On Fully Decentralized Resource Discovery in Grid Environments

Adriana Iamnitchi [1] and Ian Foster [1,2]

[1] Department of Computer Science, The University of Chicago
1100 E. 58th Street, Chicago, IL 60637, USA
{anda, foster}@cs.uchicago.edu
[2] Mathematics and Computer Science Division, Argonne National Laboratory
Argonne, IL 60439, USA

Abstract. Computational grids provide mechanisms for sharing and accessing large and heterogeneous collections of remote resources such as computers, online instruments, storage space, data, and applications. Resources are identified based on a set of desired attributes. Resource attributes have various degrees of dynamism, from mostly static attributes, like operating system version, to highly dynamic ones, like network bandwidth or CPU load.
In this paper we propose a peer-to-peer architecture for resource discovery in a large and dynamic collection of resources. We evaluate a set of request-forwarding algorithms in a fully decentralized architecture, designed to accommodate heterogeneity (in both sharing policies and resource types) and dynamism. For this, we build a testbed that models two usage characteristics: (1) resource distribution on peers, that varies in the number and the frequency of shared resources; and (2) various requests patterns for resources. We analyzed our resource discovery mechanisms on up to 5000 peers, where each peer provides information about at least one resource. We learned that a decentralized approach is not only desirable from administrative reasons, but it is also supported by promising performance results. Our results also allow us to characterize the correlation between resource discovery performance and sharing characteristics.

1 Introduction

Opportunistic sharing of Internet connected computers is a low cost method for achieving computational power. Recently, this trend was supported by research in the domain of computational grids [5]: collections of shared, geographically distributed hardware and software resources made available to groups of remote users.

In computational grids 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, accessing information about components, 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)[6].

A basic service in grids 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 in a grid is made challenging by 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 for VOs to evolve over time, with, for example, institutions joining and leaving (along with their resources and users), the number of resources shared by an institution varying, and resource characteristics such as availability and CPU load changing.

These characteristics create significant difficulties for traditional centralized and hierarchical resource discovery services. Hence, we investigate a flat, fully decentralized architecture, developing a candidate architecture design and studying the characteristics of this design via detailed simulation studies. Our architecture is distinguished by its peer-to-peer flavor: entities that participate in resource discovery are equally important for the system's correct and efficient functioning.

There are many common characteristics of grid environments and current peer-to-peer systems: dynamism, wide-area scale, and heterogeneity are perhaps the most significant. Recently, decentralized file-sharing systems, such as Gnutella and Freenet, have been extensively analyzed [2, 10], but these results are of little relevance for us because of several important differences with our problem:

1. Anonymity is a major objective of file-sharing systems like Gnutella and Freenet and, consequently, design decisions were made to achieve it. In our environment, anonymity is not only unnecessary but may also be undesirable, for example, for accountability, performance tuning, or replication decisions.
2. The dynamics of the Gnutella network are perhaps different from the dynamics of a Virtual Organization.
3. Resource sharing in a grid will often be based on different policies, such as community contributions and/or payment, that will hopefully avoid phenomena such as Gnutella's free riding behavior [1].
4. User request patterns may be different. While in Gnutella a user is unlikely to request the same file many times, in grids it is quite usual that a user will use the same kind of resource multiple times.
5. In file-sharing systems, the answer has to perfectly match the request. In the case of resource discovery, matches may be approximate (e.g., a 800 MHz processor is likely to be acceptable if we are asking for 500 MHz processor).
6. Properties used to refer to resources may be mutable.

Hence, we cannot use performance analysis of the large-scale Gnutella network [10] for estimating the performance of a similar solution for our problem. It is our objective in this paper to also present a framework for our architecture evaluation and a preliminary set of performance results. Our results show that a flat, decentralized, self-configuring architecture is a promising solution for dealing with large, variable, and heterogeneous collections of resources. These results are even more significant since we did not put effort into improving performance.

