A Grid Information Service Based on Peer-to-Peer[⋆]

Diego Puppin, Stefano Moncelli, Ranieri Baraglia, Nicola Tonellotto, and Fabrizio Silvestri

Institute for Information Science and Technologies
ISTI – CNR, Pisa, Italy
via Moruzzi, 56100 Pisa, Italy
Diego.Puppin@isti.cnr.it, stefano7625@libero.it,
{Ranieri.Baraglia,Nicola.Tonellotto,Fabrizio.Silvestri}@isti.cnr.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 may have a strong impact on scheduling algorithms and overall performance.
Many popular information services have a centralized structure. This clearly introduces problems related to information updating and fault tolerance. Also, in very large configurations, scalability may be an issue. In this work, we present a Grid Information Service based on the peer-to-peer 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 network configurations, so to trade off between redundancy (reliability) and cost.

Keywords: 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 the boundaries of 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 A system where this information is outdated, approximate or difficult to access and browse may negatively affect the performance of scheduling algorithms and of final-user code.

The Grid Information Service (GIS) is the infrastructure component responsible for collecting and distributing information about the Grid. It offers some

[⋆] We 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 MIUR GRID.it project (RBNE01KNFP) and the MIUR CNR Strategic Project L 499/97-2000.

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 [1], including the fact that updated information does not propagate very quickly and that centralized servers may become bottle-necks or points of failure.

In this work, we introduce a Grid Information Service (GIS) based on peer-to-peer (P2P) technologies and Routing Indices (RI) [2]. 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 may 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, many authors believe that in the future many of them could be implemented as P2P services, to improve scalability, performance and fault-tolerance.

This paper, which updates the results presented in [3], 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

The importance of Information Services within the Grid infrastructure has stimulated a rich research. Due to limited space, we can cite only the works that are closer to ours. Our starting point is clearly the Information Service model of the *Globus Toolkit 3* [4]. In Globus, each entity is represented by a Grid Service, which is an extended Web Service following the new conventions introduced with OGSA. These Grid Services expose their status as a collection of Service Data (SD), composed of Service Data Elements (SDEs). Service Data replace the mechanisms offered by GRIS in MDS-GT2: they replace the GIS-enabled mechanisms present in LDAP with the OGSA mechanisms for binding. The Index Service (IS) 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. Typically, there is one Index Service per Virtual Organization, which is used to build a hierarchy when several institutions are connected. Every Index Service works as a cache for all the ISs below it.

In our opinion, this hierarchical structure is cause of two main limitations: (1) when a new SDE becomes available, the new information does not propagate automatically up in the hierarchy; (2) at the top levels, each IS is required to store a very large number of SDEs.

Talia et al. [5] propose a P2P-based architecture for resource discovery that extends the GT3 information service model. It is broken into two layers: the lower one is a hierarchy of information services, which publish information owned by each virtual organization; the upper one is a P2P layer, which collects and distributes this information. Queries about non-local resource are managed by the P2P nodes. The protocol used to exchange messages extends the Gnutella protocol. It uses extensive caching and merging of queries and Grid Service invocations instead of raw TCP messages. Our work differs in that we use a more advanced query forwarding strategy based on Routing Indices, and in the fact that our system never returns cached, potentially out-dated, information.

Carmen [6], developed at the HP Labs, has a structure similar to our proposal. It offers a discovery service based on P2P networks, structured in peers and super-peers. Unfortunately, we could not find in literature any results about its effectiveness. The system is apparently very complex, and no comparisons are given with other system. At the moment, we cannot know if there is an advantage, in terms of performance, with respect to centralized systems.

Another approach to data distribution on P2P environments is based on *Distributed Hash Tables (DHTs)*. This is particularly efficient for some types of resources, e.g. data files that are searched by exact name[1]. Several systems implement this solution including Tapestry [7] and Pastry [8]. They deal very well with scalability issues, but they often limit the query language to exact matches. We are verifying if our query language can be mapped to DHT or some extension thereof. Approaches based on space-filling curves, such as Squid [9], seem to offer an initial answer to this problem.

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 the use of super-peers;
– redundant configurations, when high reliability is needed.

The system is made up of two main entities (see Figure 1):

– 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.

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.

[1] To be more precise, DHT offers an effective way to find the hash keys obtained by manipulating the query string.

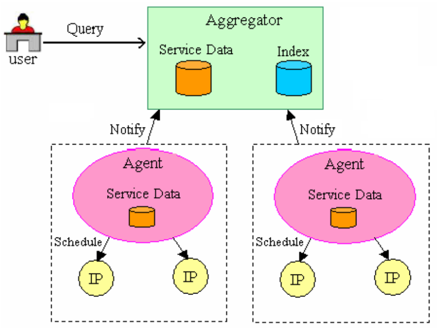


Fig. 1. Overview of our system.

