

# A Peer-to-Peer Approach to Resource Discovery in Grid Environments

Adriana Iamnitchi<sup>1</sup>

Ian Foster<sup>1,2</sup>

Daniel C. Nurmi<sup>1,2</sup>

<sup>1</sup>Department of Computer Science, University of Chicago

<sup>2</sup> Mathematics and Computer Science Division, Argonne National Laboratory

March 2002

## Abstract

Two resource sharing environments have polarized the distributed systems research in the last years: computational Grids and Peer-to-Peer communities. They seem likely to converge into a large-scale, decentralized, and self-configuring environment that provides complex functionalities. Resource discovery is discussed in this context: we propose a framework along 4 axes that guides the design of any resource discovery architecture. We present an emulated environment for resource discovery and preliminary performance evaluations of a simple resource discovery mechanism based on request propagation.

## 1 Introduction

In large sets of shared resources, a basic problem is locating resources in the absence of a naming scheme: that is, a resource is often described as a set of desired attributes ("a Linux machine with more than 128 MB of available memory") rather than via a globally unique identifier (such as "berlioz.cs.uchicago.edu"). In the absence of a global, fixed naming scheme, the resource discovery problem is made more challenging by variations in resource attributes (e.g., CPU load, available bandwidth, even software versions), potentially large or variable number of resources (as resources join, leave, or fail), and resource heterogeneity (resources may be computers, data, storage space, services, or some previously unheard of scientific instrument connected to the Internet).

This document proposes a study of the resource discovery problem in a resource sharing environment that combines the complexity of computational Grids with the scale and dynamism of peer-to-peer communities. While the two environments have the same final objective—to pool large sets of resources—they initially emerged from different communities, and hence their current design approaches highlight different requirements. It is our belief that the design objectives of the two environments will eventually converge and, consequently, analyzing their requirements and characteristics is important for the long term.

The resource discovery problem is presented in Section 2 in the context of the two resource sharing environments. By comparing the Grid and peer-to-peer environments, we infer a set of design decisions for a resource discovery mechanism. Section 3 proposes four components that describe any decentralized resource discovery mechanism. An emulated large-scale resource sharing environment is presented in Section 4, along with preliminary performance evaluations of a simple resource discovery solution based on request propagation.

## 2 Resource Discovery in Large-Scale Resource Sharing Environments

In recent years significant interest focused around two resource sharing environments: computational grids and peer-to-peer systems. While the two environments have the same final objective—to pool large sets

of resources—they initially emerged from different communities, and hence their current design approaches highlight different requirements. Current Grids are characterized by a large variety of services provided to (traditionally) scientific communities of hundreds of users. Peer-to-peer environments boast a larger community of users (hundreds of thousands of simultaneous users [27]) but they offer limited, specialized services. It is our belief that the design objectives of the two environments will eventually converge and, consequently, analyzing their requirements and characteristics is important for the long term.

## 2.1 Grid vs. Peer-to-Peer Environments

Computational grids [16] are sharing environments in which collections of geographically distributed hardware and software resources are made available to groups of remote users. Resources can be computers, storage space, sensors (e.g., telescopes), software applications, and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management, etc. Resources are owned by various administrative organizations and shared under locally defined policies that specify what is shared, who is allowed to share and under what conditions. A set of individuals and/or institutions defined by such sharing rules is called a Virtual Organization (VO)[17].

Grids emerged in scientific communities spanning multiple institutions, usually research labs and universities. The primary objectives in building the Grids are providing functionality to scientists: a complex set of orthogonal services for resource management, security, information services, etc. The achieved scale of such Grids is now in the order of tens of institutions with thousands of pooled computers.

Referring strictly to the deployed Grids (and not to their objectives), these resource sharing environments are currently characterized by:

1. *Scale* in the order of thousands of pooled computers and hundreds of simultaneous users.
2. *Some level of centralization*, e.g., centralized repositories for shared data.
3. Support for *complex operations*: program execution, read and write access to data, resource monitoring, security mechanisms, accountability, etc.
4. *Stability in resource participation*: with the exception of occasional failures and administrative interventions, resources in a Grid are available to the community for long, predefined periods of time.
5. Some level of *homogeneity in usage behavior*, due to incentives or policies that encourage or enforce fair sharing. In addition, trust enforcing mechanisms such as authorization and authentication ensure a selected, somehow homogeneous group of users.
6. *Homogeneity in resources*: resources are often powerful (the "traditional" grids concentrated around sharing supercomputers and large clusters), with good network connectivity.
7. Existence of *technical support personnel* capable of fixing things and willing (paid) to adapt the system to new requirements (e.g., install new software versions to ensure seamless collaboration, use human communication means to announce new usage policies, etc.)

In the last couple of years a new term became fashionable: peer-to-peer (P2P) systems. While P2P systems did not emerge from a novel conceptual idea, they emphasize two specific attributes of resource sharing communities: scale and dynamism. Compared to Grid environments, deployed P2P provide limited, specialized functionality (e.g., file sharing only [12] or embarrassingly parallel computation [38]) to larger and less homogeneous communities (half-million simultaneous Gnutella nodes reported in March 2002 by LimeWire [27], over 3 millions participants in the SETI@home [38] project so far).

Among the characteristics of P2P systems are:

1. *Large scale*: in the order of hundreds of thousands of simultaneous computers/users.
2. *Less centralization* (e.g., Gnutella is fully decentralized. However, SETI@home is centralized).

