A Peer-to-Peer Information Service for the Grid

Diego Puppin, Stefano Moncelli, Ranieri Baraglia, Nicola Tonellotto, Fabrizio Silvestri
Institute for Information Science and Technologies
ISTI - CNR, Pisa, Italy
via Moruzzi, 56100 Pisa, Italy
{*Diego.Puppin, Ranieri.Baraglia, Nicola.Tonellotto, Fabrizio.Silvestri*} *@isti.cnr.it*
stefano7625@libero.it

Abstract

Information Services are fundamental blocks of the Grid infrastructure. They are responsible for collecting and distributing information about resource availability and status to users: the quality of these data can have a strong impact on scheduling algorithms and application performance.

Many popular information services have a centralized structure. This clearly introduces problems related to information updating and tolerance to faults. Also, in very large configurations, scalability can be an issue.

In this work, we present a Grid Information Service (GIS) based on the peer-to-peer (P2P) technology. Our system offers a fast propagation of information and has high scalability and reliability. We implemented our system complying to the OGSA standard using the Globus Toolkit 3. Our system can run on Linux and Windows systems, with different configurations, so to trade off between redundancy (reliability) and cost.

Key Words*: Grid information service, Grid middleware, Peer-to-peer.*

1. Introduction

The Grid is an emerging computing framework where resources are shared and inter-operate across independent organizations. In such an environment, it is very important to be able to discover efficiently which resources are available, what their status and cost are. A system where this information is outdated, approximate or difficult to access and browse can negatively affect the performance of scheduling algorithms and final-user code.

The Grid Information Service (GIS) is the infrastructure component responsible for collecting and distributing information about the Grid. It offers some tools to register resources, to query the data base, to remove lost nodes.

The first implementations of a GIS, used techniques based on *directories*, which are still used by Globus MDS-GT2 (LDAP). Directory-based systems suffer from a series of problems [7], including the fact that updated information does not propagate very quickly and that centralized servers can become bottlenecks or points of failure. Also, the underlying formalism limits the type of queries that can be submitted to the system. More modern approaches are based, for instance, on Relational Databases, on techniques for Internet Knowledge Discovery and on Agents.

In this work, we introduce a Grid Information Service (GIS) based on peer-to-peer (P2P) technologies and Routing Indices (RI) [3]. There is a growing interest to the interaction of the Grid computing paradigm and the peer-to-peer technology: both work within a very dynamic and heterogeneous environment, where the role and availability of resources can quickly change; both create a virtual working environment by collecting the resources available from a series of distributed, individual entities.

Even if nowadays some Grid-related tasks are performed by central servers, we believe that in the future all of them could be implemented as P2P services, to improve scalability, performance and fault-tolerance. The Peer-to-Peer Community Grids project, among others, is working in this direction.

Talia and Trunfio [10] suggested that a new version of OGSA could integrate the concept of Grid and P2P, as did for Grid and Web Services. A P2P Grid middleware could be used to develop Grid applications based on this technology. In particular, they believe that the Globus Toolkit MDS and the Replica Management Service could benefit from being redesigned as P2P applications.

This paper is structured as follows. In Section 2, we give an overview of some existing information services, which represent the background of our work. Our infrastructure is presented in Section 3. In Section 4, we show the results of our preliminary tests. Finally, we conclude and we give an overview of future work.

2. Related Work

Due to the importance of Information Service within the Grid infrastructure, a variety of approaches has been presented in literature.

The *Globus Monitoring and Discovery Service* (MDS) [4] in GT2 uses LDAP as a back-end to store and manage data. It has a hierarchical structure, based on three main components.

- The Grid Index Information Service (GIIS) stores the information about a set of GRIS (see below), so to offer a complete picture of the status of an entire virtual organization. GIIS can be used to build up a hierarchical structure. GIIS can work as a cache for the monitored data.
- The Grid Resource Information Service (GRIS) is a distributed information service able to answer users' query about the status of a resource. Each machine can host a local GRIS, connected to one or more GIIS. It forwards each query to some local Information Providers, and then assemble the results. It can have a caching system to speed the query up.
- The Information Providers (IP) are local services, responsible for collecting information about the status of a given resource. Users can easily create new IPs as needed.

