Peer-to-Peer Protocols and Grid Services for Resource Discovery on Grids

Domenico Talia [a] and Paolo Trunfio [a]

[a]DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy

Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be found and selected. Classical approaches to Grid resource discovery, based on centralized or hierarchical approaches, do not guarantee scalability in large-scale, dynamic Grid environments. On the other hand, the Peer-to-Peer (P2P) paradigm is emerging as a convenient model to achieve scalability in distributed systems and applications. This chapter describes a protocol and an architecture that adopt a pure-decentralized P2P approach to support resource discovery in OGSA-compliant Grids. In particular, the chapter describes a modified Gnutella protocol, named *Gridnut*, which uses appropriate message buffering and merging techniques to make Grid Services effective as a way to exchange discovery messages in a P2P fashion. We present the design of Gridnut, and compare Gnutella and Gridnut performances under different network and load conditions. The chapter presents also an architecture for resource discovery that adopts the Gridnut approach to extend the model of the Globus Toolkit 3 information service.

1. INTRODUCTION

The Grid computing paradigm is today broadly applied to many scientific and engineering application fields, and is attracting a growing interest from business and industry. At the same time, *Peer-to-Peer* (*P2P*) computing is emerging as an important paradigm for developing distributed systems and applications.

Many aspects of today's Grids are based on centralized or hierarchical services. However, as Grids used for complex applications increase their sizes, it is necessary decentralize their functionalities to avoid bottlenecks and ensure scalability. As argued in [1], [2] and in other work, a way to improve scalability in large-scale Grids is to adopt P2P models and techniques to implement non-hierarchical decentralized services.

Within the Grid community, the *Open Grid Services Architecture* (*OGSA*) model is being widely adopted to achieve integration and interoperability among the increasing number of Grid applications. OGSA defines *Grid Services* as an extension of Web Services [3] to take advantage of important Web Services properties, such as service description and discovery, automatic generation of client and service code, compatibility with emerging standards and tools, and broad commercial support [4].

The OGSA model does not only support client-server applications, but it 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. A core Grid functionality that could be effectively redesigned using the P2P paradigm is *resource discovery*. Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be searched, discovered and selected.

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 in terms of memory consumption, processing time, and bandwidth requirements. 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, pure decentralized P2P protocols based on a pervasive exchange of messages, such as Gnutella [5], are inappropriate on large OGSA Grids where a high number of communications take place among hosts. On the other hand, this class of protocols offer useful properties in dealing with the high heterogeneity and dynamicity of Grid resources.

To take advantage of the pure decentralized approach and, at the same time, controlling the bandwidth consumption rate, we proposed a modified Gnutella protocol, named *Gridnut* [6], which uses appropriate message buffering and merging techniques that make Grid Services effective as a way for exchanging discovery messages among Grid nodes in a P2P fashion. 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.

We simulated the protocol by implementing a Java prototype of a Gridnut peer, which can also work as a standard Gnutella peer for comparison purposes. To verify how significantly Gridnut reduces the workload of each peer, we evaluated the Gridnut and Gnutella behaviors in different network topologies and load conditions.

The Gridnut approach can be an effective way to discover active nodes and support resource discovery in OGSA Grids. We designed an architecture for resource discovery that adopts such an approach to extend the model of the Globus Toolkit 3 (GT3) information service.

The remainder of the chapter is organized as follows. Section 2 presents the main features of Grid Services and discusses their performances in supporting the exchange of messages among tightly-coupled applications. Section 3 discusses the use of P2P models and techniques for Grid resource discovery. Section 4 presents the design of the Gridnut protocol focusing on message routing and buffering rules. Section 5 compares the performance of Gridnut and Gnutella protocols under different network and load conditions. Section 6 discusses a P2P architecture for resource discovery that extends the model of the GT3 information service. Finally, Section 7 concludes the chapter.

2. GRID SERVICES FEATURES AND PERFORMANCES

The goal of OGSA is to provide a well-defined set of basic interfaces for the development of interoperable Grid systems and applications. The attribute "open" is used to communicate architecture extensibility, vendor neutrality, and commitment to a community standardization process [4].

OGSA adopts Web Services as basic technology. Web Services are an important paradigm focusing on simple, Internet-based standards, such as the *Simple Object Access Protocol* (*SOAP*) [7] and the *Web Services Description Language* (*WSDL*) [8], to address heterogeneous distributed computing. Web services defines techniques for describing software components to be accessed, methods for accessing these components, and discovery mechanisms that enable the identification of relevant service providers.

In OGSA every resource (e.g., computer, storage, program) is represented by a service, i.e., a network enabled entity that provides some capability through the exchange of messages. More specifically, OGSA represents everything as a *Grid Service*: a Web Service that conforms to a set of conventions and supports standard interfaces. This service-oriented view addresses the need for standard interface definition mechanisms, local and remote transparency, adaptation to local OS services, and uniform service semantics [9].