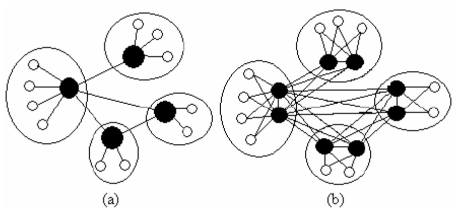


Fig. 2. Examples of super-peer networks: (a) with no redundancy, (b) with 2-redundancy. Black nodes represent super-peers. White nodes are clients. Clusters are limited by circular lines (from [10]).

3.1 Super-peer Redundant Networks

It is well known that unstructured P2P networks spend useful bandwidth in functions that can be performed by local caches [11]. This is why super-peer networks emerged as a trade-off between totally distributed systems and cache-based services [10].

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 2). This solution introduces two main benefits. (1) Replicas hold a copy of the same data. In the case of failure of one replica, the system will not stop working. (2) The workload 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 can be higher.

On the other side, communication costs may 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 gathered information 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 users choose to publish information about a given resource, they will describe the type of information using our simple taxonomy. 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[2], 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.

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. We implemented a basic set of Information Providers (*memory, processor, processorLoad, operatingSystem, and diskSpace*). 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 database. 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. This way, the super-peer will always have timely information about the clients connected to it.

3.3 Routing Index

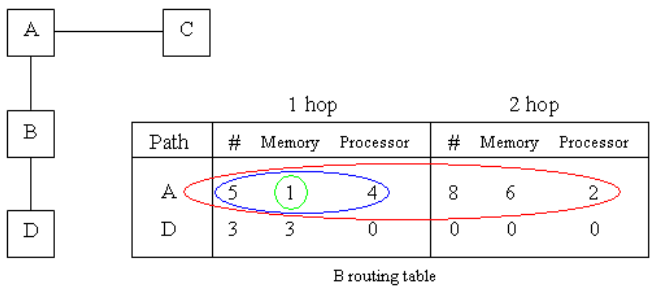
The *Routing Index* (RI) is used to improve the performance of our peer-to-peer routing, and to prevent the network from being flooded. The RI is a technique to choose the node to which a query should be forwarded: the RI represents the availability of data of a specific type in the neighbor's information base. 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 elements within a given number of hops. Data are then divided in classes by their keyword.

We used HRI as described in [2]: 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 the m-th row is the number of data elements that can be reached going through neighbor m, within n hops.

Suppose that, from node B, we are looking for data about memory (see Figure 3). Our *goodness function* (see [2]), 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 [2] to deal with cycles in the network.

[2] There may be more than one Aggregator in a redundant network.



Path	#	Memory	Processor	#	Memory	Processor
A	5	1	4	8	6	2
D	3	3	0	0	0	0

Fig. 3. HRI table for node B.

3.4 Search Technique

In literature, several techniques are used for searches in P2P networks, including flooding (e.g. Gnutella), centralized servers (e.g. Napster). More effective searches are performed by systems based on distributed indices. In these configurations, each node holds a part of the index. The index optimizes the probability of finding quickly the requested information, by keeping track of the availability of data to each neighbor.

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, in the form of a Hop-Count Routing Index. An outline of our algorithm is shown in Figure 4.

Each query is tagged with an expiration time. At each step, the expiration is checked. If the query is still valid, it is stored in a local hash table (*QueryStatus)*, with some key information. In particular, we store what is the next neighbor to try.

The HRI is used to determine which the best neighbor aggregator is for the given query. The query is forwarded to it, while it is elaborated locally, by matching the local SDEs. 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), i.e. the neighbor which has the second largest number of matching resources in the HRI. The algorithm will continue with the next best neighbor every time the query returns back (*QueryStatus*, and so *ToTry*, are increased each time).

Please note that the algorithm tries, within the given time, to find as many resources as possible. This choice is due to our goal of mimicking the behavior of the Globus Information Service. From the found resources, the user will choose those that best match his/her needs.

The current strategy suffers from two major limitations: first, under certain conditions, our algorithm may fail to find existing resources (if the query expires too early); second, it may query more Aggregators than strictly needed. Nonetheless, it offers a series of interesting features: very quick response (the

```
For each incoming query
   // check if query is still alive
   If ExpirationTime(query) < CurrentTime
      Discard

   If QueryStatus(query)=not present
      // store query in the hash table
      QueryStatus(query) := 1
      QuerySeenFirstTime := true

   ToTry := QueryStatus(query)
   // find in the Hop-Count R. Index the next
   // best neighbor of rank ToTry
   NextBestNeighbor := HRI(query, ToTry)

   If not exists NextBestNeighbor
      //the query is bounced back
      Recipient := Sender(query)
   Else
      Recipient := NextBestNeighbor
      QueryStatus(query) += 1

   Forward query to Recipient
   If QuerySeenFirstTime
      Find local matching to query
      Send local results to Requester(query)
End for
```

Fig. 4. Our search algorithm.

