Chapter 1

A PEER-TO-PEER APPROACH TO RESOURCE LOCATION IN GRID ENVIRONMENTS

Adriana Iamnitchi[1] and Ian Foster[1,2]
[1]*Department of Computer Science, The University of Chicago*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract Resource location (or discovery) is a fundamental service for resource-sharing environments: given desired resource attributes, the service returns locations of matching resources. Designing such a service for a Grid environment of the scale and volatility of today's peer-to-peer systems is not trivial. We explore part of the design space through simulations on an emulated Grid. To this end, we propose four axes that define the resource location design space, model and implement an emulated Grid, evaluate a set of resource discovery mechanisms, and discuss results.

1. INTRODUCTION

When dealing with large sets of shared resources, a basic problem is locating resources in the absence of a naming scheme, that is, dealing with requests that specify a set of desired attributes ("Linux machine with more than 128 MB of available memory") rather than a globally unique identifier (such as "ficus.cs.uchicago.edu"). Attribute-based search is challenging when resource attributes can vary over time (e.g., CPU load, available bandwidth, even software versions) and when the number of resources is large and/or dynamic (as resources join, leave, or fail).

We study the resource discovery problem in a resource-sharing environment that combines the complexity of 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 emerged from different communities, and hence their current designs highlight different requirements. We believe that the design objectives of the two environments will eventually con-

verge. Consequently, it is important to analyze, compare, and contrast their current requirements and characteristics.

To this end, we discuss the resource location problem in the context of the two resource-sharing environments (Section 2). We identify four critical design objectives for resource discovery (Section 3), and we present a scheme for characterizing resource discovery mechanisms (Section 4) that defines the design space and provides the basis for comparing existing solutions. We then describe an emulated large-scale resource-sharing environment (Section 5), which we use for a preliminary performance evaluation of resource discovery techniques (Section 6). We conclude with a brief summary.

2. GRID AND PEER-TO-PEER ENVIRONMENTS

In recent years significant interest has focused on two resource-sharing environments: Grids and peer-to-peer (P2P) systems. The two systems have followed different evolutionary paths. Grids have incrementally scaled the deployment of relatively sophisticated services, connecting small numbers of sites into collaborations engaged in complex scientific applications. P2P communities, on the other hand, have developed rapidly around unsophisticated but popular services such as file sharing, focusing on scalability and support for intermittent user and resource participation. As a result of these different evolutionary paths, the two systems differ in three respects: target communities, resources, and applications. We discuss each of these below.

Despite these differences, however, we maintain that the two environments are in fact concerned with the same general problem, namely, resource sharing within *virtual organizations (VOs)* that link resources and people spanning multiple physical organizations. Moreover, the two environments seem likely to converge in terms of their concerns, as Grids scale and P2P systems address more sophisticated application requirements [FI03].

2.1 Target Communities and Incentives

The development and deployment of Grid technologies were motivated initially by the requirements of professional communities to access remote resources, federate datasets, and/or to pool computers for large-scale simulations and data analysis. Participants in contemporary Grids form part of established communities that are prepared to devote effort to the creation and operation of required infrastructure and within which exist some degree of trust, accountability, and opportunities for sanctions in response to inappropriate behavior.

In contrast, P2P has been popularized by grass-roots, mass culture (music) file-sharing and highly parallel computing applications [ACK+02, AK02] that scale in some instances to hundreds of thousands of nodes. The "communities" that underlie these applications comprise diverse and anonymous individuals with little incentive to act cooperatively and honestly. Thus, for example, we find that in file-sharing applications, there are few providers and many consumers [AH00]; the operators of SETI@home [SET] devote significant effort to detecting deliberately submitted incorrect results; and people tend to intentionally misreport their resources [SGG02].

The two target communities differ in scale, homogeneity, and the intrinsic degree of trust. The natural tendency of Grids to grow, however, will inevitably lead to less homogeneous communities and, consequently, to smaller degrees of trust. Participation patterns will change also with scale: intermittent participation is likely to become the norm. All these characteristics have a strong impact on defining the assumptions one can make (or, rather, cannot) about the environment. We need support for volatile user communities; and we need to deal with the lack of incentives for and interest in centralized, global administrative control.