OGSA defines standard mechanisms for creating, naming, and discovering transient Grid Service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. OGSA also defines mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification.

A first specification of the concepts and mechanisms defined in the OGSA is provided by the *Open Grid Services Infrastructure* (*OGSI*) [10], of which the open source *Globus Toolkit 3* [11] is the reference implementation.

The research and industry communities, under the guidance of the *Global Grid Forum* (*GGF*) [12], are contributing both to the implementation of OGSA-compliant services, and to evolve OGSA toward new standards and mechanisms. As a result of this process, the *WS-Resource Framework* (*WSRF*) was recently proposed as a refactoring and evolution of OGSI aimed at exploiting new Web Services standards, and at evolving OGSI based on early implementation and application experiences [13].

WSRF provides the means to express state as stateful resources and codifies the relationship between Web Services and stateful resources in terms of the *implied resource pattern*, which is a set of conventions on Web Services technologies, in particular XML, WSDL, and *WS-Addressing* [14]. A stateful resource that participates in the implied resource pattern is termed a *WS-Resource*. The framework describes the WS-Resource definition and association with the description of a Web Service interface, and describes how to make the properties of a WS-Resource accessible through a Web Service interface. Despite OGSI and WSRF model stateful resources differently - as a Grid Service and a WS-Resource, respectively - both provide essentially equivalent functionalities. Both Grid Services and WS-Resources, in fact, can be created, addressed, and destroyed, and in essentially the same ways [15].

As mentioned before, Grid Services operations are subject to an invocation overhead that can be significant both in terms of memory/processing consumption and bandwidth

requirements. The goal of this section is to evaluate, in particular, the performances of Grid Services in supporting the exchange of messages among tightly-coupled applications. To this end we developed a Grid Service S and a client application C:

- S exports one operation, called `deliver`, which receives in input an *array of messages* to be delivered to it.
- C invokes the `deliver` operation to deliver one or more messages to S.

The client C was executed on a node N_c, while the service S was executed on a node N_s using the Globus Toolkit 3. By using a different number of input messages we measured both the generated *network traffic* and the *execution time* needed to complete a `deliver` operation. In particular, tests have been performed with a number of messages per operation ranging from 1 to 1024, where each message has a length of 100 bytes. Each single test was run 100 times. The traffic and time values reported in each row of Table 1 and Table 2 are computed as an average of the 100 values measured in the tests.

Table 1
Network traffic generated by a `deliver` operation for different number of messages.

Number of messages per `deliver` operation	Mean traffic per `deliver` operation (byte)	Mean traffic per message (byte)
1	2613	2613.0
2	2740	1370.0
4	2995	748.75
8	3635	454.38
16	4652	290.75
32	6948	217.13
64	11408	178.25
128	20330	158.83
256	37840	147.81
512	72134	140.89
1024	140988	137.68

Table 1 reports the network traffic measured between N_c and N_s when the `deliver` operation of S is invoked by C. The second column reports the mean traffic per operation, whereas the third column reports the mean traffic per message delivered. The values in the third column are obtained by dividing the mean traffic per operation by the number of messages per operation.

The traffic per operation is the sum of a fixed part (of about 2500 bytes) and a variable part that depends from the number of messages. For instance, the delivery of a single message (100 bytes) generates 2613 bytes of traffic, while the delivery of two messages (2×100 bytes) requires 2740 bytes. The fixed overhead is mainly due to the Grid

Service invocation mechanism, which uses SOAP messages for requests to the server and responses to the client. Obviously, by increasing the number of messages per operation the traffic per message decreases, since a single SOAP envelope is used to transport more application-level messages. In particular, the mean traffic per message passes from 2613 bytes for one message to 137.68 bytes for 1024 messages, as shown in Table 1.

Table 2
Execution time of a `deliver` operation for different number of messages.

	LAN		WAN	
Number of messages per `deliver` oper.	Mean time per `deliver` oper. (msec)	Mean time per message (msec)	Mean time per `deliver` oper. (msec)	Mean time per message (msec)
1	5.60	5.60	62.68	62.68
2	5.71	2.86	65.34	32.67
4	5.88	1.47	67.44	16.86
8	6.25	0.781	70.12	8.765
16	7.12	0.445	75.63	4.727
32	8.33	0.260	90.05	2.814
64	11.25	0.176	113.21	1.769
128	16.71	0.131	144.93	1.132
256	28.90	0.113	197.14	0.770
512	55.70	0.109	291.07	0.568
1024	107.38	0.105	558.86	0.546

Table 2 reports the time needed to complete a `deliver` operation, measured in two configurations:

- *LAN*: N_c and N_s are connected by a 100 Mbps direct link, with an average RTT (Round Trip Time) equal to 1.41 msec.
- *WAN*: N_c and N_s are connected by a WAN network, with a number of hops equal to 10, bottleneck bandwidth equal to 1.8 Mbps, and an average RTT equal to 28.3 msec.

For each configuration, execution times are reported in Table 2 both per operation and per message delivered.

