QoS-Aware Discovery of Wide-Area Distributed Services *

Dongyan Xu, Klara Nahrstedt, Duangdao Wichadakul
Department of Computer Science
University of Illinois at Urbana-Champaign
{*d-xu, klara, wichadak*}*@cs.uiuc.edu*

Abstract

The global computational grids bring together distributed computation/communication resources. Beyond this, we envision the emergence of global 'service grids', which provide a 'market' of application-level distributed services for clients to discover and to request. In this paper, we study the issue of wide-area service discovery in service grids. We start with an existing basic wide-area service discovery framework. The framework adopts a scalable architecture consisting of a hierarchy of Discovery Servers. We then identify problems with the basic framework, and propose our enhancement of query responsiveness *and* QoS awareness*. The key techniques we introduce include: (1) the addition of QoS feedback capability to clients; and (2) the caching and propagation of discovery results with QoS feedbacks in the discovery server hierarchy. With these techniques, the enhanced service discovery framework will be faster in finding qualified service providers. Furthermore, it will select a 'good' (with respect to the QoS to be delivered) service provider for each querying client, based on QoS feedbacks.*

1 Introduction

The emergence of global computational grids brings changes to the traditional paradigm of distributed computing. It is now possible that one computation task is executed by autonomous, distributed, and heterogeneous computing and communication resources, virtually brought together by the grids. More importantly, a computation or communication resource can be *discovered* on-demand in the grids. If multiple 'instances' of the resource (for example, CPU or storage space) are found, one of them will be selected dynamically based on the their current load and capacity.

Taking one step further, we envision the emergence of global *service grids* on top of the computational grids. The grids will not just provide raw computation/communication resources. They will appear as a 'market' of *application-level distributed services*. Examples of such services include virtual scientific laboratory services, virtual community services, E-commerce, and multimedia data distribution, processing, or storage services. These services will be deployed in the wide-area service grids, and can be discovered and requested by any networked clients.

One challenge in such a wide-area service grid is to build a scalable and efficient service discovery framework. The main requirements for such a framework include: (1) The distribution and management of service information should not incur excessive overhead and create performance bottleneck, with the rapid increase of available services and querying clients; (2) Response to a service query should be reasonably fast. In particular, the query response time should not always depend on how far away a service provider is from the querying client[1]; and (3) The discovery result returned to a client should be likely to bring good service quality to the service session that follows, if there exist multiple service providers of the same service.

In this paper, we propose our solution to the challenge which meets the above requirements. It is based on an existing basic wide-area service discovery framework. The framework adopts a scalable architecture consisting of a *hierarchy* of *Discovery Servers*. Furthermore, we identify problems with the basic framework, and propose our enhancement of *query responsiveness* and *QoS awareness* to the framework. The key techniques we introduce include: (1) the addition of QoS feedback capability to clients; and

* Accepted to the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), Brisbane, Australia, May 2001. This work was supported by the National Science Foundation under contract number 9870736, the Air Force Grant under contract number F30602-97-2-0121, National Science Foundation Career Grant under contract number NSF CCR 96-23867, NSF PACI grant under contract number NSF PACI 1 1 13006, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, NSF CISE Infrastructure grant under contract number NSF CDA 96-24396, and NASA grant under contract number NASA NAG 2-1250.

[1] Instead, in our framework, query response time will depend more on the request frequency of a service.

(2) the caching and propagation of discovery results with QoS feedbacks in the Discovery Server hierarchy. With these techniques, the enhanced framework will be faster in finding qualified service providers. Furthermore, it will select a 'good' (with respect to the QoS to be delivered) service provider for each querying client, based on QoS feedbacks.

The rest of the paper is organized as follows: Section 2 presents a basic wide-area service discovery framework and identifies its problems. This is the basis on which we propose our query responsiveness and QoS awareness enhancement. Section 3 describes the enhancement to the basic framework in detail. Section 4 shows the performance of the enhanced wide-area service discovery framework, using our initial simulation results. Section 5 discusses related work and justifies the novelty of our work. Finally, Section 6 concludes this paper.

