A P2P Grid Services-Based Protocol: Design and Evaluation

Domenico Talia and Paolo Trunfio

DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{talia,trunfio}@deis.unical.it

Abstract. Several aspects of today's Grids are based on centralized or hierarchical services. However, as Grid sizes increase from tens to thousands of hosts, functionalities should be decentralized to avoid bottlenecks and guarantee scalability. A way to ensure Grid scalability is to adopt Peer-to-Peer (P2P) models and techniques to implement nonhierarchical decentralized Grid services and systems. Standard P2P protocols based on a pervasive exchange of messages, such as Gnutella, appear to be inadequate for OGSA Grids, where peers communicate among them through Grid Services mechanisms. This paper proposes a modified Gnutella discovery protocol, named *Gridnut*, which makes it suitable for OGSA Grids. In particular, Gridnut uses appropriate message buffering and merging techniques to make Grid Services effective as a way to exchange messages in a P2P fashion. We present the design of Gridnut, and compare Gnutella and Gridnut performances under different network and load conditions.

1 Introduction

Many aspects of today's Grids are based on centralized or hierarchical services. However, as Grids used for complex applications increase their size from tens to thousands of nodes, we should decentralize their functionalities to avoid bottlenecks and ensure scalability. As argued in [1] and [2], a way to provide Grid scalability is to adopt *Peer-to-Peer* (*P2P*) models and techniques to implement nonhierarchical decentralized Grid systems.

Recently, the Grid community has undertaken a development effort to align Grid technologies with Web Services. The *Open Grid Services Architecture* (*OGSA*) defines *Grid Services* as an extension of Web Services and lets developers integrate services and resources across distributed, heterogeneous, dynamic environments and communities [3]. OGSA adopts the Web Services Description Language (WSDL) to define the concept of a Grid Service using principles and technologies from both the Grid and Web Services communities. Web Services and the OGSA both seek to enable interoperability between loosely coupled services, independent of implementation, location, or platform. The OGSA model provides an opportunity to integrate P2P models in Grid environments since it offers an open cooperation model that allows Grid entities to be composed in a decentralized way.

In [4], Fox and colleagues explore the concept of a *Peer-to-Peer Grid* designed around the integration of Peer-to-Peer and OGSA models. A Peer-to-Peer Grid is built in a *service* model, where a *service* is a Web Service that accepts one or more inputs and gives one or more results. These inputs and results are the messages that characterize the system. All the entities in the Grid (i.e., users, computers, resources, and instruments) are linked by messages, whose communication forms a distributed system integrating the component parts. In a Peer-to-Peer Grid, access to services can be mediated by "servers in the core", or by direct Peer-to-Peer interactions between machines "on the edge". The server approach best scales within pre-existing hierarchical organizations, but P2P approaches best support local dynamic interactions. The Peer-to-Peer Grid architecture is a mix of structured (Grid-like) and unstructured dynamic (P2P-like) services, with peer groups managed locally and arranged into a global system supported by core servers. A key component of a Peer-to-Peer Grid is the messaging subsystem, that manages the communication among resources, Web Services and clients to achieve the highest possible system performance and reliability.

Although Grid Services are appropriate for implementing loosely coupled P2P applications, they appear to be inefficient to support an intensive exchange of messages among tightly coupled peers. In fact Grid Services operations, as other RPC-like mechanisms, are subject to an invocation overhead that can be significant both in terms of activation time and memory/processing consumption [5]. The number of Grid Service operations that a peer can efficiently manage in a given time interval depends strongly on that overhead. For this reason, standard P2P protocols based on a pervasive exchange of messages, such as Gnutella [6], are inappropriate on large OGSA Grids where a high number of communications take place among hosts.

To overcome this limitation, we propose a modified Gnutella protocol, named *Gridnut*, which uses appropriate message buffering and merging techniques that make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P fashion. Although the pure Gnutella protocol is not scalable (since the load on each node grows linearly with the total number of queries), and several advancements have been proposed to improve the performance of decentralized search (see for instance [7]), we worked on it because it is a reference model for several more sophisticated systems to which our approach can be also applied.

Gnutella defines both a protocol to discover hosts on the network, based on the *Ping/Pong* mechanism, and a protocol for searching the distributed network, based on the *Query/QueryHit* mechanism. Here we discuss only the Gridnut discovery protocol, even if we are also designing the Gridnut search protocol.

