Scalable Computing: Practice and Experience
Volume 6, Number 2, pp. 23–33. http://www.scpe.org
ISSN 1895-1767
© 2005 SWPS

TOWARDS A ROBUST AND FAULT-TOLERANT MULTICAST DISCOVERY ARCHITECTURE FOR GLOBAL COMPUTING GRIDS

ZOLTAN JUHASZ[*], ARPAD ANDICS[†], KRISZTIAN KUNTNER[†] , AND SZABOLCS POTA[†]

Abstract. Global grid systems with potentially millions of services require a very effective and efficient service discovery/location mechanism. Current grid environments, due to their smaller size, rely mainly on centralised service directories. Large-scale systems need a decentralised service discovery system that operates reliably in a dynamic and error-prone environment. Work has been done in studying flat, decentralised discovery architectures. In this paper, we propose a hierarchical discovery architecture that provides a more scalable and efficient approach. We describe our design rationale for a k-ary tree-based fault-tolerant discovery architecture that also acts as an intelligent routing network for client requests. We show that it is possible to provide ad hoc multicast discovery of services even without end-to-end multicast availability. The system provides clients with the ability to search for services with incomplete information using descriptive service attributes. Our results demonstrate that this approach is well suited to wide-area discovery if fast discovery, high availability, scalability and good performance are crucial requirements.

Key words. Grid, global computing, discovery, multicast, hierarchical overlay, service

1. Introduction. Service discovery is an essential component of global grid systems where the number of services can be in the order of millions. Services are dynamic; they may join or leave the system as they wish. Hardware or software components may fail at any time, services can be migrated to or service replicas can be launched at new locations. Consequently, without having an effective solution for locating suitable services in a wide-area environment, grids will not be usable.

A global discovery architecture must provide the user with the ability to discover services or resources dynamically, as in general it cannot be assumed that the exact machine name or URL of the service is known. To be highly available, it should also be fully distributed with no single point of failure. It should be scalable in order to cope with the increase in size, and it should provide powerful search facility to locate services based on incomplete information.

Different aspects of wide-area service location and discovery have been the focus of research in various contexts. New directory services such as X.500, LDAP, UDDI have been developed to facilitate attribute-based service/resource search. Extensions have been proposed to create wide-area name/directory systems and augment the DNS name service [3, 24]. These systems, however, still require knowledge of at least one directory service. Discovery has been proved to be an effective solution for this entry-point problem as demonstrated by several local area discovery frameworks [8, 11, 26]. It is also a key element of peer-to-peer (P2P) systems that, consequently, have proposed various discovery schemes [7, 15, 21, 6]. Overlay routing networks have been proposed in certain peer-to-peer and distributed multimedia systems mainly to provide better global properties and multicast traffic among non-multicast reachable system components [5, 7].

Although wide-area service discovery is crucial to grids, it has received somewhat less attention within the grid research community. Current grid environments typically rely on an LDAP-based directory service that has proven scalability, update and query problems [19]. Future grid infrastructures based on Web Services technology will rely on UDDI directories, but it is yet unclear how this will provide a solution for large-scale, wide-area systems.

In this paper, we propose and describe a hierarchical discovery architecture that integrates techniques developed in different areas of distributed systems. This architecture relies on infrastructure services to create a self-configuring hierarchical routing overlay network that provides effective support for global resource and service discovery; it provides multicast discovery without the need for universal multicast availability; it is scalable, fully decentralised, self-configuring and robust; can be updated dynamically. The system also supports the execution of complex search queries based on service interfaces and attributes and—acting as a gateway—provides seamless wide-area access to services and resources.

The structure of our paper is as follows. In Section 2 we briefly review recent results related to wide-area service location and discovery, focusing on discovery, hierarchical naming systems, overlay networks and multicasting. Section 3 outlines the main requirements for our proposed discovery architecture. Section 4

[*]Department of Information Systems, University of Veszprem, Egyetem u 10., Veszprem, 8200 Hungary. ({juhasz, kuntner, pota}@irt.vein.hu). Zoltan Juhasz is also with the Department of Computer Science, University of Exeter, UK.

[†]Computer and Automation Research Institute, Hungarian Academy of Sciences, Hungary. (andics@sztaki.hu).

concentrates on the design aspects of the system discussing the need for Broker services, flexible matching, and an overlay routing network. Section 5 focuses on the details of the configuration, operation and fault-tolerance of the hierarchical broker network. In Section 6 we describe our results to date obtained on small testbeds. We discuss potential ways for improving our system in Section 7, then end the paper with our conclusions.

2. Related work. With the widespread acceptance of the Internet and World Wide Web, the limitations of commonly used name and directory services have become apparent. Researchers have focused on various aspects of wide-area service discovery but little research has been done for global grid service discovery.

Accepted directory service technologies, such as LDAP [25] and UDDI [2] work on the assumptions that stored data is mainly static, updates are rare, and clients know how to connect to the directory. These systems do not provide global directory services and the removal of stale entries is a key problem.

Several alternative systems have been proposed as alternative wide-area naming and directory systems. Both the Intentional Naming System [3] and the Globe Location Service [24] use attribute-based matching and hierarchies to access information effectively. They however focus on information access rather than on providing access to services via discovery.

