deployment is finished, all the machines have their container automatically restarted. At this time, the new search is already available in all the machines' services.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<System>
   <Host>
      <Name> pos-03.cic.unb.br </Name>
      <IP> 164.41.14.83 </IP>
      <Memory>
         <Total> 109880.0 </Total>
         <Available> 1496.0 </Available>
      </Memory>
      <CPU>
          <Model> AMD Athlon(tm) Processor </Model>
          <Total> 896.0 </Total>
          <Available Percent> 1.36148 </AvailablePercent>
      </CPU>
      <OS>
         <Name> Linux </Name>
         <Version> 2.4.20-0 </Version>
      </OS>
      <Process>
          <Number> 78 </Number>
      </Process>
      <PersonalizedServiceInfo>
        <MethodsInfo>
           <Name> getFileLength </Name>
           <Result> 1242796843 </Result>
        </MethodsInfo>
      </PersonalizedServiceInfo>
      <PersonalizedServiceInfo>
        <MethodsInfo>
           <Name> isSoftwareInstalled </Name>
           <Result> true </Result>
        </MethodsInfo>
      </PersonalizedServiceInfo>
   </Host>
</System>
```

Figure 9. Resources information with a new service definition

On the slave machine, the retrieval of information when personalized searches have been defined occurs as follows. Every time a slave machine receives a resource's information request, it verifies if there is any personalized module defined to join its execution's results to the ones retrieved by default.

If there is any personalized method, the basic module's infrastructure is prepared to read the personalized module's configuration file to retrieve the class responsible for performing the personalized search. By using the Java's Reflection [15] mechanism, this class is instantiated and its methods are invoked. All retrieved information (basic and personalized ones) is stored in the output XML file, as shown in figure 9.

In this file, the information contained until the tag *PersonalizedServiceInfo* is retrieved by default by our resource discovery mechanism. The <PersonalizedServiceInfo> tag shows that there are two methods that were used to extend the basic service. In this case, the methods "getFileLenght" and "isSoftwareInstaleld" were the methods incorporated by the proposed extensible mechanism.

5. Experimental Results

A prototype of our extensible resource discovery mechanism was implemented using JAVA version 1.4.2_04, jakarta ant 1.6.2 and globus toolkit 3.2.1.

The test environment was composed by 8 machines spread over 2 laboratories (LAICO and LabPos), interconnected by a campus-area network at the University of Brasilia. All the machines had installed Linux kernel 2.4.20-8. Some of the hardware characteristics of the machines are shown in table 2. In our current prototype, only one virtual organization is used and only one master machine was defined.

Table 2. Machines used in our grid testbed

Machine	Lab	CPU	Memory	Disk
pos03	LabPos	AMD 900MHz	120MB	20GB
pos06, pos08 pos10, pos15	LabPos	AMD 1GHz	256MB	20GB
carbona, fau, magicien	LAICO	P 1.7GHz	256MB	20GB

In our tests, there was an interest to extend the resource discovery mechanism in order to incorporate new resource searches that would be useful for our other grid research projects. Therefore, we chose to focus on the needs of PackageBlast [13], a grid service to compare biological sequences against a genomic sequence database.

In this real problem, all the machines involved in the comparison must have the same genomic database, with the same version. Nowadays, this verification is manually done, where a person is responsible to verify if the file exists and its length. Therefore, it would be very interesting if the resource discovery mechanism could obtain this information automatically. So, we have defined a new module that is responsible to do that.

In this section, we present our experimental results considering this new module's incorporation.

In our mechanism, there is a thread that keeps periodically looking for the configuration file (section 4.2). In our tests, we considered this pause period as 30s and 5s and the tests were done varying the number of slave machines (figure 10).

The times measured were unavailable and incorporation time. The unavailable time is measured from the time the resource discovery service is stopped in the master to the time the extended resource discovery service is restarted. Since the master service is the last one to be stopped, this time does not depend on the number of slave machines, as can be seen in figure 10. On the other hand, the incorporation time does depend on the number of slave machines, since it comprises the time between the detection of the personalized configuration file and the termination of the old resource discovery grid service in the master machine.