2.2 Resources

In general, Grid systems integrate resources that are more powerful, more diverse, and better connected than the "desktop at the edge of the Internet" [Shi00] that constitutes a typical P2P resource. A Grid resource might be a cluster, storage system, database, or scientific instrument of considerable value that is administered in an organized fashion according to some well-defined policy. This explicit administration enhances the resource's ability to deliver desired qualities of service and can facilitate, for example, software upgrades, but it can also increase the cost of integrating the resource into a Grid. Explicit administration, higher cost of membership, and stronger community links within scientific VOs mean that resource availability tends to be high and uniform.

In contrast, P2P systems deal with intermittent participation and highly variable behavior. For example, one study of Mojo Nation [WO02] showed that a node remained connected on average for about 28% of time. Moreover, the connection time distribution was highly skewed, with one sixth of the nodes remaining always connected.

Large-scale Grids will borrow some of the characteristics of today's P2P systems in resource participation: unreliable resources and intermittent participation will constitute a significant share. At the same time, Grid resources will preserve or increase their diversity. Consequently, services - and resource discovery in particular - will have to tolerate failures and adapt to dynamic resource participation.

2.3 Applications

The range and scope of scientific Grid applications vary considerably. Three examples that show the variety of deployed Grid applications are the HotPage portal, providing remote access to supercomputer hardware and software [TMB00]; the numerical solution of the long-open `nug30` quadratic optimization problem using hundreds of computers at many sites [ABGL02]; and the NEESgrid system that integrates earthquake engineering facilities into a national laboratory [PKF+01].

In contrast, P2P systems tend to be vertically integrated solutions to specialized problems: currently deployed systems share either compute cycles or files. Diversification comes from differing design goals, such as scalability [RFH+01, SMK+01, RD01], anonymity [CSWH00], or availability [CSWH00, KBC+00].

Grid applications also tend to be far more data intensive than P2P applications. For example, a recent analysis of Sloan Digital Sky Survey data [AZV+02] involved, on average, 660 MB input data per CPU hour; and the Compact Muon Solenoid [Neg94] data analysis pipeline involves from 60 MB to 72 GB input data per CPU hour. In contrast, SETI@home moves at least four orders of magnitude less data: a mere 21.25 KB data per CPU hour. The reason is presumably, in part at least, better network connectivity.

The variety of Grid applications requires significant support from services. Applications may use not only data and computational power, but also storage, network bandwidth, and Internet-connected instruments at the same time. Unlike in file-sharing systems such as Gnutella, this variety of resources requires attribute-based identification (such as "Linux machine with more than 1 GB memory"), since globally unique names are of no significant use. Also, Grid services must provide stronger quality-of-service guarantees: a scientist that runs data-analysis applications in a Grid is less willing to wait until data is retrieved than is a typical P2P user in search for music files.

3. REQUIREMENTS FOR RESOURCE DISCOVERY

As we have noted, we expect Grid and P2P systems to converge in a unified resource-sharing environment. This environment is likely to scale to millions of resources shared by hundreds of thousands of participants (institutions and individuals); no central, global authority will have the means, the incentive, and the participants' trust to administer such a large collection of distributed resources. Participation patterns will be highly variable: there will be perhaps a larger number of stable nodes than in today's P2P systems, but many resources will join and leave the network frequently. Resources will have highly diverse types (computers, data, services, instruments, storage space) and characteristics (e.g., operating systems, number of CPUs and speed, data of various

sizes, services). Some resources will be shared following well-defined public policies, such as "available to all from 6 pm to 6 am". Other resources will participate rather chaotically, for example, when idle. Technical support will be variable: some participants will benefit from technical support, whereas others will rely on basic tools provided by community (e.g., today's Gnutella nodes run various implementations of the protocol, each with its own particularities).

To perform efficiently in these conditions, a resource discovery mechanism should have the following features:

- Independence of central, global control. This is a departure from previous Grid solutions and a step toward the fully decentralized solutions typical of P2P approaches.

- Support for attribute-based search, a feature not found in current P2P solutions.

- Scalability, which becomes more important to the Grid community with the increase in scale and participation.

- Support for intermittent resource participation, a characteristic frequent in today's P2P systems but rare in current Grid solutions.

In the following sections, we propose a scheme for characterizing resource discovery techniques. Although we are interested primarily in designing mechanisms that adhere to the requirements above, our characterization scheme comprises a more general set of solutions.