Several local discovery schemes and protocols have been proposed for network and directory discovery. Examples are Universal Plug-and-Play [23], Service Location Protocol [11], Salutation [22], Jini [26]. While they solve the discovery entry point problem, none of them provide a solution for wide-area discovery. The Service Discovery Service [8] architecture was an attempt to create a global discovery architecture. It is based on a hierarchical, secure network of directory nodes. Unfortunately, only plans existed for the wide-area support, no implementation and global tests have been reported. Another proposed hierarchical service discovery architecture is the Carmen system [17]. It is a dynamic, updatable wide-area service discovery scheme, but at the time of writing it lacks full wide-area support and security, and it has to be manually configured.

Another line of research is to use peer-to-peer computing [18] concepts for discovery. One example system is the scalable content-addressable network [20] that uses a d-dimensional hash-based routing overlay over the mesh of nodes. Another proposed P2P discovery system is by Iamnitchi and Foster for grids [14]. It is a decentralised, self-configuring architecture based on P2P routing in a flat network. Their system can use four different request-forwarding policies that are combinations of random and best neighbour routing with learning from experience. The Web Services Discovery Architecture [13] proposal is an interesting system, providing advanced query facilities. The architecture is based on Web Services technology, but uses P2P concepts to provide discovery. It is unclear however, how effective wide-area discovery is supported.

The main general problem with P2P discovery approaches is the unbounded discovery time due to the unbounded hops to travel during discovery, which makes their effective use in large discovery systems used by millions rather doubtful.

Internet content distribution systems rely on overlay networks, mainly to provide end-to-end multicast without router support. The Overcast [15] system was designed to work in a single source setting, building a distribution tree automatically and dynamically. Scattercast [7] on the other hand is designed to provide multicast between multiple sources and sinks. It relies on special infrastucture service agents, Scattercast Proxies, to make multicast available over a unicast network.

Although much work has been done on various aspects of wide-area service discovery to date, no discovery architecture yet exists that would provide seamless access to global services, offer a powerful and effective query functionality at the same time, and operate in a spontaneous, highly-available, fault-tolerant, and dynamic manner. Such a discovery system is necessary for the success of grids, and for this reason, it should be more in centre of attention of the grid research community.

3. Service discovery system requirements and design rationale. Designing a discovery architecture capable of supporting service and client populations in the order of hundred millions is not a trivial task. The system has to meet a set of demanding requirements. In the following, we only describe the most important ones.

The number one requirement for the discovery system is to provide seamless access to services at a global scale, since future grid users would like to access grid resources and services without any hassle, administrative difficulty or knowing that they are actually accessing the grid.

A good discovery system should support the discovery of services without *a priori* information about their location (*location transparency*), or about the number of instances a service may have in the network (*replication transparency*) using flexible and powerful associative search mechanisms. Since the discovered services most

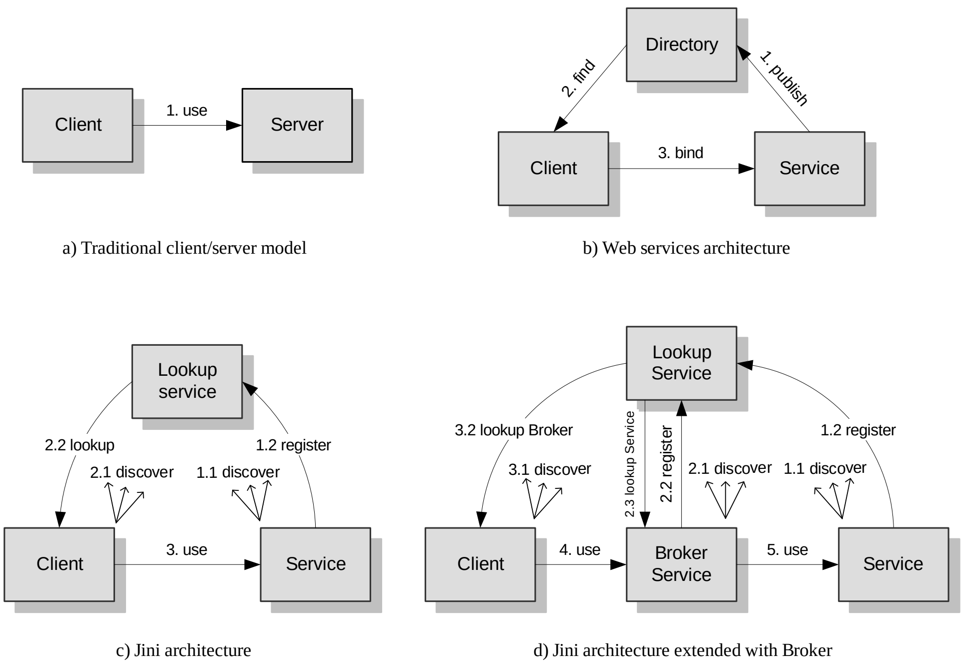

FIG. 2.1. *Different levels and approaches of de-coupling in client-server architectures.*

likely will be used by other programs, the search should be based on well-known service interfaces and additional service attributes. The system should provide means to perform complex queries effectively on a wide-area scale.