In the LAN configuration the execution time of a `deliver` operation ranges from 5.60 msec for one message to 107.38 msec for an array of 1024 messages, whereas in the WAN configuration the execution time passes from 62.68 msec for one message to 558.86 msec for 1024 messages. As before, the execution time is the sum of a fixed part - that includes the network latency - and a variable part. As the number of messages per operation increases, the mean time per message decreases, ranging from 5.60 msec for one message to 0.105 msec for 1024 messages in the LAN configuration. This is even more evident in

the WAN configuration, in which the mean execution time ranges from 62.68 msec for one message to 0.546 msec for 1024 messages.

To better evaluate the performances of Grid Services in supporting the delivery of messages, we can compare two opposite cases: *i*) n `deliver` operations are executed to deliver n messages (*one message per operation*); *ii*) one `deliver` operation is executed to deliver n messages (n *messages per operation*).

In the following, the term *serial time* indicates the sum of the times needed to execute n operations in sequence, and *parallel time* indicates the time needed to complete n operations executed concurrently. All the execution times are referred to the LAN configuration.

For instance, considering $n = 16$ messages to be delivered, we have the following performances:

- *one message per operation*: the overall traffic is $2613 \times 16 = 41808$ bytes; the serial time is $5.60 \times 16 = 89.6$ msec; the parallel time is 65.43 msec.
- n *messages per operation*: the overall network traffic is 4652 bytes (37156 bytes less than the first case, saving the 88.9% of traffic); the overall execution time is 7.12 msec (58.31 msec less than the parallel time of the first case, saving the 89.1% of time).

Moreover, considering $n = 64$ we have:

- *one message per operation*: the overall traffic is $2613 \times 64 = 167232$ bytes; the serial time is $5.60 \times 64 = 358.4$ msec; the parallel time is 186.6 msec.
- n *messages per operation*: the overall network traffic is 11408 bytes (saving the 93.2% of traffic); the overall execution time is 11.25 msec (saving the 94.0% of time).

Tests results show that by decreasing the number of processed Grid Service operations (for a given number of messages to be delivered), both the overall traffic generated and the delivery time are substantially reduced. The Gridnut protocol, described in Section 4, makes use of message buffering and merging techniques that produce significant performance improvements, both in terms of number and distribution of Grid Service operations processed, as discussed in Section 5.

3. P2P AND GRID RESOURCE DISCOVERY

While P2P and Grids share the same focus on harnessing resources across multiple administrative domains, they differ in many respects: Grids address support for a variety of applications and therefore focus on providing infrastructure with quality-of-service guarantees to moderate-sized, homogeneous, and partially trusted communities. In contrast, P2P systems concentrate on providing support for intermittent participation in vertically integrated applications for significantly larger communities of untrusted, anonymous individuals.

However, the convergence of the two systems is increasingly visible: the two research communities started to acknowledge each other by forming multiple research groups that study the potential lessons that can be exchanged; P2P research focuses more and more on providing infrastructure and diversifying the set of applications; Grid research is starting to pay particular attention to increasing scalability.

In [1] Foster and Iamnitchi compare and contrast Grid and P2P computing, reviewing their target communities, resources, scale, applications, and technologies. On the basis of this review, they argue that both Grids and P2P networks are concerned, in essence, with the same general problem: the organization of resource sharing within virtual communities. The complementary nature of the strengths and weaknesses of Grids and P2P suggests that an integration between the two computing models will tend to accelerate progress in both disciplines.

As pointed out before, 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. As a significant example, Fox and colleagues explored the concept of a *Peer-to-Peer Grid* designed around the integration of Peer-to-Peer and OGSA models [16]. 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.

In [2] we outlined some areas where a P2P approach can produce significant benefits in Grid systems. These include security, connectivity, fault tolerance, access services, resource discovery and presence management. In particular, the P2P model is proposed as a practical approach to implement resource discovery on the Grid.

Grid users and applications need to get information about dynamic resources status such as current CPU load, available disk space, free memory, job queue length, network bandwidth and load, and other similar information. All this information is necessary to efficiently configure and run applications on Grids. As the Grid size increases, hierarchical approaches to Grid information systems, do not guarantee scalability and fault tolerance. As mentioned before, a practical approach towards scalable solutions is offered by P2P models. Some P2P systems for resource discovery in distributed systems and Grid environments have been proposed (see for instance [17] [18] [19] [20]).

P2P resource sharing systems can be classified in two categories: *unstructured networks*, in which the placement of data is completely unrelated to the network topology, and

structured networks, in which the topology is tightly controlled and pointers to data items are placed at precisely specified locations. Structured P2P networks generally make use of distributed hash tables (DHTs) to perform mappings from keys to locations in an entirely distributed manner. Examples of unstructured networks are Gnutella and Morpheus [21]; examples of structured networks include Chord [22], CAN [23] and Tapestry [24].