4. RESOURCE LOCATION: COMPONENTS AND SOLUTIONS

We assume that every participant in the VO—institution or individual—publishes information about local resources on one or more local servers. We call these servers nodes, or peers. Nodes hence provide information about resources: some advertise locally stored files or the node's computing power, as in a traditional P2P scenario; others advertise all the resources shared by an institution, as in a typical Grid scenario.

From the perspective of resource discovery, a 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 failure). 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 one or more nodes.

4.1 Four Axes of the Solution Space

We partition a general resource discovery solution into four architectural components: membership protocol, overlay construction, preprocessing, and query processing. This partitioning helps us recognize the unexplored regions in the solution space. It also provides the basis for comparing previous solutions from both the P2P area and the traditional distributed computing domain, solutions that we present in Section 4.2.

4.1.1 Membership Protocol

The membership protocol specifies how new nodes join the network and how nodes learn about each others (we refer to the latter part of the membership problem as *peer discovery*, although it has multiple names in the literature [KP00]).

Imagine a graph whose vertices are peers and whose edges indicate whether vertices know of each other. Ideally, despite frequent vertex departures and joins, this graph is a clique; that is, every member of the network has accurate information about all participants. In practice, however, this situation is impossible [CHTCB96], and different protocols have been suggested, each involving different tradeoffs. For example, Gnutella uses an aggressive membership protocol that maintains the highly dynamic nodes in the membership graph connected, but at a significant communication cost [RFI02]. More scalable with the number of nodes are membership protocols based on epidemic communication mechanisms [GT92].

4.1.2 Overlay Construction

The overlay construction function selects the set of collaborators from the local membership list. In practice, this set may be limited by such factors as available bandwidth, message-processing load, security or administrative policies, and topology specifications. Hence, the overlay network often contains only a subset of the edges of the membership graph. 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 [RFI02]) but knows of many more peers (hundreds) at any given time.

The overlay topology has a significant effect on performance. For example, Barabási and Albert [BA99] show a strong correlation between robustness and the power-law topology; Adamic et al. [AHLP01] give a search algorithm that exploits the power-law topology in a cost-efficient way; and Kleinberg [Kle00] 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 net-

works, all exhibit the same power-law and small-world patterns (as surveyed in [AB02] and discussed in detail in [Bar02] and [Wat99]).

4.1.3 Preprocessing

Preprocessing refers to off-line processing used to enhance search performance prior to executing requests. For example, prefetching is a preprocessing technique, but caching is not. Another example of a preprocessing technique is dissemination of resource descriptions, that is, advertising descriptions of the local resources to other areas of the network for better search performance and reliability. A third example of preprocessing is rewiring the overlay network to adapt to changes in usage characteristics.

It is not obvious, however, that such preprocessing strategies work in the dynamic environments that we consider, in which resources may leave the pool and resource characteristics and user behavior may change suddenly. A recent result [CS02] shows that, in a static environment, the optimum replication of an item for search in unstructured networks is proportional to the square root of the popularity of that item.

4.1.4 Request Processing

The request-processing function has a local and a remote component. The local component looks up a request in the local information, processes aggregated requests (e.g., a request for A and B could be broken into two distinct requests to be treated separately), and/or applies local policies, such as dropping requests unacceptable for the local administration.

The remote component implements the request propagation rule. Request propagation is currently an active research topic in the P2P area [RFH+01, SMK+01, RD01, LCC+02, ZKJ01, IRF02, SMZ03]. In some cases, request propagation rules are dictated by other components of the resource discovery mechanism, as with distributed hash tables [RFH+01, SMK+01, RD01, ZKJ01], where the overlay and the propagation rules are strongly correlated. In an unstructured network, however, there are many degrees of freedom in choosing the propagation rule. Various strategies can be employed, characterized by the number of neighbors to which a request is sent and the way in which these neighbors are selected.

4.2 Previous Solutions to Resource Discovery

To provide a basis for our proposed characterization scheme, we discuss here existing solutions and related work from the perspective of the four architectural components presented above.

Many solutions to resource discovery presume the existence of globally unique names. In some cases, this naming scheme is natural (for example,

filenames used as global identifiers in P2P file-sharing systems); in others, it is created to support discovery. In the context of Grid computing it is difficult (if even possible) to define a global naming scheme capable of supporting attribute-based resource identification. We now present resource location solutions that exploit natural naming schemes.