2 A Wide-Area Service Discovery Framework

We first describe the basic wide-area service discovery framework. Then we will identify some problems with the basic framework, which motivate our work. The basic framework adopts a *hierarchy* architecture, which has been recognized as a scalable approach to wide-area service directory organization and management [2, 10, 11, 7].

2.1 Basic Service Discovery Architecture

The architecture of the basic service discovery framework is shown in Figure 1. The wide-area environment is divided into *domains* with different ranges. More specifically, the domain partition is hierarchical, i.e. one domain consisting of multiple domains at the next lower level. The key entities in the architecture include the service clients, the Service Providers (SPs), and the Discovery Servers (DSes).

- **Service clients** query the service discovery framework for the providers of their requested services. A client only interacts with its home Discovery Server (to be defined shortly), i.e. the Discovery Server of the lowest level domain where the client resides.
- **Service Providers (SPs)** provide application-level distributed services. To advertise its service, each SP sends *service ad* to the service discovery framework. Similar to a client, an SP only interacts with its home Discovery Server.
- **Discovery Servers (DSes)** are key entities in the architecture. They act as *brokers* between service clients and SPs. There is a DS in each domain. Due to the hierarchical partition of domains, the DSes are also organized into a hierarchy. In particular, each of the DSes at the lowest level is called the *home DS* by the hosts in that lowest level domain. To hide complexity, an SP (a client) sends service ad (service query) only to its home DS. However, to perform wide-area service discovery, there will be exchange of service queries and service ads between the DSes.

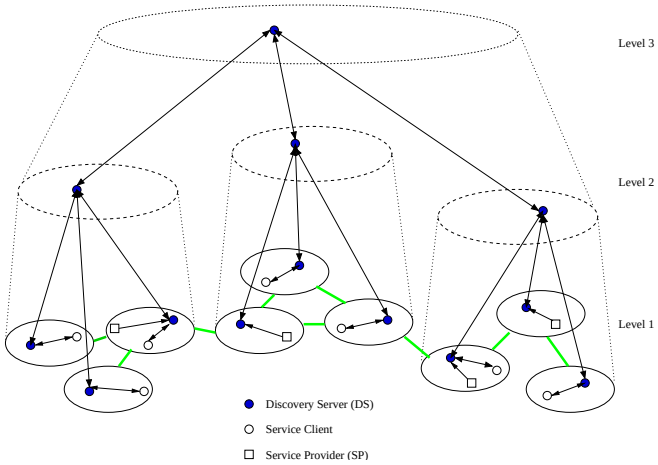

Figure 1. Architecture of the Basic Service Discovery Framework

2.2 Basic Service Discovery Mechanism

To describe the basic service discovery mechanism, we use the Berkeley Service Discovery Service (SDS) [2] as a reference. However, we omit the security feature in SDS for simplicity. There are two key issues in the mechanism: (1) *lossy aggregation of service ads* as they are propagated up in the DS hierarchy, and (2) *distributed service query processing (routing)* in the DS hierarchy.

- **Lossy aggregation of service ads** involves the summarization of local service ads by each home DS before sending them up to its parent DS. The parent DS stores the *summarized* service ads from all its children DSes. In turn, it further summarizes the summarized service ads before sending them up to its parent DS... This lossy aggregation is performed recursively in the DS hierarchy, in order to prevent higher level DSes from being overloaded by the otherwise full service ad propagation traffic. In SDS, the aggregation technique is hashing and hash summarization via Bloom filtering [2].
- **Distributed service query processing (routing)** involves the forwarding of a service query in the DS hierarchy, in order to find the DSes which store the

full service ad of a qualified SP. Figure 2 shows an example of service query processing. Starting from the home DS to which a query is submitted (step 1), if no service ad (or summarized service ad - if it is a higher level DS) matches the query, the DS will forward the query *up* to its parent DS, and so on (steps 2, 3). If there is a match, the DS will forward the query *downward* to the children DS(es) which originally sent up this qualified summarized service ad (steps 4, 5). From this point, the query may be coming down along multiple 'paths' in the hierarchy, because there may exist multiple qualified service ads. Each DS on such a path verifies if the query has a real match with the corresponding (and more detailed) service ad maintained by the DS. If not, the forwarding stops (step 5.b). Otherwise, the query is forwarded further downward (step 5.a). Finally, all home DSes which have the full and qualified service ads will return these ads to the original DS (step 6). The DS will then return the discovery results to the querying client (step 7).