Structured P2P networks are designed to locate objects with complete identifiers. Some recent structured systems provide support also for keyword search, multi-attribute, and range queries [25] [20] [26]. However, structured approaches are not well suited to handle decentralized contents about dynamic Grid resources whose values change continuously over the time and that need to be computed when requested. For this reason, we adopt an unstructured P2P approach, which allow for handling highly dynamic information, at the cost of a high bandwidth requirement for searching the network. To control the bandwidth consumption we use appropriate buffering and merging techniques, as described in the next section.

4. PROTOCOL DESIGN

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

1. *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.

2. *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.

Each peer in the network executes a Grid Service, called *Peer Service*, through which remote peers can connect and deliver messages to it.

The Peer Service is a persistent Grid Service, activated at the peer's startup and terminated when the peer 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 peer to which it is associated. For instance, a valid handle could be:

```
http://node1.deis.unical.it:8080/ogsa/services/p2p/PeerService
```

The Peer Service supports four operations:

- `connect`: used by a remote peer to connect this peer. The operation receives the *handle* of the requesting peer 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 peer to disconnect this peer. The operation receives the *handle* of the requesting peer.

- `deliver`: used by a connected peer to deliver messages to this peer. The operation receives the *handle* of the requesting peer and an *array of messages* to be delivered to this peer.
- `query`: invoked by a client application to submit a query to this peer. Query responses are returned to the client through a notification mechanism.

4.1. Messages

A peer connects itself to the Gridnut network by establishing a connection with one or more peers currently in the network (a discussion of the connection and disconnection phases is given in Section 6). Once a peer joined successfully the Gridnut network, it communicates with other peers by sending and receiving Ping and Pong messages:

- A Ping is used to discover available nodes on the Grid; a peer 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 Peer Services, each one representing an available peer (or Grid node).

The logical structure of Ping and Pong messages is shown in Figure 1.

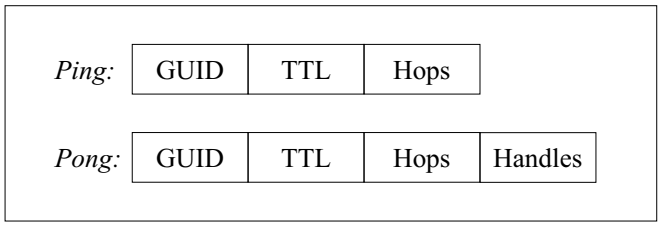

Figure 1. Structure of Gridnut messages.

The meaning of the fields in Figure 1 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 peers before it is removed from the network.
- *Hops*: the number of times the message has been forwarded by peers.
- *Handles*: an array of zero, one or more reachable Peer Services' URLs.

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

4.2. Data structures

Each peer uses a set of logical data structures to perform its functions.

A *connection list* (*CL*) is used to maintain a reference to all directly connected 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 Peer Service with a given *Handle*.

The results of the discovery tasks are stored into a *result set* (*RS*).

Finally, each peer uses a set of internal *transmission buffers*, in which messages are stored and processed before to deliver them to the proper Peer Service. In particular, a peer S_0 uses two separated transmission buffers for each of its neighbors:

- A *pong buffer* (B_p), in which Pong messages with the same 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 Peer Service S_k.
- A *fast buffer* (B_f), used for Ping and Pong messages that are to be fast delivered to a given Peer Service. We use the notation $B_f(S_k)$ to indicate the fast buffer in which S_0 inserts messages directed to a Peer Service 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.

4.3. Routing rules

In Gridnut, like in Gnutella, Ping messages are forwarded to all directly connected peers, whereas Pong messages are sent along the same path that carried the incoming Ping message. The Hops value is increased each time a Ping is forwarded, and whenever a Pong is sent in response to a Ping, the Hops value is assigned to the TTL field, so that the TTL will hold the number of hops to reach the source of the Ping. Hence, the TTL will be 0 when the result reaches the source of the discovery 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 (based on version 0.4 of the protocol [5]), 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 peer.

Consider a peer 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=∅] into B_f(S_k);
else if (t == 0)
    insert a Pong [GUID=g, TTL=h, Hops=0, Handles=S_0] into B_f(S_k);
else {
    insert a record [GUID=g, Handle=S_k] into RT;
    insert a Pong[GUID=g, TTL=h, Hops=0, Handles=S_0] into B_p(S_k);
    for (i:1..n; i ≠ k)
       insert a Ping [GUID=g, TTL=t, Hops=h] into B_f(S_i);
}
```

First of all - as shown above - the TTL and Hops values of this message are updated. Then, if the message is a duplicated Ping (since the routing table already contains its GUID), a "dummy Pong" (i.e., having Handles=$\emptyset$) is fast delivered to S_k. Else, if this Ping terminated its TTL, it is not further forwarded, and a Pong response is fast delivered to S_k. In the last case, first the routing table is updated, then a Pong response is inserted into the pong buffer, and finally the Ping is forwarded to all the neighbors, except the one from which it was received.

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 Peer Services' 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) {
    S_r = value of the Handle field of R;
    insert a Pong [GUID=g, TTL=t, Hops=h, Handles=H] into B_p(S_r);
}
```

As before, the TTL and Hops fields are updated. Then, if this Pong terminated its TTL (and so this peer is the final recipient), its handles are inserted into the result set. Else, the Pong is forwarded, through the corresponding pong buffer, to the proper peer, as specified by the routing table.