Domain Name Service [Moc87] is perhaps the largest such system that provides name-based location information. Its hierarchical topology dictates the design of all four components: nodes (domains) join at a specified address in the hierarchy, the overlay function maintains the domain-based tree structure, requests are propagated upward in the hierarchy.

Recent contributions to name-based resource location solutions have been proposed in the context of P2P file-sharing systems, such as Gnutella and Napster. The basic mechanism used in Gnutella is *flooding*. Its flooding-based membership component manages a highly dynamic set of members (with median lifetime per node of about 60 minutes [SGG02]) by sending periodic messages. Its overlay function selects a fixed number of nodes from those alive (in most instances, the first nodes of the membership list). Flooding is also the core of the request-processing component: requests are propagated in the overlay until their time-to-live expires. No preprocessing component is active in Gnutella. Answers are returned along the same trajectory, from node to node, to the node that initiated the request. Gnutella's relatively good search performance (as measured in number of hops) is achieved at the cost of intensive network use [RFI02].

Napster uses a centralized approach: a file index is maintained at a central location, while real data (files) are widely distributed on nodes. The membership component is centralized: nodes register with (and report their locally stored files to) the central index. Hence, the request-processing component is a simple lookup in the central index. Napster does not use a distinct overlay function.

Distributed hash table structures such as CAN [RFH$^+$01], Chord [SMK$^+$01], Tapestry [ZKJ01], and Pastry [RD01] build search-efficient overlays. All have similar membership and request processing components, based on information propagation in a structured overlay. Differentiating these four solutions is the definition of the node space, and consequently the overlay function that preserves that definition despite the nodes' volatility: ring in Chord, d-coordinate space on a torus in CAN, Plaxton mesh [PRR97] in Pastry and Tapestry.

The file location mechanism in Freenet [CSWH00] uses a request-propagation component based on dynamic routing tables. Freenet includes both file management and file location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear.

The solutions discussed above are concerned with locating resources that inherently can be named. Solutions that create an artificial name have been pro-

posed for attribute-based service location (Ninja) and as location-independent identifiers (Globe).

In Ninja's service location service [GBHC00, GWvB+01], services are named based on a most relevant subset of their attributes. Its preprocessing component disseminates lossy aggregations (summaries) of these names up a hierarchy. Requests are then guided by these summaries up or down the hierarchy, in a B-tree search fashion. The fix overlay function (hence, the construction of the hierarchy) is specified at deployment.

The location mechanism in Globe [vSHT99] is based on a search-tree-like structure where the search keys are globally unique names. Its naming service [BvST00] transforms a URL into a location-independent unique identifier.

Among the few attribute-based resource location services is Condor's Matchmaker (Chapter **??**, [RLS98]). Resource descriptions and requests are sent to a central authority that performs the matching.

Lee and Benford [LB98] propose a resource discovery mechanism based on request propagation: nodes (called *traders*) forward unsolved requests to other nodes in an unstructured overlay. The overlay function takes into account neighbors' expertise and preference: a node connects to a node that has useful services and/or good recommendations. This evaluation uses information collected by the preprocessing component: traders explore the network off-demand, whenever necessary, and disseminate state changes via flooding.

Another solution is provided by the Globus Toolkit MDS [CFFK01]. 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 ("nodes" in our terminology) via a registration protocol. Nodes and users can use an enquiry protocol to query other nodes to discover entities and to obtain more detailed descriptions of resources from their information sources. Left unspecified is the overlay construction function, the techniques used to associate information sources to nodes and to construct an efficient, scalable network of index servers.

5. EXPERIMENTAL STUDIES

Our objective is to observe and quantify the synergies emerging from the interaction of the four components of resource discovery in flat, unstructured networks. In the absence of a large-scale, deployed Grid available to test design ideas, we modeled an environment in which we experimented with a set of resource discovery mechanisms. This emulated Grid, while specifically designed to test resource location ideas, can be easily expanded to evaluate other services on large-scale testbeds, such as resource selection and scheduling. More important, it provides a framework for evaluating aggregations of cooperative services, such as resource location, resource selection, and scheduling.

5.1 Emulated Grid

Existing Grid simulators are specialized for certain services, such as scheduling [LMC03] or data replication [RF02]. Others, such as the MicroGrid [SLJ+00], run Grid software and applications on virtual resources. No current simulator is appropriate for or easily extensible to evaluating generic Grid services. We built an emulated Grid that is scalable and is suitable for resource discovery but also is easily extensible to other purposes.