MDS-GT2 is usually described as a hierarchical service. Nonetheless, the nodes on the top levels of the structure act as central servers for nodes in the lower levels, as they hold a cached copy of a part of the data published (this can be configured from 100% to no cache). They can become bottle-necks in the structure, if the number of nodes they serve and the caching level are high.

Another problem is given by the fact that they use a procedural language for queries, and users need to know the directory structure in advance. LDAP is fast when it knows the query structure, so it can build its own internal database accordingly, but performance is lower for general queries.

Some of these limitations are overcome by more modern distributed directory services, which are also gaining attention in the commercial world.

A system based on relational databases is *Relational GIS* (RGIS) [5], developed as a part of the Unified Relational Grid Information Services (URGIS) project. This system uses a SQL-like declarative language for queries. Thus, users can ask, for instance, for a set of hosts totaling a given memory size. Like any relational database, RGIS creates independent indices for each attribute, in order to speed up query response time. Furthermore, very complex queries are elaborated non-deterministically, or approximately. This reduces the quality of searches, but can result in a much shorter response time. Clearly, this introduces an important trade-off problem in the design of this system.

Grid Monitor [1] is a tool developed as a part of the UK e-science project and it is used to monitor information servers within the UK Grid. It is an extension of Globus MDS aimed to a more natural user interface. The software is a three-tiered Web system:

- Web Clients allow user to inspect the status of each resource;
- Web Servers and Servlets are used to connect to the data-base;
- an underlying relational data-base stores the historical data and keeps a list of each organization registered.

Each new resource is required to register to the UK Grid Support Centre, submitting all the meta-data needed to complete its description. A central server stores this information. After registering, the MDS server, i.e. the highest-level GIIS, is connected to the Web server. Servlets connect to this server and read the data into a local cache. Local Java applets are used to interface with it: they can show a map which highlights the available MDS servers. They can be browsed in order to get more information about some resources.

The *A4 Agent System*, presented in [6], integrates MDS-GT2 with a system of agents. As in MDS-GT2, each resource is associated with a GRIS, which communicates to GIIS via a Local Resource Manager (LRM). When a user wants to run a given application on the Grid, they communicate with the agent. The agent communicates then with its GIIS or with other agents in the hierarchy. If the GIIS cannot answer, it communicates with the LRM, which holds a cached copy of the information available from the GRIS. This system is able to speed up the response time of a MDS query. Furthermore, the agents are able to predict the performance of a given application on the chosen resources.

With the introduction of the *Globus Toolkit 3*, the Globus architecture of the information service has dramatically changed [11]. Now, each entity is represented

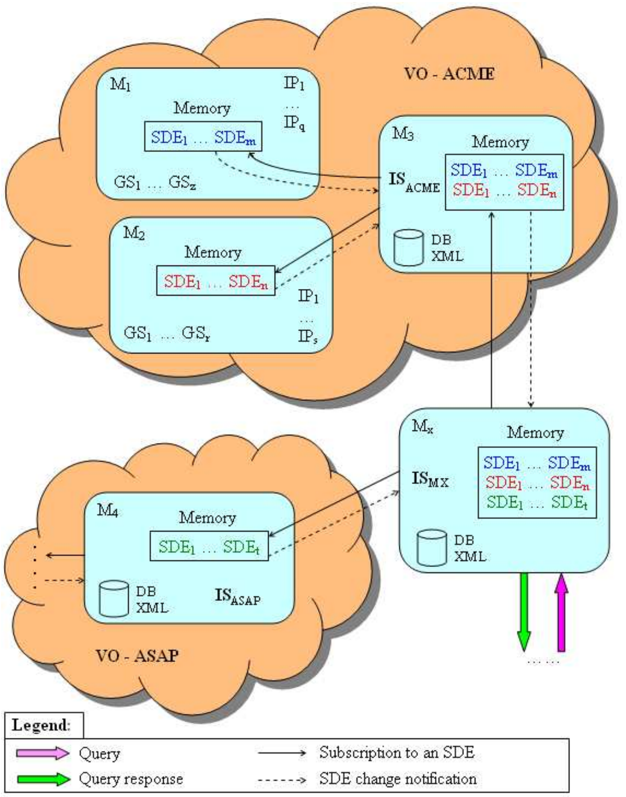

Figure 1. Index Service Hierarchy for MDS-GT3.

by a Grid Service, which is an extended Web Service following the new conventions introduced with GT3 and OGSI. These Grid Services expose their status as a collection of Service Data (SD), composed of Service Data Elements (SDE). Service Data replace the mechanisms offered by GRIS in MDS-GT2: they replace the GSI-enabled mechanisms present in LDAP with the OGSA mechanisms for binding.