The rest of this paper is as follows. We discuss related work in Section 2. Section 3 presents our design decisions. We describe in Section 4 the emulated grid that we have built as a testbed for evaluating various design alternatives and we present our measurements in Section 5. We conclude in Section 6 with lessons learned and future research plans.

2 Locating Resources in Wide-Area Systems

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 named file, including aggressive flooding (Gnutella [15]), combination of informed request forwarding and automatic file replication (Freenet [2]), and intelligent positioning of data into search-optimized, reliable, and flexible structures for efficient and scalable retrieval (as in CAN [9], Chord [12], and Tapestry [14]).

The most successful wide-area service for locating resources based on names is DNS. Its hierarchical organization and caching strategies take advantage of the rather static information managed.

All of the above-mentioned systems use names as their search criteria. However, in our context, requests specify sets of desired attributes and values: for example, the name and version of the operating system and the CPU load. From this perspective, Web search engines are resource discovery mechanisms more similar to what we need for grids: given a set of criteria (search keys), they return the addresses of relevant resources (web pages). However, Web search engines do not deal well with dynamic information.

Nonetheless, using unique names as global identifiers is an appealing idea: one can position information, based on global IDs, into search-optimized structures (as in search trees or, more scalably and reliably, Plaxton networks [7]) for faster search. Globe [13] is one system that assigns location-independent names to resources as a means for retrieving mobile resources (e.g., mobile services) in wide-area systems. However, while we could imagine defining a mapping from attribute-value pairs to names (e.g., by assigning a distinct name to each meaningful combination of attribute-value pairs) and then maintaining a mapping between names and the physical resources that have those attributes, the volatility of attribute values would make the utility of this mapping uncertain. Moreover, unlike in name-based search systems, matching resource descriptions and requests can benefit from a certain degree of approximation: for example, a computer with a CPU load of 10% is most likely appropriate for matching a request for a computer with a CPU load of at most 20%.

While aware of the potential existence of clever methods for organizing grid resource information in search-efficient structures, we believe that using name-based search is not feasible for resource discovery grids. Among the distributed resource sharing systems that do not use global names for resource discovery is Condor's Matchmaker [8]: 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 [3]: initially centralized, this service moved to a decentralized structure as its pool of resources and users grew. In MDS-2, a Grid is assumed to consist 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.

3 Resource Discovery Problem Restated: a Peer-to-Peer Approach

A grid is a collection of resources shared by different organizations or/and individuals. Many organizations will have strictly specified sharing policies, like time intervals when their resources are available for non-internal users or the types of projects to which their resources may contribute. Attempting to enforce uniform rules over the grid would drastically limit participation. Moreover, the pool of resources shared by an organization may vary over time, subject to local computing load and sharing policies. Therefore, a natural solution is to allow every organization to control access to information about its local, shared resources.

We assume that every participant in the VO (organization or individual) has one or more servers that store and provide access to local resource information. We call these servers *nodes* or *peers*. A node may provide information about one resource (e.g., itself) or multiple resources (e.g., all resources shared by an organization).

From the perspective of resource discovery, the grid is 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.

These observations and assumptions are direct consequences of the problem characteristics. We present our design decisions in the following.

3.1 Framework

The basic framework is as follows. We assume users send their requests to some known (typically local) node. The node responds with the matching resource descriptions if it has them locally, otherwise it forwards the requests to another node. Intermediate nodes forward a request until its time-to-live (TTL) expires or matching resources are found, whichever occurs first. If a node has information matching a forwarded request, it sends the information directly to the node that initiated the forwarding (rather than via intermediate nodes), which in turn will send it to its user.