In addition, it should be possible to expand the system in size without changing either its overall structure or the internals of any of its components (*scalability*) due to either the increase of system components or to the increased load on certain components of the system (*performance transparency*). The system should also hide the location details of clients and services thus enabling the use of client and service mobility (*mobility transparency*).

The system should not become vulnerable to errors, it needs to be fault-resilient and robust. There cannot be a single point of failure in the system, and in the case of unrecoverable errors, the system should degrade gracefully both in its functionality and performance. In very large service communities, it is inevitable that services may come and go (leave willingly or due to a crash, and appear after initial start up or reboot), therefore the system should provide facilities for change management. It should be able to handle resources in a dynamic way and provide means to keep and present a soft system state the users.

4. Architectural design. In this section, we describe the general architecture of our discovery system step by step. First, we make a case for using Broker services, discuss how flexible search can be provided, then outline our proposed hierarchical, self-configuring and fault-tolerant discovery overlay network. To create this architecture we rely on the following techniques: multicast communication, distributed events, object mobility, proxy pattern, smart replication and caching.

The current architectural trend for distributed systems is based around the notion of directory services. Directories, as illustrated in Fig. 2.1(b), de-couple the traditionally tight client-server relationship, Fig. 2.1(a), by introducing a service registration, lookup and use operational mechanism. These techniques can be found in all major distributed technologies such as CORBA, Java RMI, and Web Services Architecture.

We believe that in their current form these directory services are not suitable for wide-area service discovery. As a search result, they all return a list of services to the client and require client interaction to select the suitable service. Furthermore, should a larger set of services to be discovered, the client will need to iteratively discover and look up several additional directories. Figures in Table 4.1 and 4.2 show that for a large number of directories, this can take considerable time.

Another obvious disadvantage of these systems is the so-called entry point problem; clients need to know the access details of at least one directory to be able to discover services. Should that central directory fail, the

Table 4.1
Unicast discovery times [msec] of lookup services running at different locations. The operation was initiated from Veszprem.

LUS location	Lookup service discovery time [msec]			
	min	median	mean	max
Local LAN	35	60	61	288
United Kingdom	140	165	195	9586
Norway	178	204	216	18999
USA	602	637	648	10914

Table 4.2
Minimum response time [msec] for lookup operations invoked from Veszprem on lookup services at different locations. Each lookup service was empty in order to obtain a lower bound on lookup execution time.

Location	Service lookup time [msec]			
	min	median	mean	max
Local LAN	4.8	5.2	5.5	29.1
Norway	54	58	69	10141
USA	154	160	174	8822

entire system becomes paralysed.

4.1. The Broker service. Jini extends this structure with multicast discovery, so the location of the directory no longer needs to be known. Once the client discovers the directory, as shown in Fig. 2.1(c), it can look up services then use the matching results the regular way. Multicast discovery removes the need for the client to discover directories iteratively. As the result of discovery is a set of directory proxy objects, caching can be used to keep directory references alive avoiding the need for initiating the discovery process repeatedly. This solution, however, still has problems. Firstly, it still relies on the client to carry out the potentially complex evaluation process of the returned search results. Secondly, caching directory proxies places extra memory and processing overhead on clients, which—in case of limited devices (PDAs, mobile phones)—is not permissible.

An intermediate service—a Broker—can be used to solve these problems. The Broker service (introduced in our Jini Metacomputer (JM) prototype [16], now called JGrid) is a mediator between clients and services, a single point of contact, whose role is to off-load work form the client. The main characteristic of the Broker is that it can perform tasks on behalf of the client and return only the result, not solely matching services, which distinguishes it from matchmaking [10] directory services, such as LDAP, UDDI, RMI registry or the Jini Lookup Service. This architecture, Fig. 2.1(d), allows the client to access services via a single contact point, without having to worry about the discovery and selection of services.

In the JM system, the Broker automatically discovers and caches Compute services. When clients ask the Broker to perform a computational task by sending a Java program to it, the Broker will try to find matching services stored in its lookup cache. The Jini event mechanism ensures that the service cache of the Broker will be automatically updated on changes to Compute service state. The Broker then selects the most suitable set of services, starts and controls the execution of the task and finally returns the result to the client.

4.2. Flexible matching. Besides providing service discovery, the other major role of the Broker is to select suitable services for clients. Normally, this would mean initiating several directory search operations and then evaluating the results. Our Broker provides more, however. To overcome the limitations of strict matching within the Jini lookup service (i.e. the only operator is '='), we provided search facility in the Broker with inexact matching capabilities. This is based on caching service proxies and attributes locally within the Broker service with our custom search operators ($<$, $\leq$, $>$, $\geq$; string matching; boolean AND, OR, NOT; set union $\cup$, intersection $\cap$) that facilitate the composition and evaluation of complex queries. Examples of possible queries are:

- *Return all available Compute services with more than 256 processors*,
- *Execute my task on a Compute service in Veszprem using 16 processors and minimum 1GB memory*, or
- *Return all services (any type) available in London and Vienna.*

With these extensions, the Broker can provide seamless access to services managed by brokers. The client can send a task for execution to the broker along with execution requirements, then wait for the result returned from the Broker upon its successful execution. Equally, the client can also use the Broker in the traditional matchmaking mode, only to receive a list of matching services, should it like to perform service selection itself, e.g. controlled by the user through a graphical interface.