3. *Specialized functionality*: file sharing or embarrassingly parallel computations, often in the absence of trust enforcing mechanisms. Not only that functionalities in a P2P environment are less complex than in Grids, but they are often conflicting with those required in Grids: for example, anonymity vs. accountability.
4. *Unpredictable resource participation*, subject to influences of social, economical, or political nature.
5. Lack of implicit incentives for good behavior may lead to *uneven user behavior*, such as free riding [2].
6. *Highly heterogeneous resources*, from old home computers connected via modem to powerful computers connected via high bandwidth links.
7. *Lack of technical expertise and administrative authority* to coordinate efforts into a specified direction.

A natural tendency of Grids is to expand in size for larger communities. A natural tendency of P2P systems is the increase in service complexity, for example by combining computation and data sharing and providing some level of trust. It seems thus likely that the two environments will converge into large-scale, dynamic sets of resources that can be exploited in a variety of ways.

The resource sharing environment to which the two systems seem to converge is the focus of this study. This environment may have the following properties:

1. Large scale: millions of resources shared by hundreds of thousands of participants (institutions and individuals).
2. Lack of global centralized authority.
3. Highly variable participation patterns: there will be perhaps a larger number of stable nodes than in today's Gnutella [37], but many resources will join and leave the network frequently.
4. Strong diversity in:
  - (a) Shared resources: resources can be of different *types*: computers, data, services, instruments, storage space. Resources of the same type can be highly heterogeneous: e.g., computers with different operating systems, number of CPUs and speed; data of various sizes; services of various sorts.
  - (b) Sharing characteristics: some resources will have well defined, public sharing policies, such as "available to anyone from 6pm to 6am". Other resources will participate rather chaotically, e.g., when idle.
5. Lack of homogeneous administrative support: some participants will benefit from technical support, others will rely on basic tools provided for the community (e.g., today's Gnutella nodes run various implementations on the protocol, each with its own particularities).

## 2.2 Requirements for a Resource Discovery Mechanism in Large-Scale Resource Sharing Environments

A basic service in large-scale resource sharing environments is resource discovery: given a description of resources desired, a resource discovery mechanism returns a set of (contact addresses of) resources that match the description. Resource discovery is challenging because of the potentially large number of resources and users (perhaps millions) and considerable heterogeneity in resource types and user requests. Resource discovery is further complicated by the natural tendency of the environment to evolve over time, with, for example, institutions joining and leaving (along with their resources and users) and resource characteristics such as availability and CPU load changing.

These characteristics create significant difficulties for traditional resource discovery services. In a large-scale, dynamic, and heterogeneous environment as the one presented above, the principles that guide the design of the resource discovery mechanism are:

- Lack of centralized control. There are multiple arguments to sustain this assumption. First, it is possible that no incentives will exist for any participant (institution or individual) to support the significant administrative costs inherent in systems that aggregate huge numbers of resources with unpredictable behavior. Second, a central architecture will be challenged by scale and dynamism. One solution is thus to adopt a **self-configuring, distributed architecture**.
- Given the highly heterogeneous nature of these environments, organizing resources in structures such as distributed hash tables (DHTs) [34, 39, 43, 36]) may be ineffective: in DHTs, all nodes have equal responsibilities, which assumes homogeneous capabilities and trust. Another challenge for structured networks is the cost of maintaining the structure despite potentially frequent nodes departures. These are the arguments that plead for the organization of nodes into **unstructured networks**.
- In an environment with multiple types of resources and possibly undefined resource descriptions it is difficult to impose a global naming scheme. In file sharing systems, the naming scheme is implicit: a resource (i.e., a file) is uniquely identified by its (file)name. In a system that shares computing resources as well, the name of a computer (say, its IP address) does not reflect its dynamic characteristics (for example, its CPU load). We hence assume there is **no global naming scheme** for describing resources.

### 3 Discovering Generic Resources in Unstructured Networks

In the context of decentralized resource discovery in large-scale, distributed, heterogeneous systems, we can assume without loss of generality that every participant in the virtual organization—institution or individual—has one or more servers that store and provide access to resource information. We call these servers *nodes* or *peers*. A node may provide information about a small set of resources (e.g., locally stored files or the node’s computing power, as in a traditional P2P scenario) or a large set of resources (e.g., all resources shared by an institution, as in a typical Grid scenario).

From the perspective of resource discovery, the grid is thus a collection of geographically distributed nodes that may join and leave at any time and without notice (for example, as a result of system or communication failures). Although we assume the sets of resources published by nodes are disjoint, there may be multiple resources with identical descriptions, for example, multiple copies of the same data or identical machines.

Users send their requests to some known (typically local) node. Typically, the node responds with the matching resource descriptions if it has them locally, otherwise it processes the request, possibly forwarding it to other node(s).

#### 3.1 Four Axes of the Solution Space

The following components define a general resource discovery solution:

1. *Membership protocol* refers to how new nodes join the network and how nodes learn about each others (we refer to the latter part of the membership problem as to “peer discovery”, while aware that it has multiple names in the literature [25]).
2. *Overlay construction function* selects the set of active collaborators from the local membership list. In practice, this set is limited by available bandwidth, message processing load, security or administrative policies, topology specifications, etc. Hence, it may often be the case that the so called *overlay network* contains only a subset of the edges of the membership network. For example, a Gnutella node maintains a relatively small number of open connections (the average is less than 10, with 3.4 measured as of May 2001 [35]) compared to the number of Gnutella nodes that it knows at a given time (hundreds).
3. *Preprocessing* refers to the off-line preparations for better search performance, independent of requests: e.g., caching is not a preprocessing technique, while prefetching is. An example of preprocessing techniques is dissemination of resource descriptions: e.g., advertise descriptions of the local resources to other areas of the network for better search performance and reliability. Rewiring the overlay network (performed by an overlay network protocol based on the overlay construction function) for better performance may also be considered a preprocessing technique.