The remainder of the paper is organized as follows. Section 2 presents the design of the Gridnut protocol focusing on message routing and buffering rules. Section 3 compares the performance of Gridnut and Gnutella protocols under different network and load conditions. Finally, Section 4 concludes the paper.

2 Gridnut Design

The two basic principles of the Gridnut protocol that make it different from Gnutella are

a) *Message buffering*: to reduce communication overhead, messages to be delivered to the same peer are buffered and sent in a single packet at regular time intervals.

b) *Collective Pong*: when a peer *B* must respond to a Ping message received from *A*, it waits to receive all the Pong messages from its neighbors, then merge them with its Pong response and send back the Pong collection as a single message to *A*.

Since the Gridnut protocol is derived from the Gnutella discovery protocol, we adopt here the Gnutella terminology. Each Grid node executes a *Gridnut servent*, i.e., an application that performs both client and server Gridnut tasks.

A Gridnut servent is composed of three logical components (see Figure 1):

- *Peer Service*: a Grid Service through which remote Gridnut servents can connect and deliver messages to this servent.
- *Client Interface*: an interface through which local users and applications can issue Grid nodes discovery requests and get results.
- *Network Module*: a component that interacts with remote Gridnut servents on the basis of the Peer Service and Client Interface input.

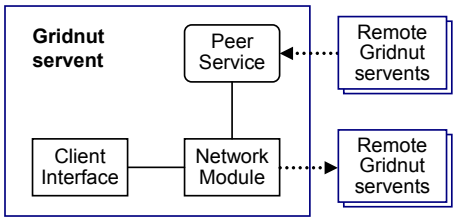

Fig. 1. Gridnut servent components

2.1 Peer Service

The Peer Service is a *persistent* Grid Service, activated at the Gridnut servent's startup and terminated when the servent leaves the network. Each Peer Service is assigned a globally unique name, the *Grid Service Handle* (*GSH*), that distinguishes a specific Grid Service instance from all other Grid Service instances. This handle is used within a Gridnut network to uniquely identify both the Peer Service and the associated Gridnut servent. For instance, a valid handle could be:

```
http://p1.deis.unical.it:8080/ogsa/services/p2p/PeerService
```

The Peer Service supports three main *operations*:

- `connect`: used by a remote servent to connect this servent. The operation receives the *handle* of the requesting servent and returns a *reject* response if the connection is not accepted (for instance, because the maximum number of connections has been reached).
- `disconnect`: used by a remote servent to disconnect this servent. The operation receives the *handle* of the requesting servent.

- `deliver`: used by a connected servent to deliver messages to this servent. The operation receives the *handle* of the requesting servent and an *array of messages* to be delivered to this servent.

2.2 Messages

A servent connects itself to the Gridnut network by establishing a connection with one or more servents currently in the network (a discussion of the connection and disconnection phases is outside the scope of this paper). Once a servent joined successfully the Gridnut network, it communicates with other servents by sending and receiving *Ping* and *Pong* messages:

- A *Ping* is used to discover available nodes on the Grid; a servent receiving a Ping message is expected to respond with a Pong message.
- A *Pong* is a response to a Ping; it includes the URL of a set of reachable Gridnut servents, each one representing an available Grid node.

The logical structure of Ping and Pong messages is shown in Figure 2.

Fig. 2. Structure of Gridnut messages

The meaning of fields in Figure 2 is the following:

- *GUID* (*Global Unique Identifier*): a string identifying the message on the network.
- *TTL* (*Time To Live*): the number of times the message will be forwarded by servents before it is removed from the network.
- *Hops*: the number of times the message has been forwarded by servents.
- *Handles*: an array of zero, one or more reachable Gridnut servents' URLs.

For the purposes of this paper, Pong messages do not include further information because here we use the discovery protocol to locate all the active nodes on the Grid. The search protocol we are designing (not discussed in the paper) will be used for host characterization, discovery of needed services, etc.

2.3 Data Structures

Each Gridnut servent uses a set of data structures to perform its functions.

A *connection list* (*CL*) is used to maintain a reference to all directly connected servents (i.e., references to the connected servents' Peer Services). Entries into the CL are updated by the `connect` and `disconnect` operations.