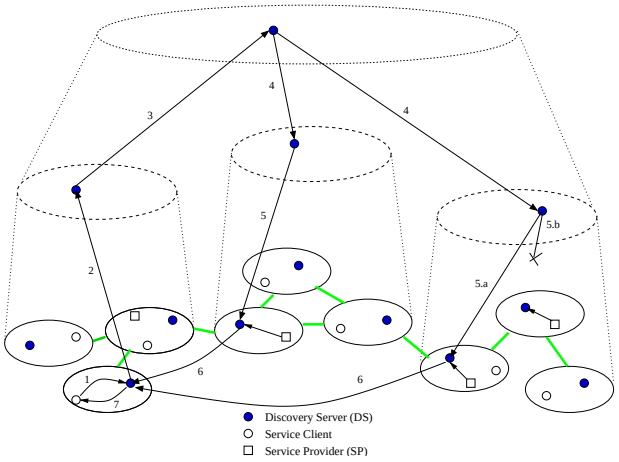

Figure 2. Service Query Processing (Routing) in the Basic Service Discovery Framework

2.3 Problems in the Framework

The basic service discovery framework provides a scalable solution to wide-area service information distribution and management. However, there exist two problems that still need to be solved.

- **Potentially long query response time** In the basic service discovery mechanism, it is clear that the query response time grows with the 'distance' between a querying client and a qualified SP. The distance is measured by the number of DSes the query has to go through (both up and down) in the hierarchy. Moreover, since higher level DSes oversee wider-range domains, they may easily be overloaded by the inter-DS service query traffic. This in turn makes the query response time even longer.

- **QoS unawareness** There is no explicit consideration on the QoS to be delivered by a discovered SP. In particular, if a query leads to multiple discovery results, there is little information the framework could use to predict which one is likely to provide good QoS to the querying client.

These problems motivate our work in bringing both query responsiveness and QoS awareness to the basic framework.

3 Enhancement to the Service Discovery Framework

In this section, we present our enhancement of query responsiveness and QoS awareness to the basic wide-area service discovery framework. The enhancement involves modifications to service clients and to Discovery Servers (DSes).

3.1 Enhancement to Service Clients

The enhancement to service clients is the addition of *QoS feedback* capability. During a service session, the client-side software component will monitor the QoS during the session, and generate a numerical average QoS level observed by the client. The definition of QoS levels are highly service-specific. However, in order to be understood by the DSes which are service-independent, the QoS level definition should conform to the following simple rule: the higher the QoS level, the better the QoS observed. Our experience with a *Video-on-Demand* service [12] and a *Distributed Visual Tracking* service [5], which have very different definitions of QoS levels, demonstrates the feasibility of implementing client-side software components in this fashion.

Each client will then send the QoS feedback to its home DS. This forms a feedback loop between the service discovery framework and the clients.

3.2 Enhancement to Discovery Servers

Our major contribution lies in the enhancement to the DSes. We propose *the caching and propagation of service ads with QoS feedbacks in the DS hierarchy*. More specifically, a DS can maintain a cache of discovery results - the service ads returned by the home DSes of their origin. Furthermore, each cached service ad will be associated with QoS-feedbacks from clients of this service in the domain

of the caching DS. The caching DS will then leverage these feedbacks to respond to service queries from the same domain, and to select a 'good' SP (Service Provider) with respect to potential service quality for each query. Therefore, these upcoming queries for the same service will experience shorter query response time, and incur less query processing overhead. More importantly, the discovery result is more likely to achieve satisfactory QoS in the service session to follow.