4.3. The overlay routing network. The architecture we presented and discussed thus far is incapable of providing access to services outside the Broker's multicast region[1], hence only local services are available. In a practical wide-area system, such as grids, Brokers should provide seamless access to remote services as well. The two major technical issues to be solved are (i) the entry point problem (how to find the first point of contact), and (ii)how to provide access to remote services.

One approach, which would solve both problems, is to provide multicast routing on the Internet at a global scale. Theoretically, this would allow anyone to discover any service on the network assuming a large enough initial TTL value; consequently, our Broker could discover all interesting services. There are two problems with this approach though. Firstly, multicast is still not available everywhere on the network, and while large progress has been made to enable multicast on major backbones [9], it will take considerable time before (if ever) it becomes pervasive. Secondly, not being able to control generated network traffic and network use is a major obstacle. If clients use unnecessary large TTL values, their packets will reach those parts of the network, and consequently increase the load there, that are not intended recipients of these packets. Hence, the pure global multicast solution does not seem as a viable approach.

The other suggested approach is to federate Jini lookup services [1] in a peer-to-peer style by having lookup services register with other lookup services. In this way, a client could detect—without even using a Broker service—another lookup service by finding its proxy in the currently accessed lookup service. Using the proxy, the client can hop to the other lookup service and continue the search for suitable services. In essence, this is *hyperlinks* for Jini lookup services. One problem with this approach is that lookup services still need to know about each other to be able to register. If multicast is not available, either multicast packet tunnelling or name/address-based unicast discovery must be used, which requires manual administration and setting up. The other major problem is related to the use of the resulting lookup service structure. Clients need to be intelligent enough to detect lookup service proxies and to know how to perform hopping and subsequent search. This unnecessarily complicates client code. However, if we were to move this operation from clients to lookup services, the Jini lookup service specification would need to be changed. Finally, the resulting flat, peer-to-peer topology can guarantee neither the discovery of existing services nor the timely generation of search response. For these reasons we think that this approach is not viable either.

5. The broker hierarchy. In this paper, we propose a new approach to discovery: augmenting Jini lookup services with a hierarchical discovery overlay network that sits on top of lookup services. The aim of wide-area discovery systems is to make services *visible* to clients regardless of their location, which in our view does not require clients to physically discover and contact remote services by multicast packets. Our approach has several advantages.

- The introduction of the discovery overlay creates a layered architecture with a clear separation of responsibilities. The lowest, lookup service, layer is responsible for local service discovery and acts as an information source to the discovery layer (Brokers), whose role is to take client/application queries and perform wide-area service search.
- Large systems that rely on frequent global operations have to have a hierarchical structure. This simplifies system management, increases performance and efficiency, supports scalability, and localises traffic as well as faults.
- The system becomes open to future extensions of the discovery process without the need to change underlying services and directories. E.g., it will be possible to embed intelligent agents into Brokers that provide learning from experience, negotiation and similar high-level functionality.
- Unicast and multicast discovery can augment each other, resulting in an overlay network that can operate with and without multicast-enabled network segments.

We suggest a wide-area discovery architecture based on Broker services as shown in Fig. 5.1. Nodes (brokers) are interconnected to form a k-ary tree topology with optional connecting edges among nodes within each hierarchy level. In order to facilitate efficient search, we associate each layer with a given geographical unit, starting from continents down to organisational units. The reason for using a geographical hierarchy is that we

[1]The default multicast region is normally a single segment of a LAN. If network routers forward multicast packets, the time-to-live (TTL) value determines how many routers the packet will pass through; a packet with TTL = 5 will travel through 5 routers, hence network segments less than 5 router hops away from the source will receive the multicast packet.

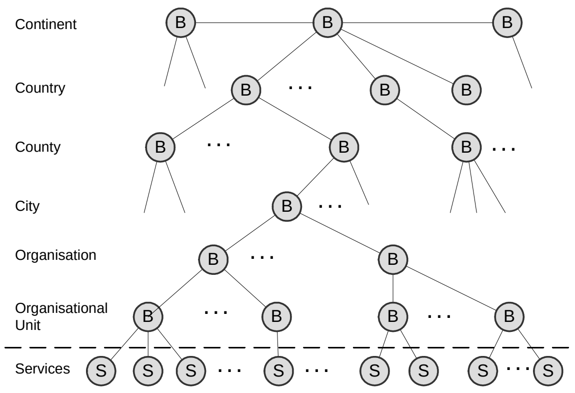

Fig. 5.1. *The high-level view of the discovery overlay system. Services are abstracted out and made available by Broker services connected in a hierarchy. A local (Organisational Unit) Broker can seamlessly provide access to distant services/resources for clients by contacting and querying other brokers.*

assumed in our system that services have rather fixed locations and institutional owners. Other hierarchies are possible as well, and we will return to this issue in Section 7.

5.1. Query routing. During operation, clients will contact and send a request to the lowest, organisational unit level broker. The request is given as a Jini `ServiceTemplate` object, which can specify a set of service interfaces and attributes a target service must match. A key attribute is `ServiceLocation` whose fields are identical to the layers of the broker hierarchy. By setting these fields to specific or 'don't care' `null` values, clients can indicate where the service they are looking for is, or similarly, where a submitted program should run.