A *routing table* (*RT*) is used to properly route messages through the network. The RT contains a set of records having a structure [*GUID*, *Handle*], used to route messages with a given *GUID* to a servent with a given *Handle*.

The results of the discovery tasks are stored into a *result set* (*RS*), that users and applications can access for their purposes.

Finally, each Gridnut servent uses a set of internal *transmission buffers*, in which messages are stored and processed before to deliver them to the proper servent. In

particular, a servent S_0 uses two separated transmission buffers for each of its neighbors:

- A *pong buffer* (B_p), in which Pong messages with an equal GUID are merged before the delivery. The notation $B_p(S_k)$ indicates the pong buffer in which S_0 inserts Pong messages directed to a servent S_k.
- A *fast buffer* (B_f), used for Ping and Pong messages that are to be fast delivered to a given servent. We use the notation $B_f(S_k)$ to indicate the fast buffer in which S_0 inserts messages directed to a servent S_k.

A thread T_k is associated to each couple of buffers $B_p(S_k)$ and $B_f(S_k)$. T_k periodically delivers the buffered messages to S_k, on the basis of the rules described below.

2.4 Routing Rules

In Gridnut, like in Gnutella, Ping messages are forwarded to all directly connected servents, whereas Pong messages are sent along the same path that carried the incoming Ping message.

However, there are two main differences between Gnutella and Gridnut message routing and transmission modalities:

1) In Gnutella implementations, messages are sent as a byte stream over TCP sockets, whereas Gridnut messages are sent through a Grid Service invocation (by means of the `deliver` operation).
2) In standard Gnutella implementations, each message is forwarded whenever it is received, whereas Gridnut messages, as mentioned before, are buffered and merged to reduce the number of Grid Service invocations and routing operations executed by each servent.

Let consider a servent S_0 having a set of neighbors $S_1...S_n$. When a neighbor delivers an array of messages to S_0, each message is processed separately by S_0 as specified below.

Let us suppose that S_0 received from S_k the message *Ping*[GUID=*g*, TTL=*t*, Hops=*h*] (this notation means that *g*, *t*, and *h* are the actual values of GUID, TTL and Hops of this Ping); S_0 performs the following operations:

```
t = t - 1; h = h + 1;
if (RT contains a record with GUID=g)
   insert a Pong[GUID=g, TTL=h, Hops=0, Handles=Ø] into Bf(Sk);
else if (t == 0)
   insert a Pong[GUID=g, TTL=h, Hops=0, Handles={S0}] into Bf(Sk);
else {
   insert a record [GUID=g, Handle=Sk] into RT;
   insert a Pong[GUID=g, TTL=h, Hops=0, Handles={S0}] into Bp(Sk);
   for (i:1..n; i ≠ k)
      insert a Ping[GUID=g, TTL=t, Hops=h] into Bf(Si);
}
```

In the following we use the term "dummy Pong" to refer to a Pong having Handles=Ø.

Let us suppose that S_0 received from S_k the message *Pong*[GUID=*g*, TTL=*t*, Hops=*h*, Handles=*H*] (where *H* is a set of servents' handles); the following operations are performed by S_0:

```
t = t - 1; h = h + 1;
if (t == 0)
   insert H into RS;
else if (RT contains a record R with GUID=g) {
   Sr = value of the Handle field of R;
   insert a Pong[GUID=g, TTL=t, Hops=h, Handles=H] into Bp(Sr);
}
```

Finally, to start a new discovery task, S_0 must perform the following operations:

```
clear RS;
g = globally unique string; t = initial TTL;
insert the record [GUID=g, Handle=S0] into RT;
for (i:1..n)
   insert a Ping[GUID=g, TTL=t, Hops=0] into Bf(Si);
```

The discovery task is completed when the RS contains the handles of all the reachable servents in the network.

2.5 Buffering Rules

Let consider again a servent S_0 connected to a set of N servents $S_1...S_n$.

Within a pong buffer $B_p(S_k)$, a set of counters are used. A counter C_g counts the number of Pong messages with GUID=g till now inserted in $B_p(S_k)$.