4. *Request processing* can have a local and a remote component:

- (a) Local processing includes looking up the requested resource in the local information, processing aggregated resources (e.g., a request for A and B could be broken into two distinct requests to be treated separately), or policy-dependent processing (drop requests that exceeded a local TTL limit or requests that are not acceptable for the local administration; alter requests, e.g., by relaxing constraints that have used a significant portion of their TTL, etc.).
- (b) Remote processing refers to the request propagation rule: sending the (potentially locally processed) requests to other peers through various mechanisms (flooding, forwarding, epidemic communication). Various strategies can be employed as to how many neighbors to send a request and how to select them.

We study the resource discovery problem along these 4 axes: the membership protocol is responsible for collecting and updating information about the currently participating nodes; the overlay function selects the best (according to some metrics) set of members from the known ones and hence influences the topology of the overlay network; preprocessing makes the necessary preparations for a more efficient search; the request processing strategy performs the search itself.

### 3.2 Discussion

The general objective of this study is to characterize the synergies between the four components of a generic resource discovery solution with regard to the environment characteristics. This section discusses some of the open research questions and relevant results in the context of the 4 design axis.

The overlay topology has a significant impact on the performance at upper level applications: e.g., [4] shows a strong correlation between robustness and the power-law topology of the underlying network; [1] gives a search algorithm that exploits the power-law topology in a cost-efficient way; [24] presents an optimal algorithm for search in small-world graphs with a particular topology (two-dimensional lattice) and knowledge about global properties, such as distance between any two nodes. On the other hand, a large number of dynamic, real networks, ranging from the Internet to social and biological networks, exhibit some recurrent patterns: power-law and small-world (as surveyed in [3] and discussed in detail in [42] and [6]). Interesting questions are *“How does the overlay network topology influence the performance of the resource location mechanisms?”* and *“How can we build an overlay topology that improves resource location performance?”*.

An attempt to answer the first question is presented in [29]: the article finds that search and replication are more efficient in uniform random topologies. The practical relevance of this result is not clear though, since the uniform random graph seems to be an ideal model that is not found in natural, self-configuring or dynamic networks [6].

An attempt to answer the second question is presented in [23]: the analysis of file sharing patterns in a scientific collaboration shows small-world characteristics, which suggests ways to exploit this pattern for performance.

The membership protocol influences the overlay network, as it collects information about current participating peers: it thus enriches the local view of the system with, ideally, up-to-date, global information. There are different design objectives for a membership protocol. For example, in Gnutella, the rather aggressive membership protocol maintains the highly dynamic nodes connected at a significant communication cost [35]. Membership protocols based on epidemic communication mechanisms [18] are scalable with the number of participants and tolerant to inaccuracies inherent in asynchronous distributed systems [10].

Preprocessing (for more efficient resource discovery) can take many forms, such as rewiring the overlay to adapt to changes in usage characteristics or replicating resource information in multiple remote places for increased availability. However, it is not obvious that these ideas would work in a dynamic environment as the one we consider, where resources leave the pool, their characteristics change rapidly, or user behavior suddenly changes. An important question is thus *“What preprocessing strategies can be effective in a dynamic system?”*

A recent paper [13] shows that the optimum replication of an item for search in unstructured P2P networks is proportional to the square root of the popularity of that item. However, this result is considered in a static environment.

In a different context [33], that of replication for achieving availability in file-sharing P2P networks, we study the problem of replicating read-only information. The question we answer there is only partially relevant to this context: we infer the number of replicas one needs to create in order to achieve a certain level of availability in a highly dynamic system.

Local request processing is concerned with multiple issues. First, it deals with aggregated requests (requests for multiple resources). Second, it can encapsulate local policies (e.g., drop requests for inappropriate content). Third, it may alter the additional information associated to requests (like time-to-live) based on some local information. (Perhaps this last idea raises important security issues. In addition, it may lead to highly undeterministic behavior.)

Request propagation is perhaps one of the currently hot research topics in the P2P area. In some cases, request propagation rules are dictated by other components of the resource discovery mechanism, as it is the case of DHTs, where the overlay and the propagation rules are strongly related [34, 39, 43, 36]). However, in an unstructured network, there are many degrees of freedom in choosing the propagation rule. Random walks, while inexpensive in terms of implementation costs, may often be inappropriate because of their lack of guarantees. A question that, despite significant effort, remains still to be answered is *“What request propagation rules are suitable for the context of generic resource discovery in unstructured overlays?”*

## 4 Resource Discovery in an Emulated Grid

Our objective is to observe, quantify, and understand how to control the synergies emerging from the interaction of the four components in the context of resource discovery in flat, unstructured networks. We study these interactions via simulations.

This section presents our current status with regard to simulations: the modeling of the Grid from the resource discovery perspective is presented in Section 4.1. In Section 4.2 we describe our simulator. Section 4.3 presents the (simplifying) assumptions we made to obtain the preliminary results presented in Section 4.4. (At various stages of progress, these results were published in [21] and [22].)

### 4.1 Modeling the Grid Environment

There are four environment parameters that influence the performance and the design of a resource discovery mechanism:

1. *Resource information distribution and density* refers to the fairness of sharing: some organizations will share a large number of resources, others will share just a few (for example, home computers). Also, some resources will be rather common (e.g., PCs running Linux), while others will be rare or even unique (e.g., particular services or powerful supercomputers).
2. *Resource information dynamism*: in the context of computer sharing, some resource attributes are highly variable (e.g., CPU load or available bandwidth between two nodes), while others vary so slowly that they can be considered static (e.g., operating system version, number and type of CPUs in a computer, etc.).
3. *Requests distribution* refers to the pattern of users' requests for resources. For example, multiple studies [9] have shown that HTTP requests follow Zipf distributions with coefficients within some intervals.
4. *Peer participation*<sup>1</sup> varies over time, more significantly in P2P systems than in current Grids. Influenced by incentives, some nodes will activate in the network for longer intervals than others.