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=S_0] into RT;
for (i:1..n)
    insert a Ping [GUID=g, TTL=t, Hops=0] into B_f(S_i);
```

As described above, the result set is reset before anything else. Then, a Ping message is created (with a new GUID and a proper TTL) and forwarded to all the neighbors through the corresponding fast buffers. The discovery task is completed when the result set contains the handles of all the reachable peers in the network.

4.4. Buffering rules

Consider again a peer S_0 connected to a set of N peers $S_1...S_n$. 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 $P_1 = Pong$[GUID=g, TTL=t, Hops=h, Handles=H_1] is inserted into $B_p(S_k)$, the following operations are performed:

```
C_g = C_g + 1;
if (B_p(S_k) contains a Pong P_0 with GUID=g) {
    add H_1 to the current Handles set of P_0;
    if (C_g ≥ N)
      mark Pong P_0 as ready;
}
else {
    insert Pong P_1 into B_p(S_k);
    if (C_g ≥ N)
      mark Pong P_1 as ready;
}
```

Whenever a Pong message is marked as *ready*, it can be delivered to the peer 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 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 B_p(S_k) and B_f(S_k);
deliver M to S_k 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` operations invoked by S_0 during an interval of time I is equal to $(I \div I_a) \times N$. 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.

5. 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 Peer Service, which can also work as a standard Gnutella peer 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 peer to complete a discovery task. In particular, $ND = P \div (N \times T)$, where: P is the total number of `deliver` operations processed in the network, N is the number of peers in the network, and T is the overall number of discovery tasks completed.

2. $ND(d)$, the average number of `deliver` operations processed by peers that are at distance d from the peer 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 peer distant one hop from S_0.

Both ND and $ND(d)$ have been evaluated considering seven different network topologies. We distinguish the network topologies using a couple of numbers $\{N, C\}$, where N is the number of peers in the network, and C is the number of peers directly connected to each peer (i.e., each peer 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 peers, the number of exchanged messages among peers was extremely high and performance trends are evident.

Resulting networks were connected graphs, that is each peer can reach any other peer in the network in a number of steps lower or equal than TTL.

5.1. 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 3 and Table 4 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 3
ND in Gnutella networks.

	{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

In Gnutella (see Table 3), 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.

Table 4
ND in Gridnut networks.

	{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 Gridnut (see Table 4), 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 peer), whereas it varies a little with the value of N (number of peers). 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.

Comparing the results in Tables 3 and 4 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.

5.2. Distribution of `deliver` operations

Table 5 and Table 6 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 peers 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 5
ND(d) in Gnutella networks.

	{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

In Gnutella (see Table 5) 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 peers 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 peers more distant from S_0.

Table 6
ND(d) in Gridnut networks.

	{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 Gridnut (see Table 6) the value of $ND(0)$ is always equal to C, because S_0 must process exactly a `deliver` operation for each peer directly connected to it. The value of $ND(d)$ increases slightly with d, reaching its maximum on the peers 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 5 and 6 we can see that Gridnut implies a much better distribution of `deliver` operations among peers in comparison with Gnutella. In