Also, the Service Data sources are tailored to comply with the WSDL standard: each information provider publishes its data as an XML file, following a precise Service Type WSDL. This replaces the MDS schema written with the LDAP schema format.

The Index Service, within MDS-GT3, offers functions to index, query and browse the gathered SDEs. It replaces GT2 GIIS. A simple example (Figure 1) will illustrate the new architecture of this system. Machines M1 and M2 host a set of Grid Services (GS) exposing some Service Data. The Index Service for the Virtual Organization ACME (VO-ACME), IS-ACME, is sitting on M3 and subscribes to the SDEs in order to be notified of changes, using the OGSI mechanisms. M3 will also keep a cached copy of the data. Similarly, in the Virtual Organization ASAP (VO-ASAP), M4 will work as an Index Service for the machines in the group (not shown). A machine Mx, higher in the hierarchy, will subscribe to each SDE present on IS-ACME and IS-ASAP. Each change on M1 and M2, for instance, will be propagated to IS-ACME and IS-MX. The list of SDEs has to be known to the Information Service, as a configuration file or through user input.

The Index Service is composed of two main parts:

- the *Provider*s are responsible for generating SDEs;
- the *Aggregator* is responsible for aggregating and indexing the SDEs coming from the hosts in the VO.

In our opinion, the main limitations come from this hierarchical structure:

- when a new SDE becomes available, the new information does not propagate automatically up the hierarchy;
- at the top levels, each IS is required to store a very large number of SDE.

We believe that centralized, hierarchical systems are not suitable to Grids and highly distributed systems. Due to their being highly heterogeneous and dynamic, more flexible and self-adapting solutions are needed.

3. P2P GIS: Description of the Architecture

In this section, we present our implementation of a Grid Information Service (GIS) based on the peer-to-peer technology. Its main features are:

- peer-to-peer technologies for propagating data and elaborating queries;
- routing indices to reduce network flooding and to optimize message forwarding;
- node clustering and use of super-peers;
- redundant configurations, when high reliability is needed.

The system is made up of two main entities (see Figure 2):

- the *Agent* is responsible for publishing information about a node to the super-peer;
- the *Aggregator* runs on the super-peer; it collects data, replies to queries and forwards them to the other super-peers; it also keeps an index about the information stored in each neighbor super-peer.

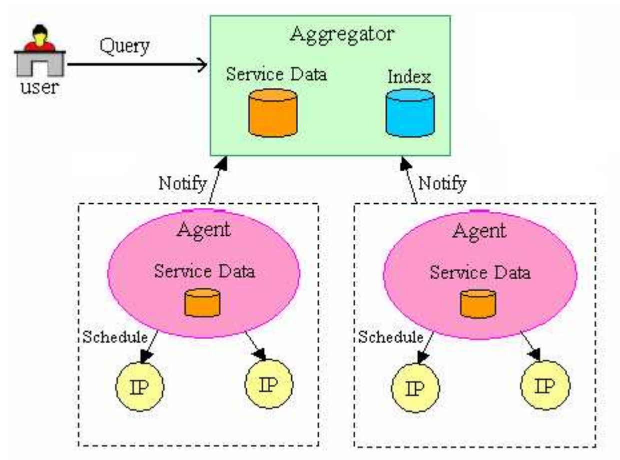

Figure 2. Overview of our system.