Failure rate in a large-scale system is inevitably high and hence necessary to model. However, two of the parameters above can easily take it into account. The effects of failures are visible at two levels: the resource level and the node level. When resources fail, the nodes that publish their descriptions may need to update their local information to reflect the change. The resource failure can hence be seen as yet another

---

<sup>1</sup>This differs from resource information dynamism in the following way: resource information dynamism affects the information maintained by nodes, while peer participation refers to nodes' joining and leaving the network.

variation of resource attributes, and treated as part of the resource information dynamism parameter. When nodes fail, not only that the resources they advertise may become unknown to others, but the nodes cease to participate in maintaining the overlay and processing remote requests. Failures of nodes can hence be incorporated in peer participation parameter as a "departure"; however, there are some distinctions:

- Failures are ungraceful (unannounced) departures.
- Failures may not be perceived the same by all peers: for example, in case of network partitioning, a node may seem failed to some peers and alive to some others.

In this study, failure rate is incorporated in other environment parameters.

## 4.2 An Emulated Grid for Resource Discovery: Implementation Details

Because our focus is on large scale simulations (tens of thousands of nodes), we built a specialized, emulated grid, to experiment with various design solutions for resource discovery. For convenience, we used a large Linux cluster to test these strategies in a controlled environment. With minimal modifications, our framework could be used in real deployments.

In our framework, nodes form an overlay network. Each node is implemented (in Python) as a process that communicates with other nodes via TCP. Each node maintains two types of information: a) information about a set of resources; and b) information about other nodes in the overlay network (including, but not restricted to membership information).

The large number of processes needed by our large scale simulation raises multiple problems, ranging from individual machine resource starvation to library limitations. For our preliminary experiments presented in Section 4.4 (in which we simulated up to 32,768 virtual nodes), we used 128 systems (256 processors) communicating over fast Ethernet of the Chiba City cluster of Argonne National Laboratory.

Given the large scale of our simulation, a challenging problem was node startup: we essentially needed to start 32,768 process across 128 nodes, all being controlled from a central node. The naive approach of using persistent `rsh` processes to start up remote processes was inadequate because of system limitations and long startup time. We implemented a startup manager to handle simulated node startup: this process, an MPI program written in C, starts up one node manager process per cluster node. Once each cluster node has its own 'manager', the manager processes communicate with a central source to obtain specific simulated node process startup information. For 128 compute node and 32,768 simulated node run, there are 128 manager processes each of which starts up 256 simulated node processes.

## 4.3 Preliminary Experimental Setup

In order to isolate some of the correlations between the many parameters of our study, we started with a simplified, basic framework. Firstly, we considered an optimistic model of the Grid, with static resource attributes, constant peer participation and no failures, thus modeling only the resource and request distributions. Secondly, we considered a trivial resource discovery mechanism with the following design:

1. We considered a rather passive membership protocol. We consider a simple join mechanism as is common in peer-to-peer systems: a node joins the grid by contacting a member node. Contact addresses of member nodes are learned through out-of-band information. A node contacted by joining members responds with its membership information. Membership information is passively enriched over time: upon the receipt of a message from a previously unknown node, a node adds the new address to its membership list.
2. The overlay function accepts unlimited number of neighbors. In our experiments, we allow the overlay connectivity grow as the membership information grows. This way, we "neutralize" one more component, aiming to understand the correlations between graph topology and discovery performance. The starting overlay was generated by a hierarchy-based <sup>2</sup> Internet graph generator (Tiers [15]).

---

<sup>2</sup>This starting topology is not the most realistic choice: [7] shows that in the presence of an increasing number of peers and preferential attachment, the topology tends to be quite different, namely a power-law graph. It is not clear to us yet whether this makes a significant difference in the measurements.

3. We assume no preprocessing.
4. The request processing mechanism is also basic: we assume only simple requests and satisfiable only by perfect matches. We hence reduced the local processing to a minimum: a node that has a matching resource responds to the requester, otherwise decrements the request TTL and forwards it (if  $TTL > 0$ ) to some other node.

The request propagation strategy decides to which node(s) (among the locally known ones) a request is to be forwarded. In addition to contact addresses, nodes can store other information about their peers, such as information about requests previously answered. The tradeoff between the amount of information about neighbors and search performance generates a large set of alternatives, from random forwarding (no information about the resources provided by other nodes) to one-hop forwarding, when nodes know exactly which node has the requested resource.

We evaluated four request propagation strategies :

- (a) Random walks: the node to which a request is forwarded is chosen randomly. No extra information is stored on nodes.
- (b) Learning-based: nodes learn from experience by recording the requests answered by other nodes. A request is forwarded to the peer that answered similar requests previously. If no relevant experience exists, the request is forwarded to a randomly chosen node.
- (c) Best-neighbor rule records the number of answers received from each peer (without recording the type of request answered). A request is forwarded to the peer who answered the largest number of requests.
- (d) Learning-based + best-neighbor: identical with the learning-based strategy, except that, when no relevant experience exists, the request is forwarded to the best neighbor.

The remaining of this section presents how we modeled the environment for these simulations.

#### 4.3.1 Resource Distributions

In order to further reduce the parameter space of our study, we “select” a subset of resources and focus only on the requests that match resources in this subset. This approach allows us to obtain results that are valid for any number of shared resources in the system. Hence, there may be nodes that have no resources from the subset of interest.