Every service in the system, including brokers, uses its interface and a set of attributes to describe itself to the world. The `ServiceLocation` attribute is used to describe the location of the service during start-up. For services, this will determine which broker they connect to; for brokers, where in the hierarchy they have their place. Configuration is described in more detail in Section 5.2.

When executing tasks or requesting services, clients can pass the interface(s) of the required service(s) and descriptor attributes to the broker. If the client request does not specify the `ServiceLocation` attribute, the broker will search only its own services to find a match. Should `ServiceLocation` be specified, the broker first compares it with its own `ServiceLocation` attribute to see whether OrganisationalUnit fields match. If there is no match, the broker delegates the request to its parent for further search. The delegation process is shown in Fig. 5.2. Delegation is based on using proxies of parent and child brokers stored in local caches. Once a matching broker is found, an exhaustive subgraph search of its children is performed to find matching services. If, e.g., we specify `ServiceLocation` as Hungary and request 1000 processors, the discovery system will try to gather Compute Services totalling 1000 processors within the country, starting with local ones.

The overlay network provides very efficient routing. A query takes a maximum of $O(log_k S)$ hops, where k is the maximum degree of a broker and S is the number of services, to reach any service on the Internet. In our hierarchy, the maximum hop count is 11.

5.2. System configuration. In the previous sections we assumed the existence of the broker hierarchy. In this section we look at the process of configuring the discovery overlay system, focusing on how brokers will find and see each other.

Our primary requirement for system operation is minimal administration. We have designed the system to be self-configuring. Every node (Broker service) of the hierarchy performs multicast discovery to locate parent and child services in order to populate their internal broker proxy caches (Fig. 5.2). Each Broker has a designated level L_i, so each broker is only interested in finding L_{i+1} and L_{i-1} brokers. Leaf brokers naturally will not have broker children, they discover the exported services instead.

As discussed earlier, global multicast is not yet available, hence a multicast-only solution is not appropriate for configuring the system. Since Jini also offers unicast discovery, brokers can be configured to discover named lookup services (that hold broker proxies) using a set of URLs in the form of
`jini://name.of.lookup.service:port`. While it is normally assumed that clients can discover Broker services

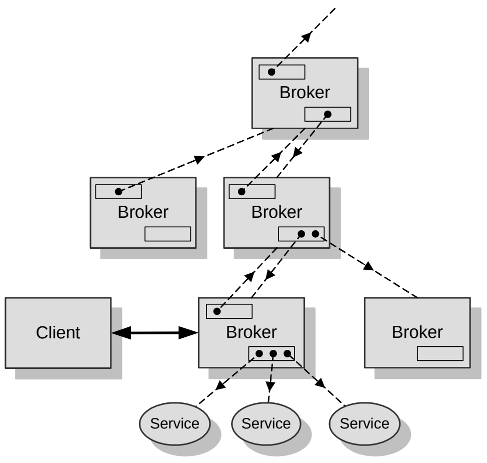

FIG. 5.2. *Request propagation in the discovery system. Client requests are routed based on attribute ServiceLocation.*

via multicast discovery on their local or institutional LAN, it is also possible to provide clients with a discovery URL to fall back to if multicast is not available. Most international and national backbones already (or soon will) provide multicast, therefore most likely it is the county/state-city-organisation layers where we need to rely on unicast discovery.

5.2.1. Multicast traffic. In a system where clients, services and the discovery system all rely to some extent on multicast, an important issue is the amount of generated multicast traffic and its effect on the network. In the following, we describe the multicast traffic patterns in our system and its effects, as well as ways to reduce the generated traffic.

The main source of multicast traffic for clients and services is discovery, during which multicast request packets are transmitted over the network. The maximum size of these packets is 512 bytes. The multicast request is sent during the client/service start-up sequence and repeated normally seven times with five-second delay intervals.

Lookup services also send multicast announcement packets to notify existing clients about their presence. These are sent by default in every two minutes. The packet size here is also max. 512 bytes. If all packets travel every segment of the Internet, the generated traffic would be high. For instance, ten million lookup services would generate a constant 40 MBytes/sec traffic by announcement packets only.

Multicast traffic can be reduced with localising multicast requests and announcements. Clients only need to discover leaf brokers that should be situated on a LAN of the client's home institution. For this, a small, typically < 5, TTL value can be used. Since brokers need to locate other brokers within small distances (few router hops away), small TTL values can be used within the broker hierarchy as well. The same is true for services, as they only need to locate lookup services within the institution. The correct threshold values depend on national network topologies, consequently require initial configuration. This approach results in mainly top-level brokers generating traffic on national and international backbones. Since their number is small (6 continents, 239 countries and approximately 3630 national regions [12]), the generated traffic is small.

Since there is no control over the initial TTL value of services and clients, two further techniques can be used to stop client and service-initiated multicast packets to travel too far. Routers can be set to stop multicast packets with an above threshold TTL, hence packets with too large TTL values, say 15, could be stopped. Another possible technique e.g. to stop multicast traffic leave a given country is to stop the Jini multicast port (4160) completely on the national gateway routers and only allow traffic over another port that tunnels the national broker multicast traffic to continental ones. These techniques could provide effective means in reducing international and national multicast traffic.