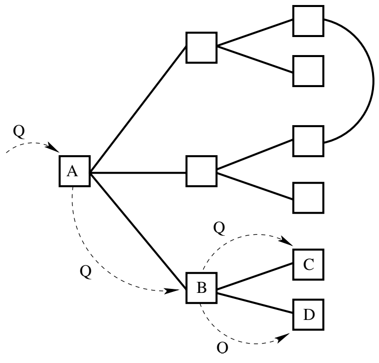


Fig. 5. A query (Q) is forwarded from A to the best neighbors (B, C, and D).

first results arrive as they are available); overlapping computation and communication; freshness of the retrieved data (which are stored very close to the resource they describe). We are investigating other algorithms, including DHT, to overcome these limitations.

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. We run two tests: the first, to compare it with Globus, and the second to verify in detail its scalability.

4.1 Comparison with Globus MDS-GT3

We compared our system with Globus MDS-GT3. The results shown in Table 1 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

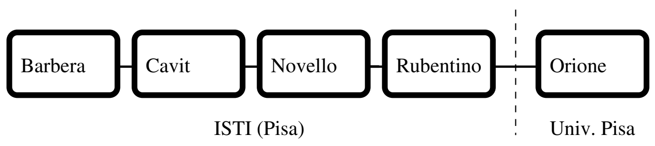


Fig. 6. Configuration for our comparison with Globus MDS. Clients are not shown.

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 utilize many machines in this test. Anyway, with these limited resources, we set up a configuration that was optimal for Globus, and very hard for our system: we created a linear chain of five Aggregators (see Figure 6), and, starting from Orione (locate at the University of Pisa), we launched queries about data down the chain (located at ISTI - Pisa). 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 its performance.

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.

Table 1. Comparison between our system (P2P) and Globus-MDS. Average response time (client-side) for subsequent results, about resources located at increasing distances (in milliseconds).

	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

4.2 System Scalability

We tested the scalability of our system by running on a Grid involving five organizations: ISTI-CNR, located in Pisa; University of Pisa; IIT-CNR, in Pisa;

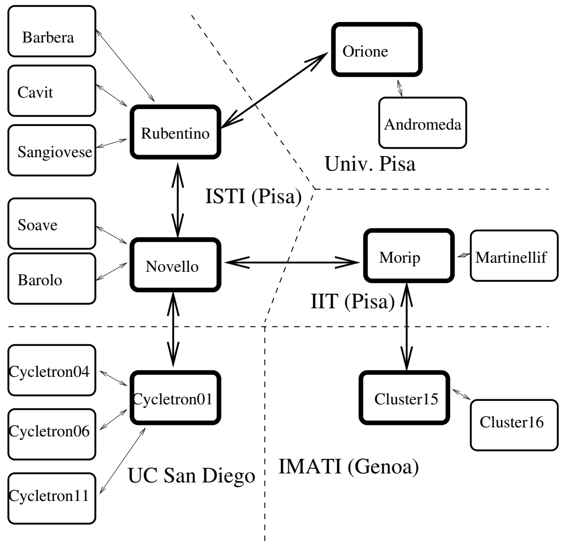


Fig. 7. Our configuration for testing scalability. An Agent is running on each machine (boxes). An Aggregator is running on thicker boxes. Arrows represent connections. The dashed line are the borders among participating institutions.

IMATI-CNR, located in Genoa; and the University of California at San Diego. The test configuration is shown in Figure 7. 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 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 take much longer to cross institutions and return to Orione. We believe that the firewall configuration, and other network effects may contribute to this large delay (see Table 2(b-c)).

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

Our system can also be used with a *redundant configuration* for improved reliability. We run some initial tests, which showed the effectiveness of this solution: when one of the replica failed, the system continued running seamlessly.

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

(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

Response time did not change significantly. We expect that, in a very large configuration, redundant peers may offer a lower response time, when they are queried alternately. We are testing this hypothesis, and results will be available in the next future.

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-Count 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.

Our system was tested using a small network, split across five different institutions. In these preliminary tests, the system scaled 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 also outperformed Globus MDS under our experimental conditions.

References

1. Plale, B., Jacobs, C., Jensen, S., Liu, Y., Moad, C., Parab, R., Vaidya, P.: 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 (2004)
2. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Proceedings of ICDCS-02. (2002)
3. Puppin, D., Moncelli, S., Baraglia, R., Tonellotto, N., Silvestri, F.: A peer-to-peer information service for the grid. In: Proceedings of the GridNets 2004 Workshop, San José, CA (2004)
4. The Globus Alliance: Globus toolkit 3, globus information services documentation (2004) Available at http://www.globus.org/mds/.
5. Talia, D., Trunfio, P.: Web services for peer-to-peer resource discovery on the grid. In: DELOS Workshop Digital Library Architectures. (2004) 73–84
6. Marti, S., Krishnan, V.: "carmen: A dynamic service discovery architecture". Technical Report HPL-2002-257, HP Laboratories Palo Alto (2002) Available at http://www.hpl.hp.com/techreports/2002/HPL-2002-257.pdf.
7. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **22** (2004)
8. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany (2001) 329–350
9. Schmidt, C., Parashar, M.: Enabling flexible queries with guarantees in p2p systems. IEEE Internet Computing (2004) 19–26
10. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the IEEE International Conference on Data Engineering. (2003)
11. Ripeanu, M.: Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, Department of Computer Science (2001)