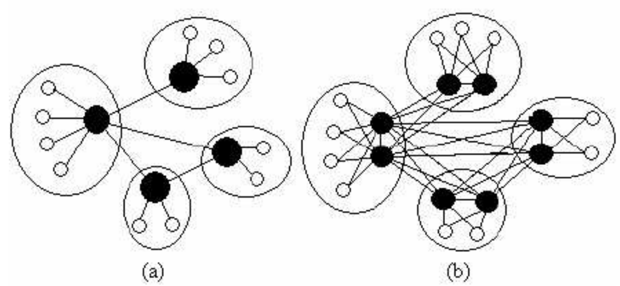

Figure 3. Example of super-peer networks: (a) with no redundancy, (b) with 2-redundancy. Black nodes represent super-peers. White nodes are clients. Clusters are limited by circular lines (from [12]).

Super-peer and redundant networks are described in the next section. Then, we outline the structure of Agents and Aggregators. Routing indices and our search technique are discussed in Sections 3.3 and 3.4. Finally, we explain how the system is bootstrapped.

3.1. Super-peer Redundant Networks

It is well known that pure P2P networks spend useful bandwidth in functions that can be performed by local caches [8, 9]. This is why super-peer networks emerged as a trade-off between totally distributed systems and cache-based services [12].

Our system is set up as a super-peer network: some nodes, called *super-peers*, work as servers for a *cluster* of nodes — which usually corresponds to a virtual organization or a subset thereof — but they work as peers in a network of super-peers. Moreover, this network can be built as a redundant network, where super-peers are replicated within each cluster (see Figure 3). This solution introduces two main benefits.

- Replicas hold a copy of the same data. In case of failure of one replica, the system will not stop working.
- The work-load can be shared among replicas. Queries can be alternately sent (or forwarded) to each of them in turn. Also, the aggregate bandwidth for forwarded queries is much higher.

On the other side, communication costs can increase, for two reasons. First, when a new node joins a cluster or its data are updated, it has to send a message to K super-peers in a K-redundant network. Second, there are $O(K^2)$ connections between two K-replicated super-peers. The choice of K is a trade-off between reliability and cost.

3.2. Agents, Aggregators and Information Providers

The *Agent* works as a Grid Service available on each machine in the network. It publishes all relevant information, as is made available by *Information Provider* tools (IP).

The Information Providers, scheduled by the Agent, periodically query the resources and store the information gathered as Service Data Elements (SDE), according to the OGSA standard. Each SDE is tagged with a list of keywords, used for subsequent queries. In our system, there is an Information Provider for each resource. When a user chooses to publish information about some resources, they will describe the type of information using our taxonomy [2], in particular they will specify a *Refresh Rate*, which describes how often the information is to be refreshed. Static data have a Refresh Rate equal to 0.

When a resource is published, the name of its Service Data is broadcast to all the Aggregators in the cluster,[1] so that they can subscribe to it. Aggregators work as servers within their cluster, and as peers in the network created by all the Aggregators. In particular, they are responsible for forwarding queries coming from other Aggregators to the most likely destination.

1 There can be more than one Aggregator in a redundant network.

memory	available main memory
processor	processor type and model
processorLoad	average load in the last 1, 5 and 15 min.
operatingSystem	operating system name and version
diskSpace	disk space on each available volume

(a)

```
<?xml version="1.0" encoding="UTF-8"?>
<Resource name="Processor">
   <Property name="Vendor" value="GenuineIntel"/>
   <Property name="Model" value="Pentium"/>
   <Property name="Speed" value="751" units="MHz"/>
   <Property name="Cache" value="256" units="KB"/>
   <Property name="key" value="processor"/>
   <Property name="key" value="CPU"/>
</Resource>
```

(b)

Table 1. Available information providers (a). An example of resource description (b).

To prevent Aggregators from polling Agents at the end of each refresh interval, our implementation uses a *push* approach: the Agents periodically send the updated information to the subscribed Aggregators. A list of currently available Information Providers is shown in Table 1, along with an example of resource description. A configuration file will list a set of SDEs to be published by the Agent at launch time, but resources can be published or removed at any time by users.

A client can explicitly choose to remove its data from the super-peer data-base. Also, the Aggregators will scan the stored information and remove all the resources that failed to send updated information before the expiration of its validity.[2] This way, the super-peer will always have timely information about the clients connected to it.