Within this framework, a particular resource discovery algorithm is defined by two mechanisms: the membership protocol that provides each node with (typically partial) membership information about other nodes and a request-forwarding strategy used to determine to which nodes requests should be forwarded. We focus in this paper on the latter, describing and evaluating four different request-forwarding strategies, described below. We are not particularly concerned here with the characteristics of the membership protocol, but note simply that we use a soft-state membership protocol as is common in peer-to-peer systems. A node joins the grid by contacting a member node. Contact addresses of member nodes can be learned through out-of-band information. A node contacted by joining members responds with its membership information. Membership lists are updated by periodic "I'm alive" messages exchanged between *neighbors*—nodes that know each other. Membership information can also be enriched over time: upon the receipt of a message from a previously unknown node, a node adds the new address to its membership list.

The request-forwarding strategy decides to which node (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. Because information is dynamic, nodes do not cache the information itself (i.e., the attribute values), but the address(es) where relevant information was previously found.

3.2 Request Forwarding

Nodes forward the requests they cannot answer to a peer selected from the locally known nodes. We evaluated four request-forwarding algorithms:

FRQ1. Random: chooses randomly the node to which a request is forwarded. No extra information is stored on nodes.

FRQ2. Experience-based + random: 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.

FRQ3. Best-neighbor algorithm 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.

FRQ4. Experience-based + best-neighbor: identical with FRQ2, except that, when no relevant experience exists, the request is forwarded to the best neighbor.

The following section presents the emulated grid that serves as a testbed for the evaluation of these four request-forwarding algorithms in different grids.

4 An Emulated Grid for Resource Discovery

We opted for building and using an emulated grid to understand whether a fully decentralized, flat design approach for resource discovery is appropriate in terms of response time, response quality, and scalability. Because our focus is on large scale simulations (thousands of nodes), we preferred to build an emulated grid instead of using a general purpose, usually unscalable, discrete event simulator. Unlike other grid simulators [11], we designed this framework with the sole purpose of analyzing resource discovery strategies.

In our framework, each node is implemented as a process that communicates with other nodes via TCP. Each node maintains two types of information: a) information about a number of resources (its contribution to the VO) and b) information about other nodes in the system (including, but not restricted to membership information). The amount of resource information hosted by nodes varies: some have information about a large number of resources; others, only one resource. In our preliminary experiments, we assume the amount of information about other nodes that can be stored locally is unlimited.

The performance of the resource discovery mechanism depends on usage and environment characteristics like scale, resource distributions, and user request patterns. We evaluated our request-forwarding strategies for a number of nodes from 1000 to 5000. We studied our architecture using requests matching a set of 10000 distinct resources, some with identical descriptions. Request similarity is difficult to quantify and model. We simplified this problem by considering only simple requests (requests for one resource) and perfect matching. While this setup is not realistic, it helps us understand the effects of each of the parameters we consider. We shall extend our experiments with more realistic scenarios when analyzing the requirements of the query language.

In the remaining of this section we give details on how we model resource distributions on nodes and user requests.

4.1 Resource Distributions

When analyzing resource distributions, we have to consider two issues. The first is the distribution of resources on nodes: some nodes share a large number of resources, others just one. We modeled three resource distributions, of different degrees of fairness, as presented in Figure 1. Because of space limitations, we present in this paper only the results measured in environments with unbalanced and balanced resource distributions.

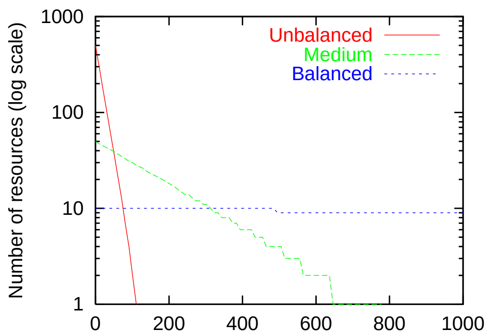

Fig. 1. Geometric distribution of resources on 1000 nodes varies from highly unbalanced (few nodes provide a large number of resources, while most nodes provide only one) to fair (all nodes share the same number of resources).

The second aspect we need to model is resource frequency: we need to distinguish between common resources (ones that are numerous and widely available) and rare (even unique) resources. Assuming R resources of D distinct types, we consider all resources in a grid are equally common (with the same frequency R/D) and we compare resource discovery performance in grids with different values for R/D. This is a simpler model than permitting the frequency to vary over individual resources and tracking requests for common and rare resources.