By the collected results, we noticed that the best results for incorporation times were obtained through the data retrieval using the pause period configured to be 5s. With a pause period of 30s, the resource discovery mechanism takes longer to notice that a new personalized module has been defined and this has an impact on the overall execution time. Thus, we decided to use 5s as the pause period.

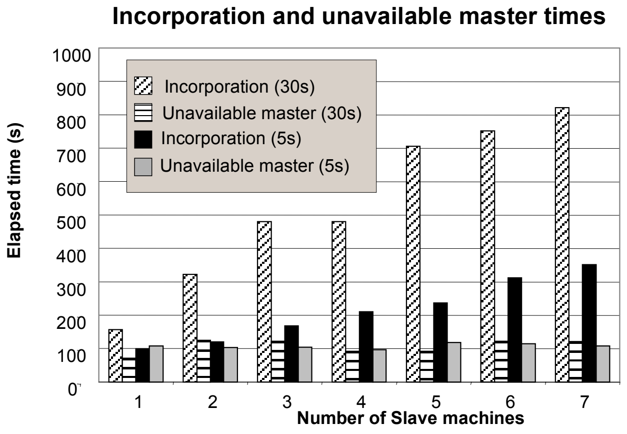

Figure 10. Service incorporation and unavailable times for the master machine for 1 to 7 slave machines, considering a 3.2MB GAR file, with pause periods of 5s and 30s

The size of the grid archive (GAR) file is a point that has also to be considered since the transfer to the slave machines could add a great overhead. Therefore, to verify the impact of the size of the new GAR file on the unavailable and incorporation time, we had our tests repeated using the pause time of 5s and the file's size doubled (6.4 MB).

This comparative analysis using these different file lengths can be seen in Figure 11.

As can be seen in figure 11, the new module's incorporation times increase when the GAR size increases. However, this increase is not so expressive.

The longest increased times were observed when 1 and 5 slave machines were being used. In this case, the incorporation's total time suffered an addition of 15.4% and 15.2% respectively when the file size was doubled. The incorporation's time using 4 and 6 machines suffered an addition of 4.7% and 4.6% respectively.

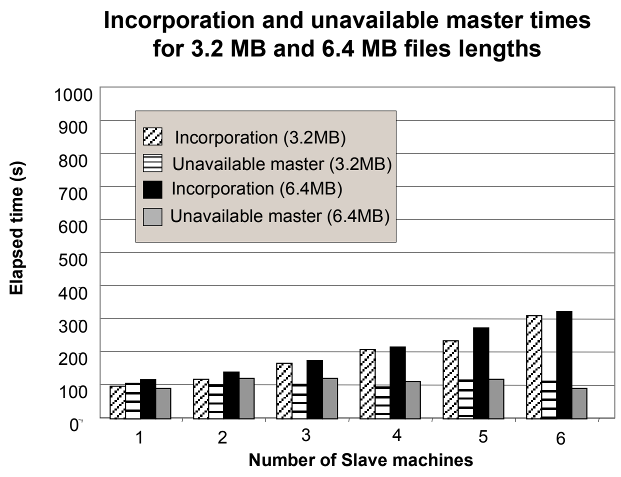

Figure 11. Service incorporation and unavailable times for the master machine for 1 to 7 slave machines, considering a 3.2MB and 6.4MB GAR file, with pause periods of 5s

That means that, although the increase on the file size causes an addition to the total time, this addition is not proportional to the file size, so it does not represent a significant impact in our extensible resource discovery mechanism's utilization, even considering the file sizes used in the tests, which were quite high.

6. Conclusion and Future Work

In this paper, we proposed and evaluated an extensible resource discovery mechanism for grid environments. The possibility of extending a resource discovery mechanism, by adding new resource searches automatically and "on-the-fly", is very useful for the user/administrator, who does not need to manually modify the resource discovery grid service to add his new search criteria.