When a Pong P_1 having GUID=g and containing a set H_1 of Handles is inserted into $B_p(S_k)$, the following operations are performed:

```
Cg = Cg + 1;
if (Bp(Sk) contains a Pong P0 with GUID=g) {
   add H1 to the current Handles set of P0;
   if (Cg >= N)
      mark P0 as ready;
}
else {
   insert P1 into Bp(Sk);
   if (Cg >= N)
      mark P1 as ready;
}
```

Whenever a Pong message is marked as *ready*, it can be delivered to the servent S_k. To avoid blocking situations due to missed Pong messages, a Pong could be marked as ready also if a *timeout* has been reached. In the following we do not consider failure situations, therefore no timeouts are used.

Differently from a pong buffer, messages inserted into a fast buffer $B_f(S_k)$ are immediately marked as *ready* to be delivered to S_k.

As we have mentioned before, a thread T_k is used to periodically deliver the buffered messages to S_k. In particular, the following operations are performed by T_k every time it is activated:

```
get the set of ready messages M from Bp(Sk) and Bf(Sk);
deliver M to Sk through a single deliver operation;
```

The time interval I_a between two consecutive activations of T_k is a system parameter. In the worst case, exactly a `deliver` operation can be invoked by S_0 for each of its N neighbors. Therefore, the maximum number of `deliver` operation invoked by S_0

during an interval of time I is equal to $(I / I_a) \times N$. Obviously, increasing the value of I_a the number of `deliver` operations can be reduced, but this could produce a delay in the delivery of messages. In our prototype we use I_a=*5 msec*.

3 Performance Evaluation

In this section we compare some experimental performance results of Gridnut and Gnutella protocols. To perform our experiments we developed a Java prototype of a Gridnut servent, which can also work as a standard Gnutella servent for comparison purposes. In our prototype the Peer Service is an object accessed through Remote Method Invocation (RMI). The goal of our tests is to verify how significantly Gridnut reduces the workload - number of Grid Service operations - of each peer. In doing this, we compared Gridnut and Gnutella by evaluating two parameters:

1) *ND*, the average number of `deliver` operations processed by a servent to complete a discovery task. In particular, $ND = P / (N \times T)$, where: P is the total number of `deliver` operations processed in the network, N is the number of servents in the network, and T is the overall number of discovery tasks completed.
2) $ND(d)$, the average number of `deliver` operations processed by servents that are at distance d from the servent S_0 that started the discovery task. For instance: $ND(0)$ represents the number of `deliver` operations processed by S_0; $ND(1)$ represents the number of `deliver` operations processed by a servent distant one hop from S_0.

Both *ND* and $ND(d)$ have been evaluated considering seven different network topologies. We distinguished the network topologies using a couple of numbers $\{N,C\}$, where N is the number of servents in the network, and C is the number of servents directly connected to each servent (i.e., each servent has exactly C neighbors). The network topologies we experimented are characterized by $\{N,C\}$ respectively equal to {10,2}, {10,4}, {30,3}, {30,4}, {50,4}, {70,4} and {90,4}. Notwithstanding the limited number of used servents, the number of exchanged messages among servents was extremely high and performance trends are evident.

Resulting networks were completely connected, i.e., each servent can reach any other servent in the network in a number of steps lower or equal than TTL.

3.1 Number of Deliver Operations

For each network topology, we measured *ND* under four load conditions. We use R to indicate the number of discovery tasks that are initiated in the network at each given time interval. The following values for R have been used: 1, 3, 5 and 10. In particular,

- R=1 indicates that, at each time interval, only one discovery task is initiated, therefore only messages with a given GUID are simultaneously present in the network;
- R=10 indicates that, at each time interval, ten discovery tasks are initiated, therefore messages with up to ten different GUID are simultaneously present in the network.

Table 1a and Table 1b report the *ND* measured in Gnutella and Gridnut networks, respectively. *ND* values are measured for network topologies ranging from {10,2} to {90,4}, under load conditions ranging from R=1 to R=10.

Table 1a. ND in Gnutella networks

{N,C}	{10,2}	{10,4}	{30,3}	{30,4}	{50,4}	{70,4}	{90,4}
R=1	3.60	4.53	4.91	5.49	6.00	6.27	6.52
R=3	3.61	4.54	4.95	5.48	6.01	6.32	6.53
R=5	3.61	4.55	4.96	5.47	6.01	6.35	6.54
R=10	3.60	4.54	4.99	5.49	6.02	6.35	6.53