By this approach, we try to achieve three goals: (1) to lower the average query response time; (2) to improve the session QoS following a query; and (3) (which is less addressed,) to make this cached service ad helpful to queries of as many clients as possible. However, the three goals are not orthogonal. To achieve (1) and (2), the service ad should be cached in a low level DS (for example, a home DS) close to a querying client, but that will limit the benefiting clients only to those in the same small domain. On the other hand, if the service ad is cached in a high level DS visible to a wider range of clients, the query response time will become longer. Moreover, the resultant session QoS may deviate from what is predicted by the caching DS. The reason is: the wider the range a DS oversees, the more diversifying (and less useful for QoS prediction) QoS feedbacks it tends to receive.

To achieve these goals and alleviate the conflicts between them in the meantime, we introduce three strategies in the following subsections.

3.2.1 First Strategy: Hierarchical Caching and Propagation of Service Ads

The first strategy is hierarchical caching and propagation of service ads. Hierarchical caching has been used in cooperative web content caching [1]. Interestingly, in service discovery, we also find it a very useful technique. It can be performed together with the normal service query processing (described in Section 2.2). Even better, since service ads are of uniform and much smaller size, the disadvantage of high latency and bandwidth consumption in web content access becomes much less significant. The operations of hierarchical service ad caching and propagation are as follows (Figure 3):

- **Service ad caching and propagation** A service query leads to a service session. After the service session completes, the client will generate a QoS feedback about the completed session, and return it to the service discovery system (step 1). The service ad with QoS feedback will then be propagated up the DS hierarchy (step 2): at each level starting from the client's home DS, the DS either caches this service ad with QoS feedback - if it has not yet done so; or update the associated access frequency and QoS feedback - if the service ad is already cached. This upward propagation will terminate at the DS which originally found this service ad in its cache; or, if the service ad was found in the home DS of the corresponding SP, the propagation will terminate at the DS of the highest-level QoS-similar domain (to be defined shortly). The replacement policy for the service ad cache is based on the items' access frequency. In addition, when a DS deletes a cached service ad, it also moves it one level up to its parent DS.

- **Service query processing** A client's service query (step 3) is forwarded up the DS hierarchy (step 4). At each level, besides normal query processing (i.e. checking the query against stored service ad summaries), the DS will also check if there are cached service ads that satisfy this query. If a cache hit occurs, the DS will select one of the qualified cached service ads, based on the QoS feedbacks associated with each of them, and return it to the querying client (step 5.a); otherwise, the normal query processing will continue (step 5.b). For the QoS-feedback-based selection, there exist different policies - for example: always selecting the one with the highest QoS feedback value; or randomly selecting one of those whose QoS feedback values are beyond some threshold. [8] provides a comparative study on the effectiveness of different selection policies.

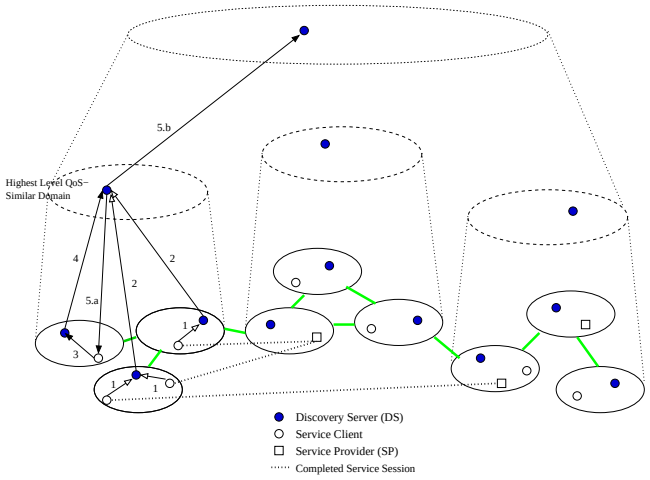

Figure 3. Operations of Hierarchical Service Ad Caching and Propagation

The new query processing operations can be easily extended from the basic service query processing operations. Therefore, the additional overhead introduced is not significant, comparing with the performance gain in query response time. More importantly, the propagation of service

ad with QoS feedback is *self-adaptive* with respect to the placement of cached service ads in the DS hierarchy. Via the operation, a cached service ad is able to move 'up' and 'down' dynamically in the DS hierarchy, based on its current query frequency. If clients in a domain request a certain service frequently, then its service ad will soon settle down at a DS close to these clients. Copies of the same service ad at higher level DSes will gradually 'evaporate', when their access frequency decrease. On the other hand, for a less frequently requested service, its service ad tends to stay in a higher-level DS, where the query frequency will again build up - from a wider range of clients. Existence of the same service ad at lower level DSes tend to be transient, due to the low query frequency.