Gnutella, the peer 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 peer remains always in the order of the number of connections per peer. This Gridnut behavior results in significantly lower discovery times since communication and computation overhead due to Grid Services invocations are considerably reduced as shown in Tables 5 and 6. 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.

6. A P2P ARCHITECTURE FOR GRID RESOURCE DISCOVERY

The Gridnut approach can offer an effective model to discover active nodes and support resource discovery in OGSA Grids. In this section we describe a framework for resource discovery that adopts such an approach to extend the model of the Globus Toolkit 3 (GT3) information service [27].

In the OGSA framework each resource is represented as a Grid Service, therefore resource discovery mainly deals with the problem of locating and querying information about useful Grid Services.

In GT3 information about resources is provided by *Index Services*. An Index Service is a Grid Service that holds information (called *service data*) about a set of Grid Services registered to it. A primary function of the Index Service is to provide an interface for querying aggregate views of service data collected from registered services. There is typically one Index Service per *Virtual Organization* (*VO*). When a VO consists of multiple large sites, very often each site runs its own Index Service that indexes the various resources available at that site. Then each of those Index Services is included in the VO's Index Service.

From the perspective of the GT3 information service, the Grid can be seen as a collection of VOs, each one indexed by a different Index Service. As mentioned before, Index Services of different sites can be included in a common higher-level Index Service that holds information about all the underlying resources. However, for scalability reasons, a multi-level hierarchy of Index Services is not appropriate as a general infrastructure for resource discovery in large scale Grids. Whereas centralized or hierarchical approaches can be efficient to index resources structured in a given VO, they are inadequate to support discovery of resources that span across many independent VOs. The framework described here adopts the P2P model to support resource discovery across different VOs.

Figure 2 shows the general architecture of the framework. Some independent VOs are represented; each VO provides one top-level *Index Service* (*IS*) and a number of lower-level Index Services.

A *P2P Layer* is defined on top of the Index Services' hierarchy. It includes two types of specialized Grid Services: *Peer Services* (introduced before), used to perform resource discovery, and *Contact Services*, that support Peer Services to organize themselves in a P2P network.

There is one Peer Service per VO. Each Peer Service is *connected* with a set of Peer Services, and exchanges query/response messages with them in a P2P mode. The connected

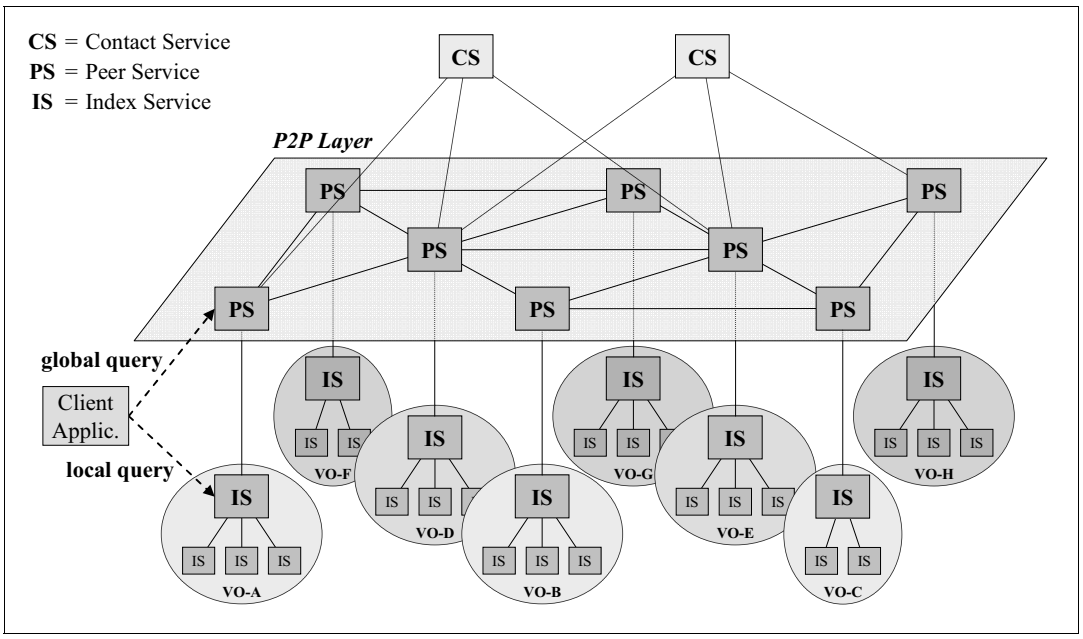

Figure 2. Framework architecture.

Peer Services are the *neighbors* of a Peer Service. A *connection* between two neighbors is a logical state that enables they to directly exchange messages. Direct communication is allowed only between neighbors. Therefore, a query message is sent by a Peer Service only to its neighbors, which in turn will forward that message to their neighbors. A query message is processed by a Peer Service by invoking the top-level Index Service of the corresponding VO. A query response is sent back along the same path that carried the incoming query message.

To join the P2P network, a Peer Service must know the URL of at least one Peer Services to connect to. An appropriate number of Contact Services is distributed in the Grid to support this issue. Contact Services cache the URLs of known Peer Services; a Peer Service may contact one or more well known Contact Services to obtain the URLs of registered Peer Services.

As shown in Figure 2, a *Client Application* can submit both *local* and *global* queries to the framework. A local query searches for information about resources in a given VO. It is performed by submitting the query to the Index Service of that VO. A global query aims at discovering resources located in possibly different VOs, and is performed by submitting the query to a Peer Service at the P2P Layer. As mentioned before, the Peer Service processes that query internally (through the associated Index Service), and will forward it to its neighbors as in typical P2P networks.