Table 1b. ND in Gridnut networks

{N,C}	{10,2}	{10,4}	{30,3}	{30,4}	{50,4}	{70,4}	{90,4}
R=1	2.12	5.91	3.86	5.74	5.75	5.72	5.73
R=3	1.96	4.54	3.48	4.81	4.76	4.70	4.89
R=5	1.85	3.98	3.11	4.28	4.22	4.16	4.03
R=10	1.70	2.93	2.52	3.19	3.22	3.10	2.91

In Gnutella (see Table 1a), *ND* is not influenced by the *R* factor, apart from little variations due to measurements errors. This is because in Gnutella no buffering strategies are adopted, and one `deliver` operation is executed to move exactly one message in the network. Obviously, the value of *ND* increases with the size of the network, ranging from an average value of 3.61 in a {10,2} network, to an average value of 6.53 in a {90,4} network.

In Gridnut (see Table 1b), *ND* depends from both network topology and load condition. For a given value of *R*, *ND* mainly depends from the value of *C* (number of connections per servent), whereas it varies a little with the value of *N* (number of servents). For instance, if we consider the value of *ND* for R=1, we see that it varies in a small range (from 5.72 to 5.91) for all the networks with C=4.

If we consider networks with the same value of *N*, we see that *ND* decreases when the value of *C* is lower. For instance, the *ND* for a network {10,2} is lower than the *ND* for a network {10,4}, with any value of *R*. Moreover, because a single `deliver` operation is performed to deliver more buffered messages, for a given topology the value of *ND* decreases when *R* increases.

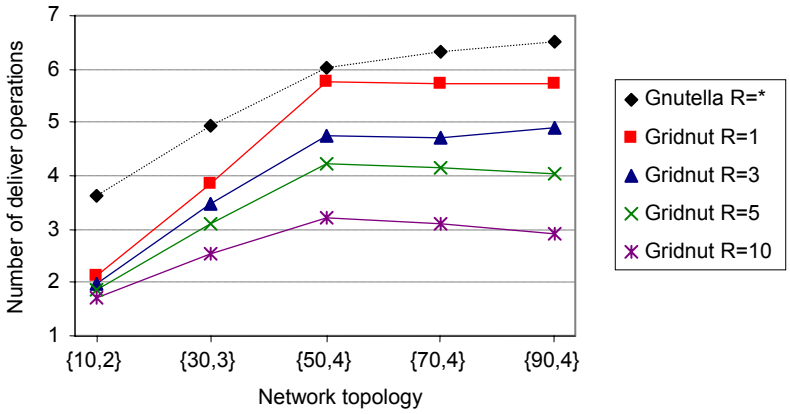

Fig. 3. Comparison between ND in Gridnut networks and ND in Gnutella networks

Figure 3 compares the values of *ND* in Gridnut and Gnutella in five network topologies: {10,2}, {30,3}, {50,4}, {70,4} and {90,4}. For Gridnut networks the

values of *ND* when *R*=1, 3, 5, and 10 are represented, whereas for Gnutella networks the average of the *ND* values measured when *R*=1, 3, 5, and 10 is represented.

We can see that the number of `deliver` operations is lower with Gridnut in all the considered configurations. In particular, when the number of discovery tasks increases, the Gridnut strategy maintains the values of ND significantly low in comparison with Gnutella.

3.2 Distribution of Deliver Operations

Table 2a and Table 2b report the value of *ND*(*d*) measured in Gnutella and Gridnut networks, respectively. Notice that in the {10,4} network the maximum distance between any couple of servents is 2, therefore no values have been measured for $d > 2$. For analogous reasons, there are no values for $d > 4$ in {30,3}, {30,4} and {50,4} networks.

Table 2a. ND(d) in Gnutella networks

{N,C}	{10,2}	{10,4}	{30,3}	{30,4}	{50,4}	{70,4}	{90,4}
d=0	9.00	9.00	29.00	29.00	49.00	69.00	89.00
d=1	4.50	4.08	9.67	7.82	12.44	17.28	22.50
d=2	3.50	4.00	4.39	4.32	5.53	6.72	8.20
d=3	2.50	–	3.04	4.00	4.11	4.41	4.46
d=4	2.00	–	3.00	4.00	4.00	4.01	4.02
d=5	2.00	–	–	–	–	4.00	4.00