However, in order to achieve good QoS prediction for service queries, the propagation of service ad with QoS feedback should be kept within certain levels of DSes. For this reason, we propose the concept of *QoS-similar domain*. A QoS-similar domain is a domain in which a majority of the clients tend to observe similar QoS provided by the same SP. For example, domains at the lowest level are usually QoS-similar domains, with relatively homogeneous network connection for each client in the domain. Upper level domains may also be QoS-similar. However, the higher the domain level, the less the similarity it tends to exhibit.

There are two possible ways to determine if a domain is QoS-similar. First, it may be manually defined by the domain administrator. Second, in a more automated approach, DSes at different levels may periodically check if their domains are QoS-similar. This is done by checking the degree of *similarity* among QoS-feedbacks at each DS, with respect to some public 'benchmark' services. A home DS checks QoS feedbacks from the clients; while a higher-level DS checks the aggregated QoS feedbacks, i.e. the *average* QoS level reported by its children DSes. The QoS-predictability of a domain tends to change only at a large time scale - for example, in the magnitude of hours or days. Therefore, the 'similarity check' does not have to be performed frequently, and will not incur significant runtime overhead.

With the concept of QoS-similar domains, the propagation operations should only be performed between DSes of QoS-similar domains. As shown in Figure 3, service ads with QoS feedbacks are only propagated between the DSes ranging from a home DS to the DS of the highest-level QoS-similar domain. Beyond the highest-level QoS-similar domain, the QoS feedbacks become too diversified to predict future service quality for the clients in such a wide range domain.

3.2.2 Second Strategy: End-to-End Propagation of Service Ads with QoS Feedbacks

We can further propagate service ads with QoS feedbacks, so that even more clients can benefit from them. We propose the second strategy: *end-to-end* propagation of service ads with QoS feedbacks. The main difference between the end-to-end propagation and the propagation described in Section 3.2.1 is the propagation direction: the latter is *vertical* - between DSes at different levels; while the former is *horizontal* - between DSes at the same level.

More specifically, if a client reports *good* QoS about an SP to its home DS, the home DS will start the end-to-end propagation, in addition to the hierarchical service ad caching and propagation. The DS sends the corresponding service ad with QoS feedback *backward* to the home DSes along the path leading to the SP. The intuition is: if clients in a domain D observe good QoS provided by an SP, then it is very likely that clients in each *intermediate* domain between D and the home domain of the SP will also receive good QoS provided by the SP.

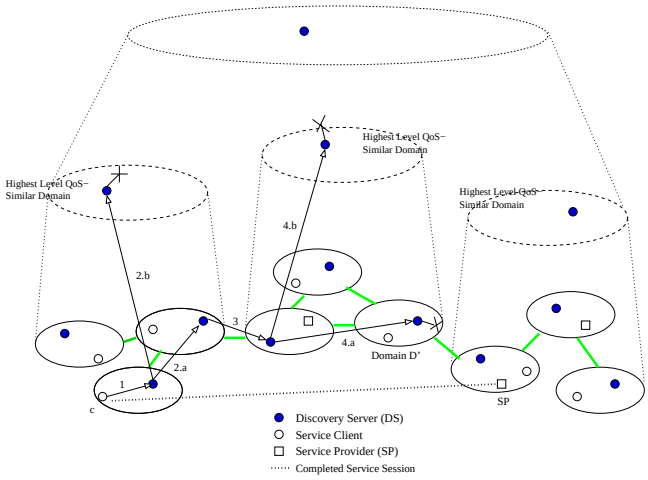

Figure 4. Operations of End-to-End Service Ad Propagation (the Hierarchical Propagation Triggered Is Also Shown

The operations of end-to-end service ad propagation is shown in Figure 4. The service ad with QoS feedback is propagated from the home DS of client c toward the home DS of service provider SP (steps 1, 2.a, 3, 4.a). However, to avoid redundancy, the propagation will stop if an intermediate DS (for example, the DS of domain D' in Figure 4) already caches this service ad. In addition, notice that the end-to-end propagation also triggers a series of hierarchical propagations: each DS at the lowest level will also propagate the service ad with QoS feedback *upward* in the hierarchy (steps 2.b, 4.b). The hierarchical propagation will stop either at a higher-level DS which already caches

this service ad, or at the DS of the highest-level QoS-similar domain.