5.2.2. Unicast only mode. As discussed earlier, multicast discovery relies on sending request and announcement packets. Request packets originate from entities hoping to discover lookup services, announcement packets are sent by lookup services. Not being able to use multicast would imply loosing these functionalities, but our routing network ensures this will not happen. As long as the client can find one lookup service with a

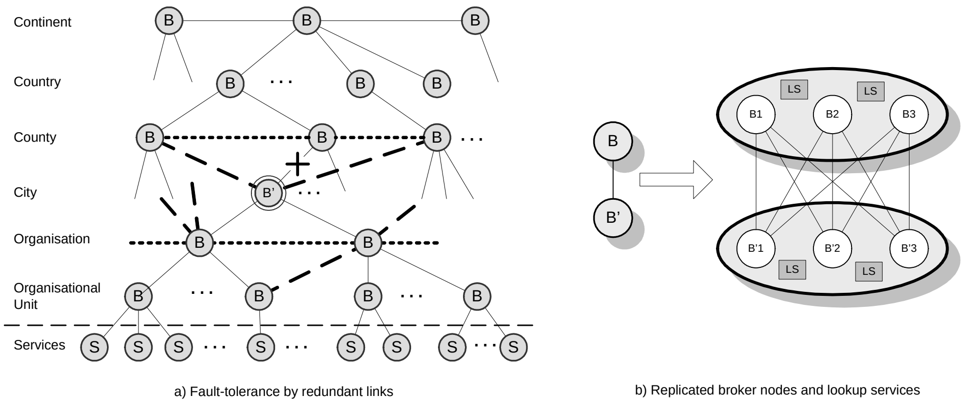

FIG. 5.3. *Link and node replication.*

leaf broker entry in it by unicast discovery, it can access any service on the network, as the broker is the only required contact point. The lack of announcement packets means that new lookup services will not be able to notify clients. Again, as long as the new lookup service can find at least one leaf broker, it will become part of the service network and clients will find the service automatically. The higher layers of the hierarchy can operate without any difficulty without multicast, as it is used only for configuration, provided the necessary configuration information is set-up.

5.3. Fault-tolerance. To guarantee service discovery, the overlay network needs to be fault-tolerant, robust and highly available. In this section we discuss what techniques can be used to achieve these goals.

The two major sources of faults in the overlay are network and node failures. To provide availability in the presence of these errors, multiple network paths and node replication can be used. In our base architecture, shown in Fig 5.1, nodes are connected in a tree topology by single links. To avoid communication breakdown among layers, redundant links can be introduced between a node and other parent layer nodes as shown in Fig. 5.3(a). Horizontal, peer links can also be introduced to provide an alternative route should all parent or children nodes fail.

Broker nodes can be made reliable by node replication. As shown in Fig. 5.3(b), a node in the overlay network can be viewed as a logical node that can have a replicated server implementation. Each broker node can be served by replicated Broker services that register with the same lookup service, hence will be automatically discovered by the relevant parent/child brokers. Note that replica brokers need not be placed at the same location; they can be physically distributed over the Internet, automatically providing alternative network paths to brokers. To avoid discovery failure, lookup services should be replicated as well, as also shown in Fig. 5.3(b). From the outside, the replica broker services are identical to the original brokers.

6. Results. To test our implementation we created small testbeds and carried out initial experiments to verify the operation and performance of the discovery architecture. We were especially interested in the response times of discovery queries on LANs and WANs, as well as the service matching time of brokers.

The crucial issue is how our discovery overlay performs with respect to the default Jini lookup service based discovery. The total lookup operation time in our overlay is $T_{lookup} = T_{delegation} + T_{matching} + T_{download}$, where $T_{delegation}$ is the time it takes for the query to reach its destination from the client, $T_{matching}$ is the time spent with service search in the leaf broker and $T_{download}$ is the time to download the service proxy objects of matching services to the client. Since $T_{download}$ is the same in both discovery systems, we ignore it in the comparisons.

6.1. Query routing performance. We performed experiments to measure delegation time, i.e. the response time of queries, to different layers in our overlay network. Two testbeds have been used. The first one, as shown in Fig. 6.1, consisted of 18 computers[2] connected to the same multicast-enabled LAN segment, while the second is a WAN configuration with computers located at two different locations in Hungary. We measured the query delegation time to different broker levels. The results of the first testbed are shown in Table 6.1. It can be seen that the average per hop time in the overlay network is around 3 *msec.*

[2]600 MHz P-III PCs running Windows 2000 and Java 1.4 HotSpot Virtual Machine.