The results obtained on a grid prototype composed by 8 machines spread over 2 laboratories presented very reasonable times for the incorporation of a new resource search, considering all the complexity involved while incorporating and distributing a new grid service to all the involved machines.

By our results, it was clear that it is not interesting for a master to manage a big number of slave machines and, if it is necessary to retrieve information from a high number of machines, the system's architecture permits an organization of these machines in different sets of master x slave machines.

For a future work, we intend to add a hot deploy mechanism, just like [7], to our mechanism. That would avoid the deploy and undeploy operations that were done in the container during the manipulation phase. Beyond this, it is interesting to evaluate tests in environments with different master machines in different virtual organizations. It would also be interesting to establish an hierarchy among the master machines to consolidate the slave machines' information retrieved, among some other possible improvements that could be done in our mechanism utilization. Also, we intend to design a mechanism to validate the personalized search before it is included in our resource discovery mechanism.

7. References

[1] R. Buyya, D. Abramson and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, Proceedings of the High Performance Computing in the Asia-Pacific Region, 2000, China, p.283-289.

[2] C. Catlett, L. Smarr, "Metacomputing". *Communications of the ACM*, 35 (6). 44-52. 1992.

[3] G. Coulouris, J. Dollimore, T. Kindberg, "Distributed Systems: Concepts and Design", Addison-Wesley, 2001, 772p.

[4] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. *A Directory Service for Configuring High-Performance Distributed Computations*. Proc. 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375,1997.

[5] I. Foster, C. Kesselman, *Globus: A metacomputing infrastructure toolkit*. The International Journal of Supercomputer Applications and High Performance Computing 11, 2 (1997),115--128.

[6] I. Foster, C. Kesselman, editors. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[7] T.Friese, M. Smith, B. Freisleben, "Hot Service Deployment in an Ad Hoc Grid Environment", Proceeding of the 2nd Int. Conference on Service Oriented Computing (ICSOC'04), November, New York, USA,2004, p.75-83.

[8] Y. Gong, F. Dong, W. Li and Z. Xu, *VEGA Infrastructure for Resource Discovery in Grids*. Journal of Computer Science Technology, 18(4): 413-422, July, 2003.

[9] S. Jayasena, C. Yee, J. Song, A. Stoelwinder, C. W. See and W. Wong, *Data Resource Discovery in a Computational Grid*, Proceedings of the Grid and Cooperative Computing International Workshops, 2004, p.303-310.

[10] C. A. Lee, J. Stepanek, R. Wolski, C. Kesselman and I. Foster. *A Network Performance Tool for Grid Environments*. In Proceedings of 7th IEEE International Symposium on High Performance Distributed Computing, pages 260--267, 1998.

[11] J. Nabrzyski, J. M. Schopf, J. Werglarz, “Grid Resource Management: State of the Art and Future Trends”, Springer, September, 2003, 598p.

[12] T. Sandholm, J. Gawor, “Globus Toolkit3 Core – A Grid Service Container Framework, Technical Report, available at www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf, 22p.

[13] M. S. Sousa, A. C. M. A. Melo, “PackageBlast: An Adaptive Multi-Policy Grid Service for Biological Sequence Comparison”, Proc. of the ACM Symposium on Applied Computing (SAC), 2006, to appear.

[14] Sun Microsystems , “Sun Cluster Grid Architecture - A technical white paper describing the foundation of Sun Grid Computing”, White Paper, 2002.

[15] Sun Microsystems. *Java Core Reflection: API and Specification*, February, 1997, available at http://java.sun.com/products/jdk/1.1/- docs/guide/reflection/.

[16] R. Wolski, N. Spring and J. Hayes, "*The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*,", Future Generation Computer Systems, vol. 15, no. 5-6, pp. 757-768, October 1999.