In our framework, nodes form an overlay network. Each node is implemented as a process that communicates with other nodes via TCP. Each node maintains two types of information: (1) information about a set of resources and (2) information about other nodes in the overlay network (including membership information).

The large number of processes needed by our large-scale emulation raises multiple problems, ranging from resource starvation to library limitations. For the preliminary experiments (of up to 32,768 virtual nodes) presented in Section 6, we used 128 systems (256 processors) communicating over fast Ethernet of the Chiba City cluster of Argonne National Laboratory. With minimal modifications, the framework could be used in real deployments.

5.2 Modeling the Grid Environment

Four environment parameters influence the performance and the design of a resource discovery mechanism:

1 *Resource information distribution and density*: Some nodes share information on a large number of resources, whereas others share just on a few (for example, home computers). Also, some resources are common (e.g., PCs running Linux), while others are rare or even unique (e.g., specific services or data).

2 *Resource information dynamism*: Some resource attributes are highly variable (e.g., CPU load or availably bandwidth between two nodes), while others vary so slowly that they can be considered static for many purposes (e.g., operating system version, number and type of CPUs in a computer, etc.).

3 *Request popularity distribution*: The popularity of users' requests for resources varies. For example, studies [BCF+99] have shown that HTTP requests follow Zipf distributions. Our analysis [IR03] of a scientific collaboration, on the other hand, reveals different request popularity patterns, closer to a uniform distribution.

4 *Peer participation*: The participation of peers, or nodes, varies, more significantly in P2P systems than in current Grids. Influenced by incentives, some nodes activate in the network for longer than others.

The failure rate in a large-scale system is inevitably high and hence necessary to model. This factor can easily be captured by two of the parameters just listed, namely, resource information dynamism and peer participation. The effects of failure 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. Resource failure can therefore be seen as yet another example of resource attribute variation and can be treated as part of resource information dynamism. When nodes fail, not only do their resources disappear, but they cease to participate in maintaining the overlay and processing remote requests. Node failures can therefore be captured in the peer participation parameter as departures. We note, however, that node failures are ungraceful (unannounced) departures. Moreover, such failures may not be perceived in the same way by all peers; for example, in the case of network partitioning, a node may seem failed to some peers and alive to others.

To isolate some of the correlations between the many parameters of our study, we used a simplified, optimistic Grid model characterized by static resource attributes, constant peer participation, and no failures. Thus, we model only the resource and request distributions.

5.2.1 Resource Distributions

In Grids and peer-to-peer environments, the total number of resources increases with the number of nodes, so we model this as well. We assume that the average number of resources per node remains constant with the increase in the network size: in our experiments, we (arbitrarily) chose this constant equal to 5.

New nodes often bring new types of resources, however, such as unique online instruments, new data, and new, possibly locally developed, applications. To account for these, we allowed the set of resource types to increase slowly (5%) with the number of nodes in the system.

In this context, we experimented with two resource distributions of different degrees of fairness, as presented in Figure 1.1: a balanced 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.

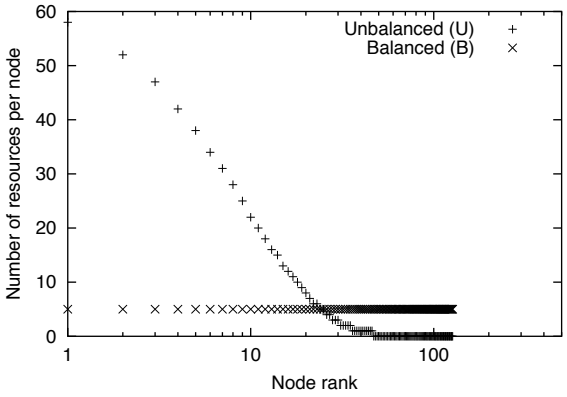

Figure 1.1. Distribution of resources on nodes: balanced (all nodes have equal number of resources) and unbalanced (a significant part of nodes have no resources).

5.2.2 Request Distributions

Although usage patterns can be decisive in making design decisions, we faced the problem of not having real user request logs, a problem inherent in systems during the design phase. We therefore logged, processed, and used one week's requests for computers submitted to the Condor [LLM88] pool at the University of Wisconsin. This pool consists mainly of Linux workstations and hence is a rather homogeneous set of resources. On the other hand, since it is intensively used for various types of computations, the requests specify various attribute values (e.g., for minimum amount of available memory or required disk space). We processed these requests to capture their variety. We acknowledge, however, that 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.