4.2 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 experimented with two request distributions, random and geometric, in which requests were chosen to match existing resources.

In our experiments, the number of distinct requests in the random distribution is approximately twice as large as that in an equally-sized geometric distribution (and hence, on average, the same request is repeated twice as much in the geometric distribution than in the random distribution).

In each of our experiments we randomly chose a set of 10 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.

4.3 Starting Topology

The connection graph defined by membership information strongly influences the resource discovery performance: the more nodes known to the local node, the more informed its request-forwarding decision can be. We described the membership protocol previously but we did not detail the starting topology: the graph whose vertices are nodes in the grid and whose edges connect pairs of nodes that know each other, as it looks before requests are processed.

We generated the starting topology using the Tiers network generator[4]. In the experiments presented here we consider that all nodes had joined the grid before the first request was generated. We assume no failures. The connection graph changes over time, influenced by design and usage factors (location of requested resources in graph, request-forwarding algorithm, number and diversity of user requests), therefore the starting topology is important mostly to avoid unrealistically optimistic starting configurations, e.g., a star topology.

5 Experimental Results

We used the testbed presented above to evaluate the performance of four request-forwarding algorithms in grids with different resource distributions. We are interested in understanding response time per request and success rate. Currently we measure response time as the number of hops traversed for answering a request. Because in our experiments requests only refer to existing resources, success rate less than 1 is due to dropped requests because of dead ends or exceeded TTL.

We present in this section the evaluation of the four request forwarding algorithms under different sharing and usage conditions. First, we consider an environment with common resources (100 resources of each type) and measure the average number of hops per request for the four forwarding algorithms. Second, we compare these results with the same experiments in an environment with less common resources (10 of each type). In these two sets of results, user requests are generated to follow a geometric distribution. To understand the influence of user request pattern on resource discovery performance, we compare performance when requests follow a geometric and, respectively, a random distribution in environments with different levels of resource frequency.

Because of the randomness encapsulated in our testbed and in our request-forwarding algorithms, we repeated our experiments multiple times. The results shown in this section are the average values of measurements obtained in multiple runs.

Note that only the experience-based forwarding algorithms take advantage of the time and space-locality of user requests. Given our assumption of infinite storage space for logs, the response time for the experience-based algorithms is a lower bound for these particular algorithms. However, these optimistic results are, at least partially, counterbalanced by our measurements including the system's warm up (while nodes learn about other nodes' resources).

Figure 2 presents the average number of hops per request for various degrees of sharing fairness, when there are on average 100 resources of each type in the system. The random forwarding algorithm (labeled FRQ1 in Figure 2) 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. This intuition is confirmed by the unbalanced resource distribution (Figure 2 left), but is infirmed in the case of the most balanced distribution, as seen in Figure 2 right. In all distributions, the experience-based + random algorithm (labeled FRQ2) performs the best, while its more expensive version (FRQ4) proves to be equally or less efficient. Best neighbor algorithm (FRQ3), which is less expensive in terms of storage space (since it records only the number of requests answered by each node, not the requests themselves), performs well only in the unbalanced distribution; in all the other cases, because of the uniformly small number of distinct resources per node, it gives false hints on locating resources. Its influence is the reason for FRQ4 performing poorer than FRQ2, most clearly seen in the balanced distribution.