In the following, the discussion is only relative to the subset of selected resources. For a realistic setup, we notice that, when the number of nodes increases:

1. The total number of resources increases. We consider that the average number of resources per node remains constant with the increase in the network size. In the experiments, we (arbitrarily) chose this constant equal to 5. The total number of resources in a network of  $N$  nodes is:  $D + 5 \times N$ ; in these experiments, we considered  $D = 100$ .
2. Under the above assumptions, we obtain a significant increase in resource frequency: the number of resources that can match a particular request increases as the number of nodes. We believe this is an unrealistic scenario, for often new nodes bring new types of resources (like unique on-line instruments, new data, and new, possibly locally developed, software applications). Therefore, in our testbed, we allowed the set of resource types increase slowly with the number of nodes in the system. The (again arbitrary) increase in the number of new resource types chosen for this experiments was 5% per node.

In this context, we experimented with two resource distributions of different degrees of fairness, as presented in Figure 2: a fair distribution, with all nodes providing the same number of resources, and a highly unbalanced one, generated as a geometric distribution, in which most resources are provided by a small number of nodes. Details on how many distinct resource types and how many identical resources were modeled for each network size are presented in Figure 7.



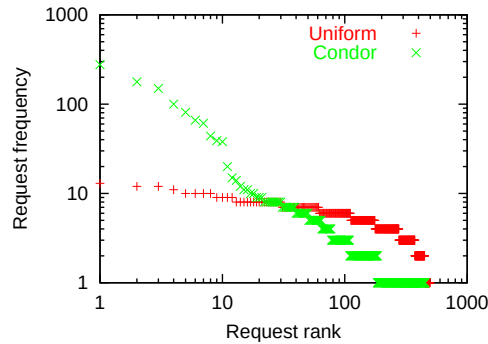


Figure 1: Distribution of user requests

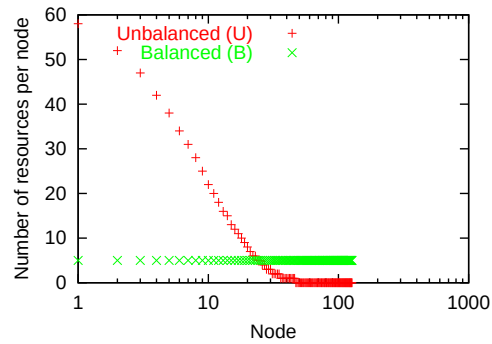


Figure 2: Distribution of resources on nodes: balanced (all nodes have equal number of resources) and highly unbalanced (a significant part of nodes have no resources from the set considered).

#### 4.3.2 User Requests

While usage patterns can be decisive in making design decisions, we tackle the problem of not having real user request logs, a problem inherent in systems during the design phase. We logged, processed, and used a one week's requests for computers submitted to the Condor pool [28] at University of Wisconsin. The Condor pool at University of Wisconsin consists mainly of Linux workstations, being hence a rather homogeneous set of resources; however, the pool is intensively used for various types of computations, and hence the requests specify various attribute values for, for example, minimum amount of available memory or required disk space. We processed these requests to capture their variety. Despite their authenticity, these traces may not accurately represent the request patterns in a sharing environment that usually comprises data and services in addition to computers: for example, the distribution of file requests in the D0 experiment during January–June 2002 looks different (Figure 6).

We also experimented with a synthetic user request distribution modeled as a uniform distribution. Figure 1 highlights the differences between the two request distributions: the Condor traces exhibit a Zipf distribution where a small number of distinct requests appear frequently in the set of 2000 requests considered; in the case of the uniform distribution, requests are repeated about the same number of times.

We evaluated the above mentioned request propagation strategies in a set of overlay networks of sizes from  $2^7$  to  $2^{15}$  nodes. In each of our experiments we randomly chose a fixed percentage of the nodes to which we sent independently generated sets of 200 requests. The same sets of requests, sent to the same nodes, respectively, are repeated to compare various request-forwarding algorithms.

### 4.4 Preliminary Experimental Results

The objectives of this preliminary study are threefold:

1. Estimate quantitatively the cost of simple resource discovery techniques based only on request-forwarding (no preprocessing).
2. Understand the effects of resource and request distributions on resource discovery performance.
3. Understand the correlation between performance and various design decisions in a realistic Grid environment.

This section discusses our current results on the first two problems. We are in the process of collecting more data to answer the third question.

#### 4.4.1 Quantitative Estimation of Resource Location Costs

A first question to answer in our study is: what are the search costs in an unstructured, static network in the absence of preprocessing? The answer is presented in Figures 3, 4 and 5: the learning-based strategies is the best performing under various sharing characteristics, with under 200 hops response time per request for the largest network in our experiment. For a network of thousands of nodes (hence, possibly thousands of participants, institutions and individuals) the average response time is around 20 hops. Assuming 20 mseconds to travel between consecutive nodes on a path (10 msec. latency in a metropolitan area network and 10 msec. necessary for request processing), then a path of 20 hops takes less than half a second.

Of course, this is the best performing algorithm of the set we tested: it takes advantage of similarity in requests and uses a possibly large storage of cached requests. It starts up with low performance until it builds its cache. We emphasis that the results we are presenting are not to advocate for one strategy, but to give a numerical estimate of the costs (in response time) involved.