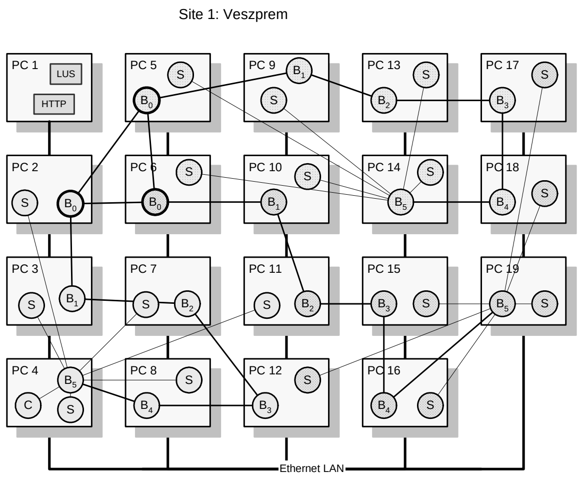

FIG. 6.1. *Wide-area search testbed. Services are assigned to three hypothetical geographical locations: America, Asia, Europe. Each computer hosts one Compute Service, S, and one node of the global broker hierarchy, B_i. Services are mapped to computers in a way to ensure all remote method invocations are performed via LAN, not in memory.*

TABLE 6.1
Minimum response time for lookup (discovery) operations initiated from Veszprem via the router overlay. No services were registered during these tests in order to obtain a lower bound on lookup execution time. Per hop time refers to the time necessary at the given level to forward the query to a parent or child node.

Delegation distance [hop]	Measured delegation time [msec]				Estimated time [msec]	
	average	min	max	per hop	total	per hop
1 (org unit)	4	0	30	1	4	2
2 (org)	6	0	50	2	6	3
3 (city)	8	0	40	2	9	5
4 (state)	10	0	40	2	14	5
5 (country)	12	10	40	2	24	10
6 (continent)	16	10	70	4	54	30
7 (other continent)	18	10	40	2	204	150
8 (country)	18	10	40	2	234	30
9 (state)	22	20	50	4	244	10
10 (city)	25	20	50	3	249	5
11 (org)	29	20	110	4	254	5
12 (org unit)	35	20	161	6	257	3

The second testbed was used to determine per hop time between two different WAN locations (9-14 *msec*). We also performed network latency measurement using `traceroute` to determine the typical latency figures for different parts of the internet. We analysed network traces between Europe, North and South America, Africa, Australia and Japan, and found that in national networks the latency is below 10 *msec*. Between countries the delay is in the order of 10 *msec*, while between continents it can be in the order of 100 *msec*. The estimated WAN per hop and accumulated response times are also given in Table 6.1. We estimate that with these broker services placed on backbone segments as infrastructure components, search results could be obtained within 100 *msec* in a continent and within few hundred milliseconds for truly global cases when intra-continent traffic is necessary.

6.2. Matching performance. To be able to reason about the total discovery time, we also measured matching time on one broker by creating several services with three attributes (`Processor`, `Storage`, `ServiceLocation`) altogether having 11 `String` and `int` fields. The measured per-service search times were 4.5 μsec for simple `int` matches (e.g. Number of processors=16), 10.6 μsec for compound `String` matches (e.g. Locality=Budapest OR Veszprem), and 17.3 μsec for compound `String` matches (e.g. (Locality=Shanghai AND MFlops $\geq$ 100000) OR organization = University of Veszprem). These results suggest that—assuming, say, maximum 1000 services per broker—matching time $T_{matching}$is in the order of 20 *msec*.

7. Discussion. Experiments with the current prototype system gave encouraging results. Without any optimisation and performance tuning, the overhead of one Broker is in the order of 2 *msec*, and global discovery can be performed in the order of hundred milliseconds. Note that these results are without using intermediate service caching. In comparison, the Globus system required seconds to discover resources of around 12 sites in case of a cache hit, and up to 10 minutes with empty GIIS caches[4].

We have also identified shortcomings of our system during the development. In our current scheme, each Broker service performs a parallel breadth-first search for services using its child Broker proxies. This, on one hand, generates large number of requests in the overlay network, on the other hand, the whole network needs to be searched if a non-existing service is requested. These problems call for a more intelligent query routing approach.

Another possible problem is the pre-determined hierarchy used in our overlay. It can be envisaged that other users or organisations would like to have services with different categorisation. Although it is certainly possible to embed other routing hierarchies in our system, that overlay would not be as efficient in its routing ability as the current one.

Finally, the geographical partitioning is not likely to be suitable for more than few million services, as the top layers of the hierarchy have low degrees. Assuming 6 continents, 239 Countries, 3630 regions, and 50027 cities [12] in the world, the average degree for continent, country and state level brokers is 47, 15 and 14, respectively. This would require many Broker services below city and organisational level, as well as result in a largely non-uniform distribution of services within the tree.

7.1. Future work. To address these issues and make our system more flexible, we are looking at ways to make the routing overlay nodes taxonomy-free, i.e. to create a general routing overlay without pre-defined classification.

This requires service attribute propagation and aggregation in the router overlay. Being able to provide content-based query routing will make search even more effective, as search request will only reach a broker if a potentially matching service can be found among its children. This architecture would naturally support the use of arbitrary service classifications and would facilitate the creation of user-defined service views. These modifications will also make it simpler to introduce redundant parent and peer links in the system. Since potential parent nodes do not belong to different "location" any longer, the fault-tolerant routing algorithm can be made simpler and more efficient.

Finally, in the next version of our overlay system, we intend to separate brokering and routing functionality by introducing router services for the overlay network, and broker services as interfaces (leaf nodes) between clients/services and the overlay network.

8. Conclusions. In this paper, we described a hierarchical discovery architecture that provides effective support for wide-area service discovery. The architecture is fully decentralised, self-configuring and scalable. It can be updated dynamically and acts as an intelligent router network for client queries. The hierarchy is based on broker services each performing multicast and unicast discovery.