We also experimented with a synthetic request popularity distribution modeled as a uniform distribution. Figure 1.2 highlights the differences between the two request distributions. The Condor traces exhibit a Zipf-like distribution, where a small number of distinct requests appear frequently in the set of 2000 requests considered. In the pseudo-uniform distribution, on the other hand, requests are repeated about the same number of times. We evaluated various resource location strategies in overlay networks ranging in size from 2^7 to 2^{15} nodes. In our experiments we randomly chose a fixed percentage of nodes to which we sent independently generated sets of 200 requests. The same sets of requests, sent to the same nodes, respectively, were repeated to compare various request-forwarding algorithms.

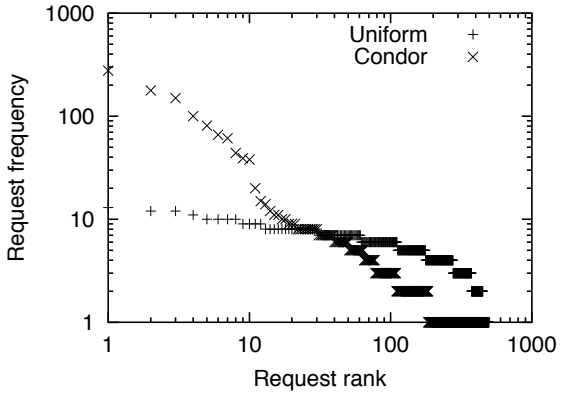

Figure 1.2. Distribution of user requests.

5.3 Resource Discovery Mechanisms

We considered a set of simple resource discovery mechanisms constructed by fixing three of the four components presented in Section 4.1 and varying the fourth: the request-processing component.

For the *membership protocol* we use a join mechanism that is commonly used in P2P systems: a node joins by contacting a member node. Contact addresses of member nodes are learned out-of-band. 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.

In our design, the *overlay function* accepts an unlimited number of neighbors: hence, we allowed the overlay connectivity to grow as much as the membership information. In this way, we neutralized one more component, aiming to understand the correlations between graph topology and discovery performance. We generated the starting overlay by using a hierarchy-based Internet graph generator [Doa96]. We assumed no *preprocessing*.

Our design of the *request-processing component* is based on forwarding. We assumed simple requests, satisfiable only by perfect matches. Hence, local processing is minimized: a node that has a matching resource responds to the requester; otherwise, it decrements TTL and forwards it (if TTL>0) to some other node. Requests are dropped when received by a node with no other neighbors or when TTL=0.

We evaluated four request propagation strategies:

1 *Random walk*: the node to which a request is forwarded is chosen randomly. No extra information is stored on nodes.

2 *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.

3 *Best-neighbor*: the number of answers received from each peer is recorded (without recording the type of request answered). A request is forwarded to the peer who answered the largest number of requests.

4 *Learning-based + best-neighbor*: this strategy is identical with the learning-based strategy except that, when no relevant experience exists, the request is forwarded to the best neighbor.

6. EXPERIMENTAL RESULTS

This section presents preliminary results in two areas: (1) quantification of the costs of simple resource discovery techniques based on request- forwarding (no preprocessing), and (2) effects of resource and request distributions on resource discovery performance.

6.1 Quantitative Estimation of Resource Location Costs

A first question to answer is: What are the search costs in an unstructured, static network in the absence of preprocessing? To this end, we considered time-to-live infinite. The answer is presented in Figures 1.3, 1.4, and 1.5: the learning-based strategy is the best regardless of resource-sharing characteristics, with fewer than 200 hops response time per request for the largest network in our experiment. For a network of thousands of nodes (hence, possibly thousands of institutions and individuals) the average response time is around 20 hops. Assuming 20 ms to travel between consecutive nodes on a path (10 ms. latency in a metropolitan area network and 10 ms. necessary for request processing), then a path of 20 hops takes less than half a second.

Key to the performance of the learning-based strategy is the fact that it takes advantage of similarity in requests by using a possibly large cache. It starts with low performance until it builds its cache.

The random-forwarding algorithm has the advantage that no additional storage space is required on nodes to record history. We also expect it to be the least efficient, however, an expectation confirmed by the results shown in Figure 1.5 (Condor-based user requests, unbalanced resource distribution). For all network sizes in our experiments, the learning-based algorithm consistently performs 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 in 1.5).