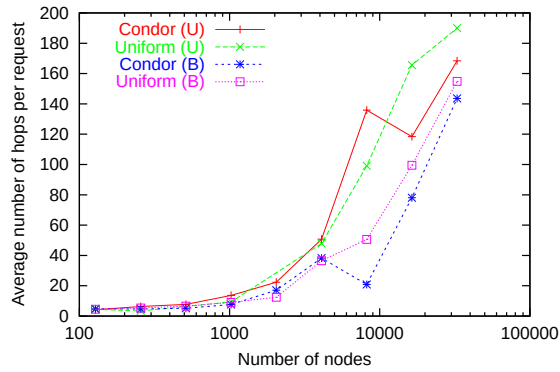


Figure 3: Performance (in average number of hops) of learning-based forwarding strategy for the three request distributions, in two environments with different resource sharing characteristics.

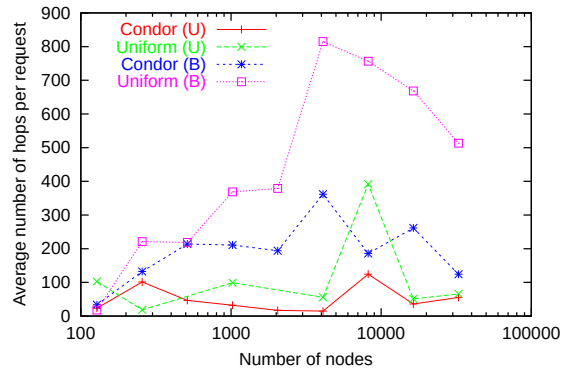


Figure 4: Performance (in average number of hops) of the best neighbor request forwarding strategy under different user request and sharing characteristics.

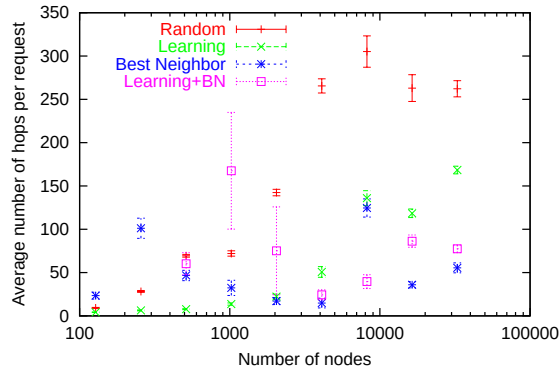


Figure 5: Performance off all four request forwarding strategies for a Condor request load in the unbalanced resource sharing environments.

The random forwarding algorithm has the advantage that no additional storage space is required on nodes to record history, but it is also expected to be the least efficient, intuition confirmed by the results shown in Figure 5 (Condor-based user requests, unbalanced resource distribution). For all network sizes in our experiments, the learning-based algorithm performs constantly well, while its more expensive version

(learning-based + best neighbor) proves to be rather unpredictable in terms of performance (see, for example, the large standard error deviation for 1024 and 2048 simulated nodes).

#### 4.4.2 Effects of the Environment

Figure 3 highlights the influence of user requests on the performance of the learning-based request forwarding strategy. (Of the strategies we considered, this is the most sensitive to user request patterns). The slightly better performance in the fair sharing environment is due to the random component of this strategy, employed when no relevant previous information on a specific request exists: random forward has a better chance to reach a useful node when information is distributed fairly on nodes. The learning-based strategy takes most advantage of the Zipf request distribution, since a significant part of requests are repeated (and hence can benefit from previous experience).

The best neighbor forwarding strategy is influenced more strongly by sharing patterns: compared to a balanced environment, in a highly unbalanced environment a node that had already answered a request is more likely to have answers to other requests as well. Figure 4 shows the response latency in unbalanced and balanced environments for the two request patterns we considered: the response latency almost doubles in the balanced sharing environment as compared to the unbalanced one. Note that the performance of the best neighbor strategy is influenced by past requests: the algorithm records the number of requests answered regardless of their type, hence it does not distinguish between nodes that answered same request  $n$  times and nodes that proved to have at least  $n$  distinct resources. This is the explanation for the algorithm performing better under a uniform user distribution load than under the Condor traces: the number of distinct requests in a uniform distribution is larger, hence the best neighbor as identified by this strategy has indeed a larger number of distinct resources.

## 5 Related Solutions to Related Problems

Two classes of related work are relevant for our study: resource discovery in dynamic, self-organizing networks; and resource discovery in wide-area systems. The former category has benefited from huge attention recently due to the popularity of peer-to-peer (P2P) file-sharing systems. Such systems identify resources (files) through their names and use a variety of strategies to locate a specified file, including aggressive flooding (Gnutella [12]), centralized index servers (Napster), combination of informed request forwarding and automatic file replication (Freenet [11]), and intelligent positioning of data into search-optimized, reliable, and flexible structures for efficient and scalable name-based retrieval (as in CAN [34], Chord [39], Tapestry [43] and Pastry [36]).

The basic mechanism in Gnutella [12] is flooding: it manages a highly dynamic set of members (with median lifetime per node per session about 60 minutes [37]) using periodic ping/pong messages. Requests for resources are propagated the same way: they flood the network until their time to live expires. The answers come back following the same trajectory, from node to node, to the node that initiated the request. Its relatively good search performance (as measured in number of hops) is achieved through intensive network usage [35].

Napster uses a centralized approach: a file index is maintained at a central location, while real data (files) are widely distributed. To provide files to the Napster community, a node registers the filenames and their location with the central index. A user queries the central index for a specific file and retrieves a set of locations that the user can later access to download the file.

CAN [34], Chord [39], Tapestry [43], and Pastry [36] build search-efficient indexing structures that provide good scalability and search performance at an increased cost for file and node insertion and removal. An implicit assumption in these systems is nodes' homogeneity: all nodes are expected to have the same capabilities.

Freenet [11], another file sharing system, includes both replica management and replica location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear. Freenet file location mechanism is based on usage patterns, using dynamic routing tables. However, the Freenet approach assumes that non-popular data is unimportant data, which is not always a valid assumption.