The main difference between a hierarchical system and the framework described here is the management of global queries. Basically, in a hierarchical information service two alternative approaches can be used:

- the query is sent separately to all the top-level Index Services, that must be known

by the user;

- the query is sent to one (possibly replicated) Index Service at the root of the hierarchy, that indexes all the Grid resources.

Both these approaches suffer scalability problems. In the P2P approach, conversely, global queries are managed by a layer of services that cooperate as peers. To submit a global query, a user need only to know the URL of a Peer Service in the Grid.

In the next subsection the design of the Peer Service and Contact Service components is discussed.

6.1. Services design

Both Peer Service and Contact Service instances are identified by a globally unique GSH.

Each Peer Service supports four operations: `connect`, `disconnect`, `deliver`, and `query`, as described in Section 4.

A Contact Service supports just one operation:

- `getHandles`: invoked by a Peer Service to register itself and to get the handles of one or more registered Peer Services.

Figure 3 and Figure 4 describe, respectively, the main software components of Peer Services and Contact Services.

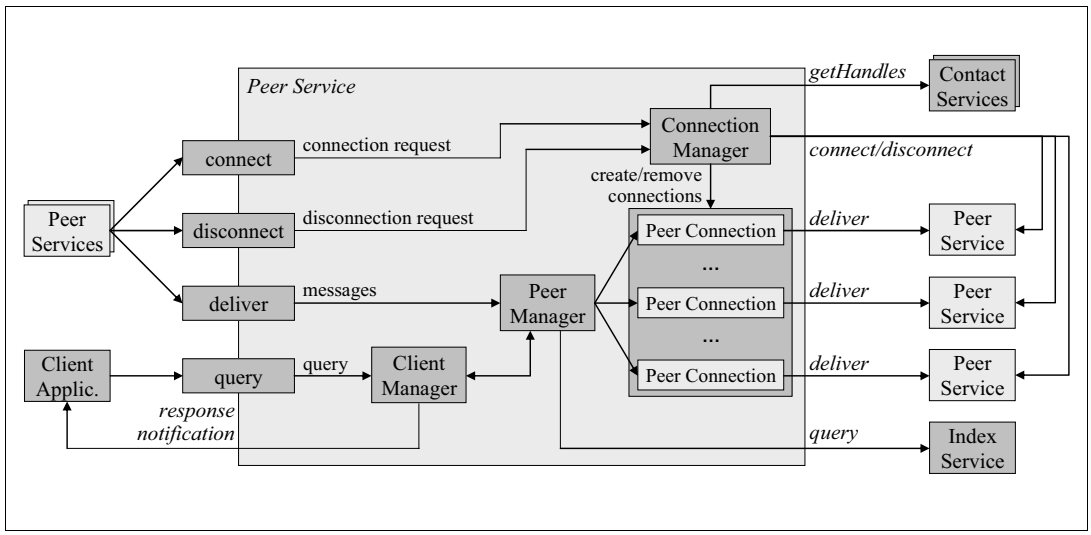

Figure 3. Peer Service software components.

The Peer Service (see Figure 3) is composed by three main modules: *Connection Manager*, *Peer Manager*, and *Client Manager*.

The goal of the Connection Manager is to maintain a given number of connections with neighbor Peer Services. A *Peer Connection* object is used to manage the connection

and the exchange of messages with a given Peer Service. A Peer Connection includes the *Grid Service Reference* (*GSR*) of a given Peer Service, and a set of *transmission buffers* for the different kinds of messages directed to it. The Connection Manager both manages connection/disconnection requests from remote Peer Services, and performs connection/disconnection requests (as a client) to remote Peer Services. Moreover, it may invoke one or more Contact Services to obtain the handles of Peer Services to connect to.

The Peer Manager is the core component of the Peer Service. It both manages the messages delivered from other Peer Services, and interacts with the Client Manager component to manage client requests and to provide responses. It performs different operations on delivered messages: some messages are simply forwarded to one or more Peer Connections, whereas query messages need also a response (that in general is obtained by querying the local Index Service). Moreover, the Peer Manager generates and submits query messages to the network on the basis of the Client Manager requests.

The Client Manager manages the query requests submitted by client applications. It interacts with the Peer Manager component to submit the query to the network, and manages the delivery of query results to the client through a notification mechanism.

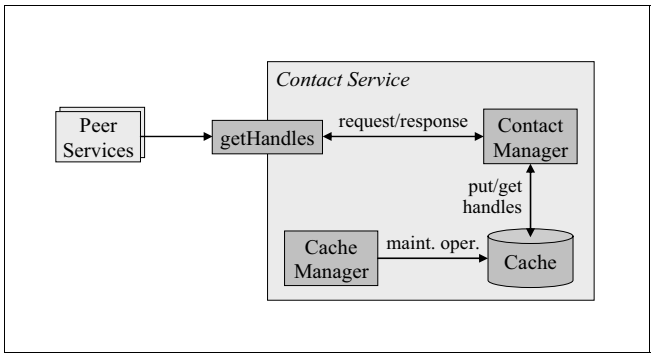

Figure 4. Contact Service software components.

The Contact Service (see Figure 4) is composed by two software modules: *Contact Manager* and *Cache Manager*.

The Contact Manager manages the execution of the `getHandles` operation. Basically, it receives two parameters: the handle h of the invoker, and the number n of handles requested by the invoker. The Contact Manager first inserts (or updates) the handle h into a *Cache*, then it extracts (if available) n distinct handles from the Cache and returns them to the invoker. The handles can be extracted from the Cache on the basis of a given policy (e.g., randomly). If a Peer Service does not receive the requested number of handles, it can try to invoke the Contact Service later.

The Cache Manager performs maintenance operations on the Cache. For instance, it removes oldest (not recently updated) handles, performs content indexing, etc.

7. CONCLUSIONS

Although P2P and Grid computing are still considered two different research areas, an integrated approach based on the merging of these two models will be profitable for the development of scalable distributed systems and applications. As Grids become very large and pervasive, the use of P2P approaches can be exploited to achieve scalability.