3.3. Routing Index

The *Routing Index* (RI) is used to improve the performance of our peer-to-peer routing, and to prevent the network from being flooded. The Routing Index is a technique to choose the destination where a query should be forwarded: the RI represents the availability of data of a specific type at the neighbors, which is related to the probability each neighbor has to satisfy a given query. We implemented a version of RI called *Hop Count Routing Index* (HRI), which considers the number of *hops* needed to reach a datum. The HRI counts the number of data within a given number of hops. Data are then divided in classes by their keyword.

2 The super-peer actually waits three times longer than the refresh time, in order to tolerate possible unexpected delays in the network.

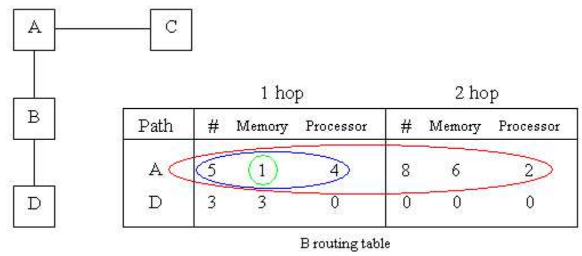

Figure 4. HRI table for node B.

We used HRI as described in [3]: in each super-peer, the HRI is represented as a $M \times N$ table, where M is the number of neighbors and N is the horizon (maximum number of hops) of our Index: the n-th position in m-th row is the number of data that can be reached passing from neighbor m within n hops.

Suppose that, from node B, we are looking for data about memory (see Figure 4). Our *goodness function*(see [3]), will give a higher value to A, because within short distance (2 hops) we can reach 6 resources. On the contrary, D could give us back information about only 3 of them.

When a new super-peer joins the network, it sends information about the data it controls to all its neighbors. They will update their table, adding the new data to those available within distance 1. Then, they will send the aggregate counts (excluding the new node) back to the new node itself. We use the techniques shown in [3] to deal with cycles in the network.

3.4. Search Techniques

In literature, three techniques are commonly used for searches in P2P networks.

1. Search is performed by flooding in systems such as Gnutella. Each search is forwarded to all the neighbors, until a matching datum is found. This is a very simple solution, but clearly introduces bandwidth problems.
2. Other systems use centralized servers to answer the query. These servers build an index of available data by crawling the network, or by asking each node to send a list of its data. Problems of scalability and fault-tolerance are typical of this

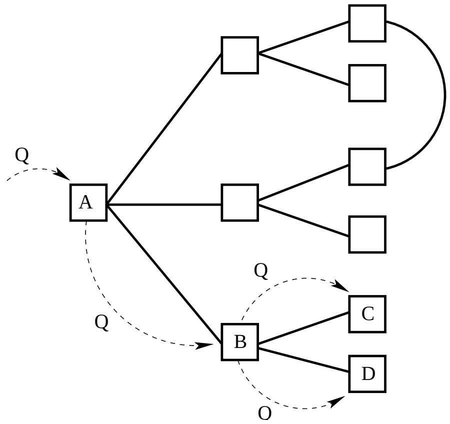

Figure 5. A query (Q) is forwarded from A to the best neighbors (B, C, and D).

approach. On the other side, the response time is generally very low.

3. A third way is followed by systems based on distributed indices. In these configurations, each node holds a part of the index. A query is forwarded to a neighbor chosen using a Routing Index. The index optimizes the probability of finding quickly the requested information, by keeping track of the availability of data to each neighbor.

The third approach is followed in our system. Each query is submitted, by each node, only to its cluster's super-peer, which will pass it to other super-peers if needed. To this purpose, the super-peer keeps information about all the nodes in its cluster, and creates an index for it.

Each query is tagged with an expiration time. At each step, the expiration is checked. If the query is still valid, it is forwarded to the best neighbor (using our Routing Index). Then, the local SDEs are matched against the query. This way, communication and computation are partially overlapped. The matching SDEs are sent back directly to the original requester as XML data.

If there are no available neighbors, as for C in Figure 5, the query is returned to the sender (B), which will choose the second best neighbor (D). An alternative approach could be to send the query to two or more neighbors, in parallel. This has better response time, but can flood the network. We are planning to experiment with this trade-off on a large network in the next future.