With both hierarchical (vertical) and end-to-end (horizontal) propagations, service ads with QoS feedbacks will quickly spread through a wide range of domains. Comparing with a hierarchical-propagation-only approach, this combined approach will benefit a larger range of clients. The benefits include both a faster discovery of an SP, and a better QoS in the service session to follow.

One remaining problem is how to identify the 'next-hop' DS on the path from a DS to an SP. One possibility is to leverage the underlying network routing information, as well as the Domain Name System (DNS). First, we propose a new *Resource Record (RR)* type 'DS' in the DNS system. A DNS lookup for a host h with the new RR type will return the address of host h's home DS. This is similar to other RR types such as *MX*, which returns the address of host h's mail exchanger. Then, to determine the 'next-hop' DS on the path toward an SP, the current DS (1) calls a *traceroute*-like routine to find the next router r on the path [9]; (2) performs a DNS lookup for r using the 'DS' RR type; and (3) if the DNS lookup returns a DS different from the current DS, then it is the 'next-hop' DS to propagate to; otherwise the current DS goes back to step (1), with the *TTL (Time-To-Live)* value [9] incremented by one for the *traceroute*-like routine.

3.2.3 Third Strategy: QoS Feedback Probing

The first and second strategies are *push-based*, i.e. the service ads with QoS feedbacks are propagated proactively to DSes for future queries. In this section, we propose a *pull-based* strategy. The strategy is useful when a service query does not experience any service ad cache hit while it is forwarded up in the hierarchy; and the query finally results in multiple discovery results returned by home DSes of the qualified SPs (via the basic query processing steps in Section 2.2). These results are without QoS feedbacks. Therefore, the DS which originally accepts the query from a client may have to make a random selection. The penalty is the potentially unsatisfactory QoS for the querying client.

We propose the strategy of QoS feedback probing, which provides a degree of QoS predictability at the price of some additional latency and overhead. As shown in Figure 5, after a DS has received the discovery results (without QoS feedbacks) for a service query, the DS will send out a *QoS feedback probing message* for each service ad received. Similar to the end-to-end service ad propagation, each probing message will be propagated to the intermediate DSes toward the SP designated by the corresponding service ad (steps 1, 2). If an intermediate DS does not cache the service ad of the SP, the probing message will first be forwarded up in the hierarchy (step 3), until it reaches the highest level QoS-similar domain; and then be forwarded again toward the SP (step 4)... If one of the DS traversed caches the service ad of the SP, it will return the associated QoS feedback to the original probing DS (step 5).

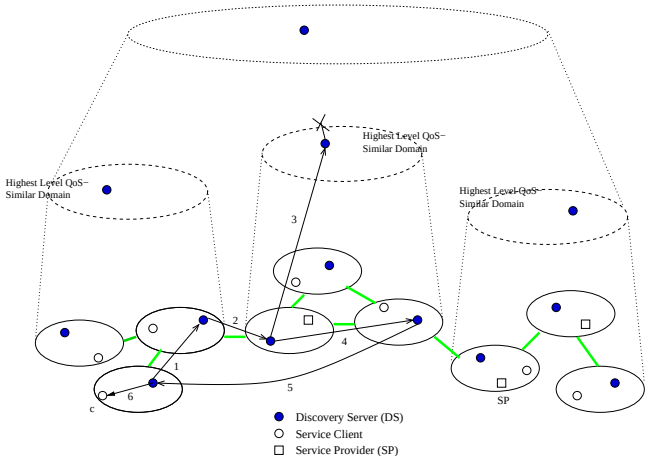

Figure 5. Operations of QoS Feedback Probing (Only Showing the Probing for *One* of the Discovery Results)