Domain Name Service [30] is perhaps the largest such system that provides name-based location information. Its performance is explained by the intense usage of cached static information.

All the solutions above are concerned with locating resources that inherently can be "named". Solutions that create an "artificial" name have been proposed for cases where resources cannot be named easily.

The Ninja project uses such a solution for locating services [20, 19]: a service is identified through a set of attributes. Lossy aggregations (using Bloom filters [8]) of local services are disseminated up a hierarchy. Requests for services are then guided by these service summaries up or down the hierarchy, in a B-tree search-like fashion.

[26] proposes a resource discovery mechanism based on request propagation: nodes (called traders) forward unsolved requests to other nodes. Like in Gnutella, the interconnection topology is a flat, dynamic network. Unlike in Gnutella, nodes choose their collaborators based on expertise and preference: traders explore the network on-demand, while serving a request, and off-demand, whenever necessary. Changes of trader's state (e.g., services provided or collaboration policies) are propagated through flooding throughout the network.

The Globe project [41] proposes a naming service [5] that transforms an URL into a location independent unique identifier. This way, the location service deals well with object migration.

Among the distributed resource sharing systems that do not use global names for resource discovery is Condor's Matchmaker [32]: resource descriptions and requests are sent to a central authority that performs the matching. The centralized architecture is efficient for the local area network for which Condor was initially designed, but it assumes the willingness of an organization to operate the central server.

Another relevant experience is provided by Globus's MDS [14]: initially centralized, this service moved to a decentralized structure as its pool of resources and users grew. In MDS-2, a Grid consists of multiple information sources that can register with index servers via a registration protocol. Index servers, or users, can use an enquiry protocol to query directory servers to discover entities and to obtain more detailed descriptions of resources from their information sources. Left unspecified is the techniques used to associate entities into directories and to construct an efficient, scalable network of directory servers. Our research is complementary to MDS, proposing and evaluating mechanisms that can be used to organize these directories (the equivalent of what we call *nodes* or *peers* in this paper) in flat, dynamic networks.

## 6 Summary

We speculate that, even if they will continue to serve different communities, the characteristics and the design objectives of Grid and P2P environments will converge: Grids will increase in scale and inherently will have to deal with more dynamism than today, P2P systems will start to provide more complex functionalities, integrating data and computation sharing with various security requirements. We study the resource discovery problem in a resource sharing environment that combines the characteristics of the two environments: the complexity of the Grids (that share a large diversity of resources: data, applications, computers, online instruments, storage) with the scale, dynamism and heterogeneity of today's P2P systems. We propose 4 components that, we claim, can define any decentralized resource discovery design: membership protocol, overlay function, preprocessing, and request processing.

We describe our emulator for evaluating resource discovery techniques. In order to isolate the four components, we evaluated a simple resource discovery mechanism based on request propagation. Our results give a quantitative measure of the influence of the sharing environment (i.e., fairness of sharing) on resource discovery.

## References

- [1] ADAMIC, L., HUBERMAN, B., LUKOSE, R., AND PUNIYANI, A. Search in power law networks. *Physical Review. E* 64 (2001), 46135–46143.
- [2] ADAR, E., AND HUBERMAN, B. A. Free riding on gnutella. *First Monday* 5, 10 (2000).

- [3] ALBERT, R., AND BARABÁSI, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74 (January 2002), 47–97.
- [4] ALBERT, R., JEONG, H., AND BARABÁSI, A.-L. Diameter of the world wide web. *Nature* 401 (1999), 130–131.
- [5] BALLINTIJN, G., VAN STEEN, M. V., AND TANENBAUM, A. Scalable naming in global middleware. In *13th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-2000)* (2000), pp. 624–631.
- [6] BARABÁSI, A.-L. *Linked: The New Science of Networks*. Perseus Publishing, 2002.
- [7] BARABÁSI, A.-L., AND ALBERT, R. Emergence of scaling in random networks. *Science* 286 (1999), 509–512.
- [8] BLOOM, B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13, 7 (1970), 422–426.
- [9] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., AND SHENKER, S. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of IEEE InfoCom Conference* (New York, NY, 1999), IEEE Press.
- [10] CHANDRA, T. D., HADZILACOS, V., TOUEG, S., AND CHARRON-BOST, B. On the impossibility of group membership. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)* (New York, USA, 1996), pp. 322–330.
- [11] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability* (Berkeley, California, 2000).
- [12] CLIP2. The gnutella protocol specifications v0.4, <http://www.clip2.com>.
- [13] COHEN, E., AND SHENKER, S. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the ACM SIGCOMM Conference* (2002).
- [14] CZAJKOWSKI, K., FITZGERALD, S., FOSTER, I., AND KESSELMAN, C. Grid information services for distributed resource sharing. In *10th IEEE Symposium on High Performance Distributed Computing* (2001).
- [15] DOAR, M. A better model for generating test networks. In *IEEE Global Internet* (1996), pp. 86–93.
- [16] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
- [17] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal on Supercomputing Applications* (2001).
- [18] GOLDING, R. A., AND TAYLOR, K. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz, Jack Baskin School of Engineering, Mar. 1992.
- [19] GRIBBLE, S. D., BREWER, E. A., HELLERSTEIN, J. M., AND CULLER, D. Scalable, distributed data structures for internet service construction. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)* (San Diego, CA, 2000).
- [20] GRIBBLE, S. D., WELSH, M., BEHREN, R. V., BREWER, E. A., CULLER, D., BORISOV, N., CZERWINSKI, S., GUMMADI, R., HILL, J., JOSEPH, A. D., KATZ, R. H., MAO, Z., ROSS, S., AND ZHAO, B. The ninja architecture for robust internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing* (2001).
- [21] IAMNITCHI, A., AND FOSTER, I. On fully decentralized resource discovery in grid environments. In *International Workshop on Grid Computing* (Denver, Colorado, November 2001), IEEE.