3.5. Bootstrapping the System

To start up the system, each Aggregator has to know the name of another one. Communicating with each other, the Aggregators will explore the topology of the system. Each Agent broadcast information about its presence at its launch time: all the Aggregators in the cluster will list it among their clients, and will update their Routing Index counting the SDEs published by the Agent.

In other words, each new Agent will connect to its super-peer just by broadcasting a message across its organization. From that moment on, its information will be available to any user through its super-peer. New Aggregators will connect to a running Aggregator, and through it they will learn about the network configuration: all data will be available to them too.

There is no need, for Aggregators and users, to know position and type of resources available, or to know the network topology. As seen, this can be not true for hierarchical services.

4. Experimental Results

Our system was developed using Globus Toolkit 3.0.2 and Java 1.4.1. The system runs under Linux Red Hat 8 and 9, Linux Debian, and Microsoft Windows 2000. It is compliant to the OGSA standard, and uses libraries and tools from the Globus Toolkit 3.

Our tests were performed on a Grid involving five organizations: ISTI-CNR, located in Pisa; University of Pisa; IIT-CNR, in Pisa; IMATI-CNR, located in Genoa; and the University of California at San Diego. The test configuration is shown in Figure 6. We artificially split ISTI-CNR into two virtual organizations by using different broadcast masks for the two subsets. This way, the Agents will connect to exactly one Aggregator.

In our first tests, we verified the performance when working within the organization's borders. Queries were sent from Rubentino about the status of resources monitored by Novello. On Novello, matching SDEs are sent back to Rubentino very fast: the first result is generated within 10 ms. The results arrive regularly, within few hundred milliseconds (see Table 2(a)).

When we cross the institutions' borders, delays related to the network are more evident. We launched several queries from Orione about the status of resources within the ISTI-CNR and the IIT-CNR organizations. Queries were elaborated by Rubentino, Novello and Morip. Again, we measured that less than 10 ms are needed to generate the first matching SDE, but results can take much longer to cross institutions and

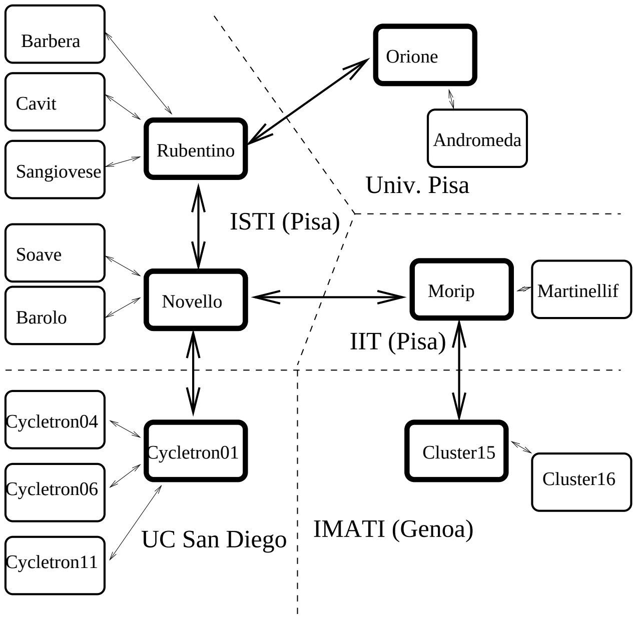

Figure 6. Our test configuration. An Agent is running on each machine (boxes). An Aggregator is running on thicker boxes. Arrows represent connections. The dashed line are the borders among participating institutions.

return to Orione. We believe that the firewall configuration, and other network effects can contribute to this large delay (see Table 2(b-c)).

For queries from farther institutions (IMATI in Genoa and UCSD), response time grows slowly with distance, and can be greater than 1 second (see Table 2(d-e)). This is a result to be expected, if we consider that the *ping* time can be 1000 times greater than among institutions in Pisa.