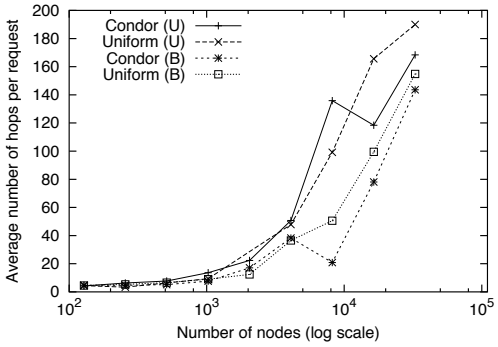

Figure 1.3. Performance (in average number of hops) of learning-based forwarding strategy for the two request distributions (Condor and uniform), in two environments with different resource-sharing characteristics (balanced B and unbalanced U).

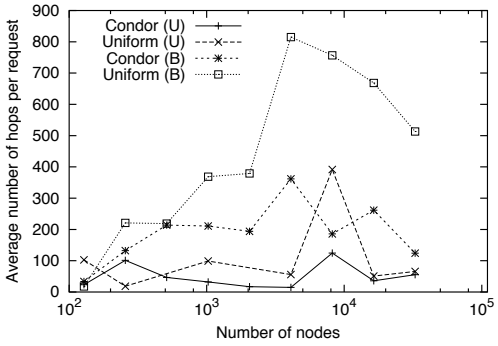

Figure 1.4. Performance (in average number of hops) of the best neighbor request forwarding strategy under different user request and sharing characteristics.

We emphasize that these results do not advocate one strategy over another but give a numerical estimate of the costs (in response time) involved. These estimates are useful in at least two ways. First, they give a lower bound for the performance of resource location mechanisms based on request propagation. They show that more sophisticated strategies (potentially including preprocessing techniques) are needed for efficient resource location in large-scale (tens of thousands institutions) Grids. Second, they can be used in estimating the performance of more sophisticated mechanisms that have a request-propagation component (as is, for example, the solution in [IRF02]).

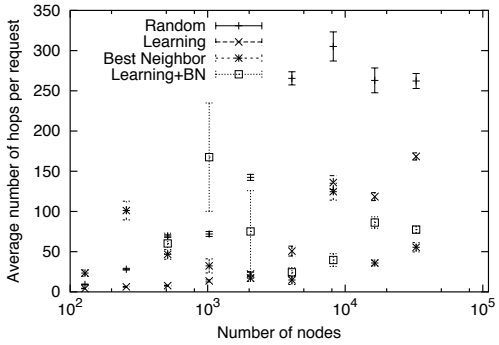

Figure 1.5. Performance of all four request-forwarding strategies for a Condor request load in the unbalanced resource-sharing environments.

6.2 Effects of the Environment

Figure 1.3 highlights the influence of user request popularity distribution 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 forwarding has a better chance of reaching a useful node when information is distributed fairly on nodes. The learning-based strategy takes most advantage of the Condor request distribution, where a significant part of the requests are repeated (and hence can benefit from previous experience).

The best-neighbor strategy is influenced more strongly by sharing patterns: compared with 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 1.4 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 with the unbalanced one. We 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 answered n distinct requests. This fact explains why the algorithm performs better under a uniform user distribution load than under the Condor traces: since the number of distinct requests in a uniform distribution is larger, the best neighbor identified by this strategy has indeed a larger number of distinct resources.

7. SUMMARY

We estimate that the characteristics and the design objectives of Grid and P2P environments will converge, even if they continue to serve different communities. Grids will increase in scale and inherently will need to address intermittent resource participation, while P2P systems will start to provide more complex functionalities, integrating data and computation sharing with various quality of service requirements. We are therefore studying 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, including data, applications, computers, online instruments, and storage) with the scale, dynamism, and heterogeneity of today's P2P systems.

We have identified four components that, we believe, can define any decentralized resource discovery design: membership protocol, overlay function, preprocessing, and request processing.

We have also proposed a Grid emulator for evaluating resource discovery techniques 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.

Acknowledgments

We are grateful to Daniel C. Nurmi for his help with deploying the Grid emulator on the Chiba City cluster at Argonne National Laboratory. This work was supported by the National Science Foundation under contract ITR–0086044.