An important Grid functionality that could be effectively redesigned using the P2P paradigm is resource discovery. To this end, we designed a Gnutella-like discovery protocol, named *Gridnut*, which uses appropriate message buffering and merging techniques to make OGSA-compliant services effective as a way to exchange discovery messages in a P2P fashion. 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.

The Gridnut approach can be an effective way to discover active nodes and support resource discovery in OGSA Grids. The chapter described an architecture for resource discovery that adopts such an approach to extend the model of the Globus Toolkit 3 information service. In particular, a P2P Layer of specialized Grid Services is defined to support discovery queries on Index Services of multiple VOs in a P2P fashion. Since the proposed architecture adopts a general P2P service-based approach, it can be also used as a model for designing an information service in future P2P-based Grid programming environments.

REFERENCES

1. I. Foster and A. Iamnitchi, On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. Proc. 2nd International Workshop on Peer-to-Peer Systems, Berkeley, USA (2003).
2. D. Talia and P. Trunfio, Toward a Synergy between P2P and Grids. IEEE Internet Computing, vol. 7, n. 4, pp. 94-96 (2003).
3. The World Wide Web Consortium, Web Services Activity. http://www.w3.org/2002/ws.
4. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid. In: F. Berman, G. Fox, and A. Hey (eds.), Grid Computing: Making the Global Infrastructure a Reality, Wiley, pp. 217-249 (2003).
5. Clip2, The Gnutella Protocol Specification v.0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
6. D. Talia and P. Trunfio, A P2P Grid Services-Based Protocol: Design and Evaluation. Proc. European Conference on Parallel Computing (EuroPar 2004), Pisa, Italy, LNCS 3149, pp. 1022-1031 (2004).
7. D. Box et al., Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.
8. E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

9. I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, Grid Services for Distributed System Integration. IEEE Computer, vol. 35, n. 6, pp. 37-46 (2002).
10. S. Tuecke et al., Open Grid Services Infrastructure (OGSI) Version 1.0. http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf.
11. The Globus Alliance, Globus Toolkit 3. http://www.globus.org/toolkit.
12. The Global Grid Forum (GGF). http://www.ggf.org.
13. K. Czajkowski et al., The WS-Resource Framework Version 1.0. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.
14. D. Box et al., Web Services Addressing (WS-Addressing), W3C Member Submission 10 August 2004. http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810.
15. K. Czajkowski et al., From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf.
16. G. Fox, D. Gannon, S. Ko, S. Lee, S. Pallickara, M. Pierce, X. Qiu, X. Rao, A. Uyar, M. Wang, and W. Wu, Peer-to-Peer Grids. In: F. Berman, G. Fox, and A. Hey (eds.), Grid Computing: Making the Global Insfrastructure a Reality, Wiley, pp. 471-490 (2003).
17. A. Iamnitchi, I. Foster, and D. Nurmi, A Peer-to-Peer Approach to Resource Discovery in Grid Environments. Proc. 11th Int. Symposium on High Performance Distributed Computing (HPDC 11) (2002).
18. A. Andrzejak and Z. Xu, Scalable, Efficient Range Queries for Grid Information Services. Proc. 2nd Int. Conference on Peer-to-Peer Computing (P2P2002) (2002).
19. A.R. Butt, R. Zhang, and Y.C. Hu, A Self-Organizing Flock of Condors. Proc. Supercomputing Conference (SC2003) (2003).
20. M. Cai, M. Frank, J. Chen, and P. Szekely, MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing (to appear).
21. K. Truelove and A. Chasin, Morpheus Out of the Underworld. http://www.openp2p.com/pub/a/p2p/2001/07/02/morpheus.html.
22. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Proc. SIGCOMM 2001, pp. 149-160 (2001).
23. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, A Scalable Content-Addressable Network. Proc. SIGCOMM 2001, pp. 161-172 (2001).
24. B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley (2001).
25. P. Reynolds and A. Vahdat, Efficient Peer-to-Peer Keyword Searching. Proc. Int. Middleware Conference (Middleware 2003), Rio de Janeiro, Brazil (2003).
26. A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram, PTree: A P2P Index for Resource Discovery Applications. Proc. 13th Int. Conference on World Wide Web (WWW 2004) (2004).
27. D. Talia and P. Trunfio, Web Services for Peer-to-Peer Resource Discovery on the Grid. Proc. 6th Thematic Workshop of the EU Network of Excellence DELOS, S. Margherita di Pula, Italy (2004).