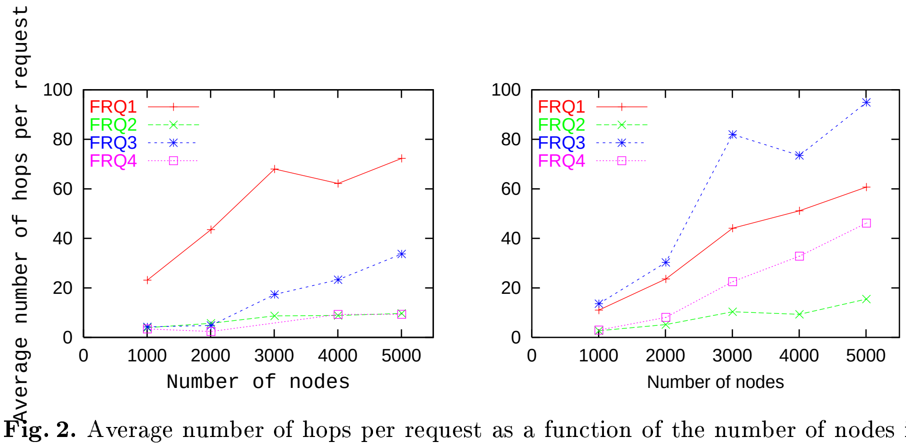

Fig. 2. Average number of hops per request as a function of the number of nodes in the grid, for two environments with different resource distributions. (Left: unbalanced. Right: balanced). The number of resources considered is 10000. Resource frequency (the number of resources of the same type) is constant:100.

Figure 3 compares resource discovery performance for grids with different resource frequencies: when there are on average 100 ($R/D = 100$) and respectively 10 ($R/D = 10$) resources of each type. For the grid with less common resources, the average number of hops is large (in the order of 100s), so we present the percentage of answered requests within a time-to-live of 100. Observe that in this environment performance decreases faster with the number of nodes; the lower 4 lines in each graph in Figure 3 represent the results measured in the environment with less common resources. Not surprisingly, the four forwarding algorithms perform comparatively almost the same, independent of resource frequency.

The influence of different user request patterns is presented in Figure 4. We measured average number of hops per request for the best performing forwarding algorithm (experience-based + random) in the two environments with different

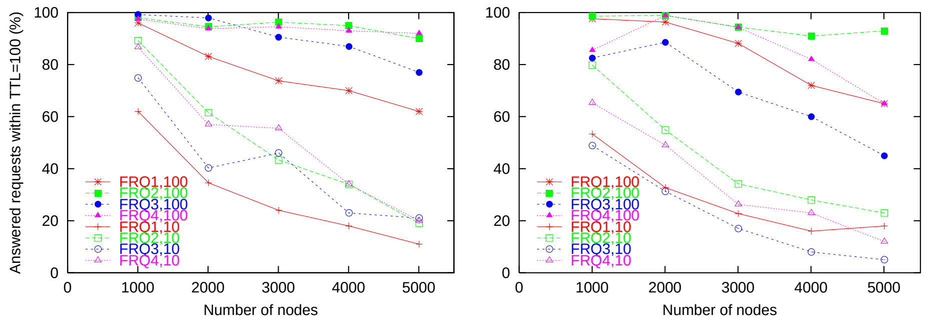

Fig. 3. Percentage of answered requests within time-to-live 100 in two environments. Left: unbalanced resource distribution. Right: balanced resource distribution. For each resource distribution we considered different resource frequencies: $R/D = 100$ (the upper 4 plots in each graph) and 10, respectively.

resource frequencies. Again, the influence of user request patterns is stronger in the environment with less common resources: the number of hops per request is more than 4 times larger when requests follow a random distribution, compared to at most 2 in the environment with common resources.

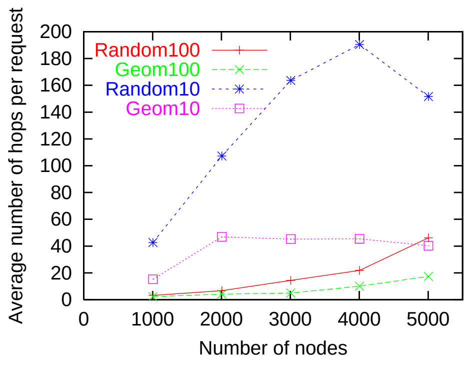

Fig. 4. The influence of different user requests patterns (random and geometric distributions) on resource discovery performance in environments with different resource frequencies (100 and 10, respectively) and unbalanced resource distribution. The request-forwarding algorithm used was experience-based+random (FRQ2).