References

[AB02] R. Albert and A. L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[ABGL02] K. Anstreicher, N. Brixius, J.-P. Goux, and J. T. Linderoth. Solving large quadratic assignment problems on computational Grids. *Mathematical Programming*, 91(3):563–588, 2002.

[ACK+02] D. P. Anderson, J. Cobb, E. Korpella, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45:56–61, 2002.

[AH00] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5, 2000. Also available from `http://www.firstmonday.dk/issues/issue5_10/adar/`.

[AHLP01] L. Adamic, B. Huberman, R. Lukose, and A. Puniyani. Search in power law networks. *Physical Review E*, 64:46135–46143, 2001.

[AK02] D. P. Anderson and J. Kubiatowicz. The worldwide computer. *Scientific American*, March 2002.

[AZV+02] James Annis, Yong Zhao, Jens Voeckler, Michael Wilde, Steve Kent, and Ian Foster. Applying Chimera virtual data concepts to cluster finding in the Sloan Sky Survey. In *Proceedings of SuperComputing (SC'02)*, 2002.

[BA99] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[Bar02] A. L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of InfoCom*, 1999.

[BvST00] G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Scalable naming in global middleware. In *Proceedings of Thirteenth International Conference on Parallel and Distributed Computing Systems (PDCS-2000)*, 2000.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[CHTCB96] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, 1996.

[CS02] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the SIGCOMM*, 2002.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[Doa96] M. Doar. A better model for generating test networks. *IEEE Global Internet*, 1996.

[FI03] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and Grid computing. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[GBHC00] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for Internet service construction. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.

[GT92] R. A. Golding and K. Taylor. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, Jack Baskin School of Engineering, University of California, Santa Cruz, 1992.

[GWvB+01] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. D. Joseph, R. H. Katz, Z. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust Internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing*, 35(4):473–497, 2001.

[IR03] A. Iamnitchi and M. Ripeanu. Myth and reality: Usage patterns in a large data-intensive physics project. Technical Report TR2003-4, GriPhyN, 2003.

[IRF02] A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 2000.

[Kle00] J. Kleinberg. The small-worlds phenomenon: an algorithmic perspective. In *Proceedings of the Thirty-Second ACM Symposium on Theory of Computing*, 2000.

[KP00] S. Kutten and D. Peleg. Deterministic distributed resource discovery. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2000.

[LB98] O. Lee and S. Benford. An explorative approach to federated trading. *Computer Communications*, 21(2), 1998.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the Sixth Annual ACM International Conference on Supercomputing (ICS)*, 2002.

[LLM88] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[LMC03] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The SimGrid simulation framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, 2003.

[Moc87] P. Mockapetris. Domain names–concepts and facilities. Technical Report RFC 1034, Internet Engineering Task Force (IETF), 1987.

[Neg94] M. D. Negra. CMS collaboration. Technical Report LHCC 94-38, CERN, 1994.

[PKF+01] T. Prudhomme, C. Kesselman, T. Finholt, I. Foster, D. Parsons, D. Abrams, J.-P. Bardet, R. Pennington, J. Towns, R. Butler, J. Futrelle, N. Zaluzec, and J. Hardin. NEESgrid: A distributed virtual laboratory for advanced earthquake experimentation and simulation: Scoping study. Technical report, NEESgrid, Technical Report, 2001. Available from `http://www.neesgrid.org/documents/NEESgrid_TR.2001-01.pdf`.

[PRR97] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[RD01] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware '01*, 2001.

[RF02] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data intensive applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM 2001*, 2001.

[RFI02] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing*, 6, 2002.

[RLS98] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, 1998.

[SET] SETI@home: The Search for Extraterrestrial Intelligence. `http://setiathome.berkeley.edu`.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of SPIE Multimedia Computing and Networking 2002 (MMCN'02)*, 2002.

[Shi00] C. Shirky. What is P2P...and what isn't? `http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html`, 2000.

[SLJ+00] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A scientific tool for modeling computational Grids. In *Proceedings of SuperComputing (SC'00)*, 2000.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[SMZ03] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceeding of INFOCOM*, 2003.

[TMB00] M. P. Thomas, S. Mock, and J. Boisseau. Development of Web toolkits for computational science portals: The NPACI HotPage. In *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, 2000.

[vSHT99] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7(1):70–78, January 1999.

[Wat99] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999.

[WO02] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, Berkeley, 2001.