- [22] IAMNITCHI, A., FOSTER, I., AND NURMI, D. A peer-to-peer approach to resource discovery in grid environments. In *High Performance Distributed Computing* (Edinburgh, UK, July 2002), IEEE.
- [23] IAMNITCHI, A., RIPEANU, M., AND FOSTER, I. Locating data in (small-world?) peer-to-peer scientific collaborations. In *1st International Workshop on Peer-to-Peer Systems* (2002), LNCS Hot Topics series, Springer-Verlag.
- [24] KLEINBERG, J. The small-worlds phenomenon: an algorithmic perspective. Tech. rep., Cornell University, 1999.
- [25] KUTTEN, S., AND PELEG, D. Deterministic distributed resource discovery. In *Nineteenth Annual ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing* (Portland, Oregon, 2000).
- [26] LEE, O., AND BENFORD, S. An explorative approach to federated trading. *Computer Communications* 21(2) (1998).
- [27] <http://www.limewire.com>.
- [28] LITZKOW, M. J., LIVNY, M., AND MUTKA, M. W. Condor – A hunter of idle workstations. In *Proc. 8th Intl. Conf. on Distributed Computing Systems* (San Jose, Calif., June 1988), pp. 104–111.
- [29] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th annual ACM International Conference on supercomputing (ICS)* (2002).
- [30] MOCKAPETRIS, P. Domain names—concepts and facilities. In *RFC 1034* (1987).
- [31] PELEG, D. *Distributed computing*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. A locality-sensitive approach.
- [32] RAMAN, R., LIVNY, M., AND SOLOMON, M. Matchmaking: Distributed resource management for high throughput computing. In *7th IEEE International Symposium on High Performance Distributed Computing* (1998).
- [33] RANGANATHAN, K., IAMNITCHI, A., AND FOSTER, I. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop* (May 2002).
- [34] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *SIGCOMM* (San Diego USA, 2001).
- [35] RIPEANU, M., IAMNITCHI, A., AND FOSTER, I. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal* 6, 1 (2002), 50–57.
- [36] ROWSTRON, A. I. T., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware* (2001), pp. 329–350.
- [37] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. A measurement study of peer-to-peer file sharing systems. Tech. Rep. UW-CSE-01-06-02, University of Washington, 2001.
- [38] *SETI@home*. <http://setiathome.ssl.berkeley.edu>.
- [39] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM 2001* (San Diego, USA, 2001).
- [40] VAN STEEN, M., HAUCK, F., HOMBURG, P., AND TANENBAUM, A. Locating objects in wide-area systems. *IEEE Communications Magazine* (1998), 104–109.

- [41] VAN STEEN, M., HOMBURG, P., AND TANENBAUM, A. Globe: A wide-area distributed system. *IEEE Concurrency* (1999), 70–78.
- [42] WATTS, D. J. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, 1999.
- [43] ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. CSD-01-1141, Berkeley, 2001.

## APPENDIX

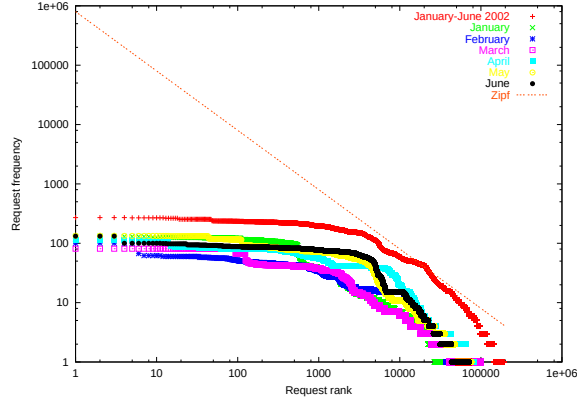


Figure 6: File requests in the D0 experiment during January-June 2002 period.

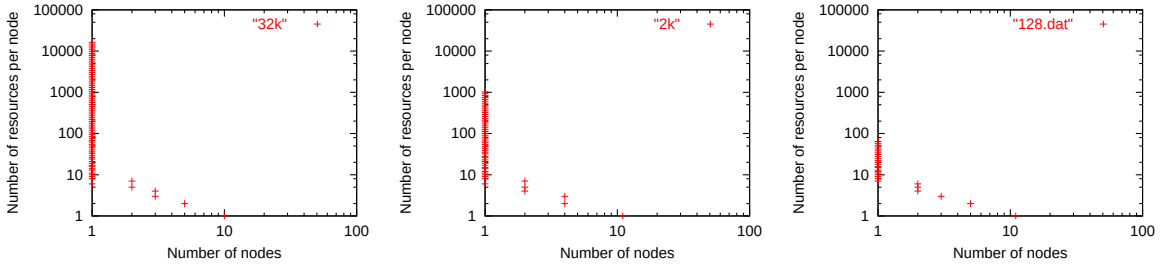


Figure 7: The highly unbalanced distribution of resources on nodes for three networks of sizes 32,768 (left), 2048 (center) and 128 (right) nodes. **Left:** 1738 distinct types and a total of 163833 resources; number of identical resources varies from 67 to 139 (mean = 94). **Center:** 202 distinct types and a total of 10238 resources; number of identical resources varies from 34 to 70 (mean= 50). **Right:** 106 distinct types and a total of 640 resources; the number of identical resources varies from 2 to 13 (mean = 6).