Table 2b. ND(d) in Gridnut networks

{N,C}	{10,2}	{10,4}	{30,3}	{30,4}	{50,4}	{70,4}	{90,4}
d=0	2.00	4.00	3.00	4.00	4.00	4.00	4.00
d=1	2.00	5.35	3.00	4.51	4.07	4.04	4.22
d=2	2.00	6.76	3.07	5.40	5.20	4.89	4.52
d=3	2.01	–	4.05	6.40	5.84	5.61	5.50
d=4	2.34	–	4.80	6.82	6.65	6.32	6.26
d=5	2.82	–	–	–	–	6.78	6.67

In Gnutella (see Table 2a) the value of *ND*(*0*) is always equal to *N-1*. This is because S_0 receives, through its neighbors, a Pong message from each of other servents in the network, and each of those messages are delivered to S_0 by means of a separated `deliver` operation. *ND*(*1*) is always greater or equal than *ND*(*0*) divided by *C*. The equality is obtained only for networks in which *C* is sufficiently little compared to *N*, as in {10,2} and {30,3} networks. In general, the value of *ND*(*d*) decreases when *d* increases, and it reaches the minimum value, equal to *C*, on the servents more distant from S_0.

In Gridnut (see Table 2b) the value of *ND*(*0*) is always equal to *C*, because S_0 must process exactly a `deliver` operation for each servent directly connected to it. The value of *ND*(*d*) increases slightly with *d*, reaching its maximum on the servents more distant from S_0. *ND*(*d*) increases with *d* because the number of "dummy Pong" messages increase moving away from S_0. Anyway, the value of *ND*(*d*) remains always of the order of *C*, even for *d* equal to TTL.

Comparing the results in Tables 2a and 2b, we can see that Gridnut implies a much better distribution of `deliver` operations among servents in comparison with Gnutella. In Gnutella, the servent that started the discovery task and its closest neighbors must process a number of Grid Service operations that becomes unsustainable when the size of the network increases to thousands of nodes. In Gridnut, conversely, the number of Grid Service operations processed by each servent remains always in the order of the number of connections per peer. This Gridnut behaviour results in significantly lower discovery times since communication

and computation overhead due to Grid Services invocations are considerably reduced as shown in Tables 2a and 2b. For example, considering a {90,4} network with *R* ranging from 1 to 10, Gnutella discovery experimental times vary from 2431 to 26785 msec, whereas Gridnut times vary from 2129 to 8286 msec.

4 Conclusions

The Gridnut protocol modifies the Gnutella discovery protocol to make it suitable for OGSA Grids. It uses message buffering and merging techniques to make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P mode. We compared Gridnut and Gnutella performance considering different network topologies and load conditions. Experimental results show that appropriate message buffering and merging strategies produce significant performance improvements, both in terms of number and distribution of Grid Service operations processed.

We are extending Gridnut to support also distributed search by modifying the original Query/QueryHit Gnutella mechanism. In doing this, the buffering mechanism is maintained, whereas the collection mechanism is modified since the number of responding nodes will be limited by the query constraints.

The Gridnut protocol can be an effective way to discover active nodes in a OGSA Grids. Currently we are designing a *Presence Management Service* (*PMS*) that uses Gridnut as mechanism to discover active Grid nodes in a P2P fashion. Presence management is a key aspect in large-scale Grids, in which hosts join and leave the network dynamically over the time, as in typical P2P environments. The PMS allows users and schedulers to efficiently locate active nodes and support execution of large-scale distributed applications in dynamic Grid environments.

References

1. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. 2nd International Workshop on Peer-to-Peer Systems, Berkeley (2003)
2. Talia, D., Trunfio, P.: Toward a Synergy between P2P and Grids. IEEE Internet Computing, vol. 7 n. 4 (2003) 94-96
3. Foster, I., Kesselman, C., Nick, J. M., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/research/papers/ogsa.pdf
4. Fox, G., Gannon, D., Ko, S., Lee, S., Pallickara, S., Pierce, M., Qiu, X., Rao, X., Uyar, A., Wang, M., Wu, W.: Peer-to-Peer Grids. http://grids.ucs.indiana.edu/ptliupages/publications/p2pGridbook.pdf
5. The Globus Alliance: Globus Toolkit 3.0 - Performance Tuning Guide. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/performance_guide.html
6. Clip2: The Gnutella Protocol Specification v.0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
7. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and Replication in Unstructured Peer-to-Peer Networks. 16th ACM Int. Conference on Supercomputing, New York (2002)