4.1. Redundant Configuration

Our system can be used with a redundant configuration for improved reliability, as in Figure 7. We run some initial tests, which showed the effectiveness of this solution: when one of the replica failed, the system continued running seamlessly. Response time did not change significantly. We expect that, in a very large configuration, redundant peers can offer a lower response time, when they are queried alternately. We are testing this hypothesis, and results will be available in the next future.

(a) Queries from Rubentino about Novello

Server side	9.1	31.9	40.0	48.3
Client side	212.2	229.2	345.4	436.4

(b) Queries from Orione about Novello

Server side	7	34.6	49.8	65.8
Client side	767.4	826.4	935.3	981.3

(c) Queries from Orione about Morip

Server side	9.6	57.8	72.6	87.4
Client side	788.0	850.9	946	999

(d) Queries from Cluster15 about Orione

Server side	10.1	40.3	52.3	64.5
Client side	890.1	905.3	958.1	1001

(e) Queries from Cycletron01 about Orione

Server side	34.2	260.8	300.2	310.1
Client side	950.4	1104.8	1187.7	1211.7

Table 2. Average time (in milliseconds) to generate (server-side) and receive (client-side) subsequent results of a given query.

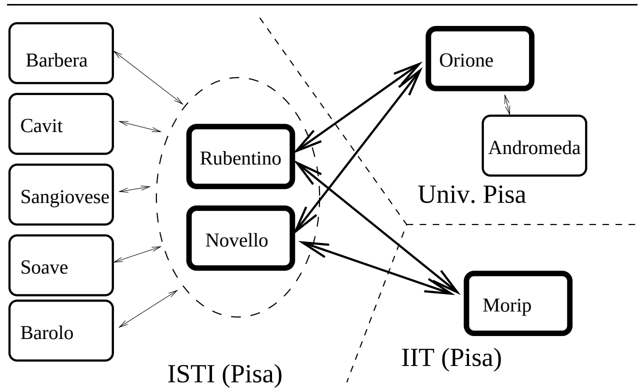

Figure 7. Our redundant configuration. The nodes within the ellipse behave as a replicated redundant super-peer.

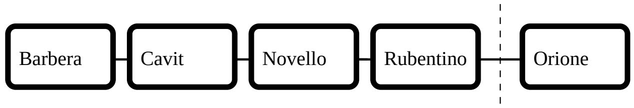

Figure 8. Configuration for our comparison with Globus MDS. Clients are not shown.

	P2P GIS		Globus
Hop #	1st	2nd	
1	743.5	801.4	3612
2	737.4	820.2	3588
3	775.5	831	3601
4	806.1	861.6	3640

Table 3. Comparison between our system (P2P) and Globus-MDS. Average response time (client-side) for subsequent results, about resources located at increasing distances (in milliseconds).

4.2. Comparison with Globus MDS-GT3

We compared our system with Globus MDS-GT3. The results shown in Table 3 come from our preliminary tests. All the data are taken at the client side, by measuring the time passed from the beginning of the query, to the arrival of results. Time was measured within the code, using the Java time API, for both Globus MDS and our system.

Due to problems with firewalls and Globus connection ports, we could not involve all the institutions in this test. Our system configuration was changed as follows: we created a linear chain of five Aggregators (see Figure 8), and, starting from Orione, we launched queries about data down the chain. Clients connected to each Aggregator are not shown. This is the worst case for our system, because clients connected to Barbera are separated by many hops from Orione.

We configured Globus Index Service (IS) with the same linear hierarchy: Cavit is subscribed to Barbera's SDEs, Novello to Cavit's and so on. In any case, all SDEs are cached by the Index Service, so the topology of ISs should not affect the results.

We can see some interesting results. As said, our system forwards incoming queries to the best neighbors before elaborating them. This way, a query can reach the Aggregator holding the desired data very fast. Then, results are sent back directly to the requester. This is the reason of the slow growth of response time with distance in our system.

For Globus, the response time is irrespective of the distance of the resources relevant to the query, as expected (all data are cached in our experiments).

Our system, under these experimental conditions, outperforms Globus. We have to consider that, at the moment, our system is extremely light-weight, while the Globus infrastructure can support a variety of tasks. Nonetheless, we can say that our system seems to scale effectively and respond very quickly, even if data are not cached: our queries read the datum — freshly updated — available to the Aggregator closest to the resources, not a potentially stale copy.