The system supports ad hoc, attribute-based service discovery, even without end-to-end multicast network support and provides seamless wide-area access to services. Clients interact with the system by connecting to brokers at the organisational unit level that will delegate search requests to other nodes as necessary.

Our first experiments confirm that this approach is well suited to discovery in large grid environments. The discovery architecture has very good scalability, it is efficient for both local and global resource discovery.

Although the focus of our work and this paper is on computational grids, it is expected that future grid systems will provide access to a far wider range of services, varying from e.g. video-on-demand, context and position aware services to mobile entertainment, e-business and collaboration services. Users of these systems will require instantaneous results for search operations involving millions of services using incomplete information.

The overlay approach makes this fast search possible and effective. The system can scale with size without performance degradation and change in the underlying software architecture. For increased availability and reliability, replicated broker nodes can be added transparently.

Although in this current prototype we used a pure Java/Jini approach, our proposed system can be opened up to a wide range of service information data sources. With appropriate adaptors, the overlay network could read in information from alternative (LDAP, UDDI, etc.) directory services, creating a universal wide-area discovery architecture for all.

REFERENCES

[1] Federale Project. `http://federale.jini.org/`, 2003.

[2] Universal Description, Discovery and Integration (UDDI) Project. `http://www.uddi.org`, 2003.

[3] W. Adije-Winoto, E. Schwartx, H. Balakrishnan, and J. Lilley. The Design and Implementation of and Intentional Naming System. In *Proceedings 17th ACM Symposium on Operating System Principles*, volume 34, pages 186–201, 1999.

[4] G. Allen, G. Aloisio, Z. Balaton, M. Cafaro, T. Dramlitsch, J. Gehring, T. Goodale, P. Kacsuk, A. Keller, H.-P. Kersken, T. Kielmann, H. Knipp, G. Lanfermann, B. Ludwiczak, L. Matyska, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, A. Reinefeld, M. Ruda, F. Schintke, E. Seidel, A. Streit, F. Szalai, K. Verstoep, and W. Ziegler. Early Experiences with the EGrid Testbed. In *IEEE/ACM International Symposium on Cluster Computing and the Grid CCGrid2001*, pages 130–127, May 2001.

[5] E. A. Brewer. When everything is searchable. *Communications of the ACM*, 44(3):53–54, Mar. 2001.

[6] Y. Chawathe, S. A. Fink, S. McCanne, and E. A. Brewer. A Proxy Architecture for Reliable Multicast in Heterogeneous Environments. In *Proceedings of the 6th ACM International Conference on Multimedia (Multimedia-98)*, pages 151–160, N.Y., Sept. 12–16 1998. ACM Press.

[7] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for internet content distribution as an infrastructure service, 2000.

[8] S. Czerwinski, B. Y. Zhao, T. Hodes, A. D. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom-99*, pages 24–35, N.Y., Aug. 15–20 1999. ACM Press.

[9] DANTE. GEANT-MULTICAST. `http://www.dante.net/nep/GEANT-MULTICAST/`, 2002.

[10] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, Aug. 1997.

[11] E. Guttman. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[12] S. Helders. World Gazetteer. `http://www.world-gazetteer.com`, 2003.

[13] W. Hoschek. The Web Service Discovery Architecture. In *SC'2002 Conference CD*, Baltimore, MD, Nov. 2002. IEEE/ACM SIGARCH. pap161.

[14] A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. *Lecture Notes in Computer Science*, 2242:51–??, 2001.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI-00)*, pages 197–212, Berkeley, CA, Oct. 23–25 2000. The USENIX Association.

[16] Z. Juhasz, A. Andics, and S. Pota. JM: A Jini Framework for Global Computing. In *Proceedings 2nd International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, IEEE International Symposium on Cluster Computing and the Grid (CCGrid'2002)*, pages 395–400, Berlin, Germany, May 21 - 24 2002.

[17] S. Marti and V. Krishnan. Carmen: A Dynamic Service Discovery Architecture. Technical Report HPL-2002-257, Hewlett Packard Laboratories, Sept.23 2002.

[18] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, Hewlett Packard Laboratories, Mar.15 2002.

[19] B. Plale, P. Dinda, and G. von Laszewski. Key Concepts and Services of a Grid Information Service. In *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002)*, `http://www.cs.northwestern.edu/ urgis/`, 2002.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.

[21] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-Level Multicast Using Content-Addressable Networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001.

[22] The Salutation Consortium. `http://www.salutation.org/`, 2002.

[23] Universal Plug and Play Forum. `http://www.upnp.org`, 1999.

[24] M. van Steen, F. Hauck, P. Homburg, and A. Tanenbaum. Locating objects in wide-area systems. *IEEE Communication*, 36(1):104–109, 1998.

[25] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3), Dec. 1997. Status: PROPOSED STANDARD.

[26] J. Waldo and K. Arnold. *The Jini specifications*. Jini technology series. Addison-Wesley, Reading, MA, USA, second edition, 2001. Rev. ed of: The Jini specification / Ken Arnold . . . [et al]. c1999.

Edited by: ¡names¿
Received: March 31, 2003
Accepted: June 5, 2003