6 Conclusions and Future Work

We argue in this paper that resource discovery in grid environments requires a decentralized, flat architecture. The inherent problems with such architectures are the tradeoffs between communication costs and performance. To better understand these tradeoffs and to evaluate different strategies for resource discovery, we built a simple grid emulator and modeled some relevant parameters: the pattern of user requests and the distribution of resource information in the grid. Results obtained for collections of up to 5000 nodes that host descriptions of a total of 10000 resources are presented. Our results suggest that a decentralized resource discovery strategy may be a feasible solution: if a request can be forwarded in 20 msec. (assuming 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.

It is interesting to observe the relation between discovery performance and environment characteristics: our experiments show that the best performing algorithm (and the most expensive, in terms of storage space) performs well independent of the type of resource distribution. The best-neighbor algorithm performs well for an unbalanced resource distribution. The least expensive forwarding algorithm—random—performs satisfactorily in all cases and is at its best when resources are equally distributed over nodes. The relative performance of the forwarding algorithms considered is independent of the average resource frequency.

Three of the four request-forwarding algorithms evaluated in this paper attempt to improve search performance by using past experience. What differentiates these algorithms is the type of information they choose to remember about the past. The algorithm that remembers the neighbor that helped the most in the past has a good guess only when few nodes have a lot to share while others share close to nothing. However, in an environment where all nodes contribute equally, this algorithm tends to perform worse than random.

The simple strategies evaluated, while promising, leave room for improvements. One limitation of our framework is the uneven spread of information: only nodes contacted by users learn. An obvious improvement is to also keep the other nodes informed, by exchanging history information (for example, along with membership information). Another simple improvement is to attach history information to request messages and hence to avoid visiting already visited nodes.

Our results suffer from limitations due to the computational expensive experiments. More accurate measurements require a larger set of (ideally larger) experiments: larger number of requests, larger number of runs per experiment, larger number of nodes. We plan to enrich our understanding of the problem through careful analytical evaluations.

The most interesting outcomes of this work are still to come. Future work includes three directions: 1) further development of the emulated grid to include capabilities for processing realistic requests and simulating data dynamism; 2) design and evaluation of different membership protocols and new request-forwarding strategies; and 3) reliability evaluation. Our final objective is to propose a coherent suite of resource discovery mechanisms and an expressive query language that are well suited to a large, heterogeneous community of grid members.

Acknowledgments

We are grateful to David Beazley for his generosity in sharing computational resources. This research was supported by the National Science Foundation under contract ITR-0086044.

References

1. Adar, E., and Huberman, B. Free riding on gnutella. *First Monday 5*, 10 (2000).
2. Clarke, I., Sandberg, O., Wiley, B., and Hong, T. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability* (2000).
3. 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).
4. Doar, M. A better model for generating test networks. In *IEEE Global Internet* (1996), pp. 86–93.
5. Foster, I., and Kesselman, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
6. Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal on Supercomputing Applications* (2001).
7. Plaxton, C. G., Rajaraman, R., and Richa, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures* (1997).
8. 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).
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content addressable network. In *ACM SIGCOMM* (2001).
10. Ripeanu, M. Peer-to-peer architecture case study: Gnutella network. In *International Conference on Peer-to-peer Computing* (2001).
11. Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., and Chien, A. A. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing* (2000).
12. Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM* (2001).
13. van Steen, M., Homburg, P., and Tanenbaum, A. Globe: Awide-area distributed system. *IEEE Concurrency* (1999), 70–78.
14. Zhao, B., Kubiatowicz, J., and Joseph, A. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Tech. Rep. UCB//CSD-01-1141, U. C. Berkeley, 2001.
15. Gnutella protocol specification. http://www.clip2.com/articles.html.