For the queries in this test, Globus returned only one result. We should emphasize that our system sends partial results as they are available, differently from Globus that waits for the complete answer. This could be exploited when a very quick result is needed.

Another importation consideration is that for this test we used a geographically limited configuration. For very large, world-wide configurations, the caching approach of Globus could hide certain network delays that could slow our system down. We are working to solve our firewall problems, and we will have extended results very shortly. Nonetheless, as showed before, our system responds very quickly on geographically wide networks too.

5. Conclusion

The Grid is a vast, dynamic, heterogeneous environments, where information about the status, configuration and cost of resources is extremely valuable: if users are able to find the best match to their needs, their applications will reach the best performance within the desired cost and time.

To monitor a Grid, a versatile system is needed, able to update very quickly, to satisfy a potentially very large number of users and queries, to tolerate delays and faults. Peer-to-peer systems, born out of the first file-sharing applications, evolved into very flexible frameworks, which are now gaining interest within the scientific community. The interaction between Grids and peer-to-peer systems is growing stronger, because P2P seems to be a very promising approach to some problems related to the Grid.

In this work, we presented a P2P Information System for the Grid. It is built as a network of super-peers, which aggregate the data about resources within a virtual organization. Queries performed by any client are passed among the super-peers, using optimization algorithms such as the Hop Counting Routing Index. Our system is based on Globus Toolkit 3 and complies to the OGSA standard: it can be easily integrated with any Globus-based Grid. In this first round of experiments, we used it for resource monitoring and discovery, but the same infrastructure could be used for file-sharing or other distributed applications, this way offering a P2P layer to Grid applications.

So far, the system was tested using a small network, split across five different institutions. In these preliminary tests, the system scales effectively. We could not measure big delays in queries for remote resources, which are constantly monitored by their Aggregators. This way, we always have updated information available to queries. Our system outperformed Globus MDS in our simple configuration.

We are currently testing a larger configuration, including more machines in the United States and Germany. Results should be available shortly.

6. Acknowledgements

We want to thank Antonio Manglaviti, who contributed to develop the first experimental prototype. Also, we thank IIT-CNR, IMATI-CNR, University of Pisa, and University of California at San Diego, which let us use their resources for our experiments.

This work has been partially supported by the Italian National Research Council (CNR) FIRB project GRID.it "Enabling platforms for high-performance computational grids oriented to scalable virtual organizations."

References

[1] M. A. Baker and G. Smith. A prototype grid-site monitoring system. Technical Report 2002.01, DSG, 2002.

[2] R. Baraglia, S. Moncelli, A. Manglaviti, F. Silvestri, and N. Tonellotto. Una tassonomia per la descrizione delle risorse in ambiente grid. Technical Report ISTI TR, ISTI-CNR, Appearing in 2004. (In Italian).

[3] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS-02)*, July 2002.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of HPDC-10*, 2001.

[5] P. Dinda and B. Plale. A unified relational approach to grid information services, 2001. Available at http://www.gridforum.org/1_GIS/RDIS.htm.

[6] H. N. L. C. Keung, J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd. Grid information services using software agents. In *Proceedings of the 18th Annual UK Performance Engineering Workshop (UKPEW 2002)*, pages 187–188, University of Glasgow, UK, July 2002.

[7] B. Plale, C. Jacobs, S. Jensen, Y. Liu, C. Moad, R. Parab, and P. Vaidya. Understanding grid resource information management through a synthetic database benchmark/workload. In *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, Chicago, IL, USA, April 2004.

[8] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, Department of Computer Science, 2001.

[9] J. Ritter. Why gnutella can't scale. no, really, 2001. Available at http://www.eecs.harvard.edu/~jonathan/papers/2001/ritter01gnutella-cant-scale.pdf.

[10] D. Talia and P. Trunfio. Toward a synergy between p2p and grids. *IEEE Internet Computing*, July/August 2002.

[11] The Globus Alliance. Globus toolkit 3, globus information services documentation, 2004. Available at http://www.globus.org/mds/.

[12] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the IEEE International Conference on Data Engineering*, March 2003.