The probing DS will then use the collected QoS feedbacks to select one of the candidate SPs, which is likely to bring good QoS for the querying client (step 6). However, comparing with the first and the second strategies, the strategy of QoS feedback probing has more limitations. First, it is less predictive: the returned QoS feedbacks may wrongly indicate the session QoS for clients in the current domain. This is due to a lack of *justification for QoS similarity* between the current domain and the domain where the QoS feedback is found. Second, The probing overhead and latency is non-trivial. For this reason, we may limit the number of DSes to traverse during the QoS feedback probing; and the original probing DS only selects from the SPs whose QoS feedbacks are found within that limit.

4 Simulation Results

In this section, we present our initial simulation results. We show the improvement in both query responsiveness and session QoS achieved by the enhanced service discovery framework. The wide area environment we simulate is shown in Figure 6. The domains are partitioned into a four-level hierarchy. The corresponding DS hierarchy is also shown in the Figure.

We assume that there are 20 different services in the simulated environment. For each service S_i, we deployed n_i replicated SPs for S_i, and n_i is randomly distributed in

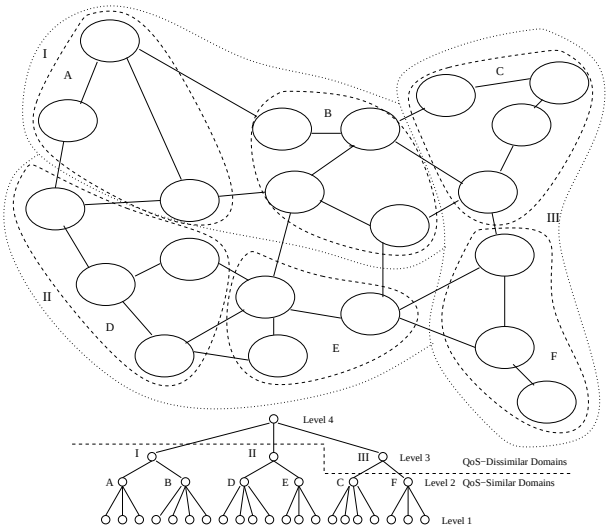

Figure 6. Simulated Wide-Area Environment

$[1, 3]$. We make sure that for each level-3 domain (with a dotted-line border), there is no more than one SP for each service. The popularity of each service, measured in number of requests per minute, is randomly distributed in $[5, 60]$ requests per minute. For each service query, the client that makes the query is randomly selected from one of the level-1 domains. For each service session that follows a query, the duration of the session is randomly distributed in $[1, 10]$ minutes. We also assume that each DS has a cache with the capacity to store the service ads for five different services (note that for each service, there may be multiple service ads for the replicated SPs). Finally, the highest level QoS-similar domains are level-2 domains (with dashed-line borders) C and F, as well as level-3 domains I and II.

Service query responsiveness: We simulate the above scenario for a period of 4000 minutes. As a comparison, we also simulate the same scenario under the *basic* service discovery framework. For each service query, the number of DSes involved in the query processing is recorded. We use the number of DSes involved in a query as an indication of query responsiveness. Table 1 shows the average numbers of DSes involved among queries from each of the level-2 domains, under the basic (row 1) and the enhanced (row 2) service discovery frameworks. The results show a significant improvement in query responsiveness under the enhanced framework.

Session QoS: To demonstrate improved session QoS, our simulation focuses on one of the services. We simulate the following scenario in Figure 7: service S has two replicated SPs: SP_X in domain D and SP_Y in domain C. The average request rate for S is 60 requests per minute. Each request is made by a client from a randomly chosen domain. Each session of service S will require one unit of local resource on the SP and one unit of end-to-end network bandwidth between the SP and the client. The total amount of each resource is randomly distributed between 50 and 300 units. Here, a resource can be the local resource on SP_X or SP_Y, or the bandwidth of any backbone network segment shown in Figure 7. For simplicity, we assume that service S only has two QoS levels - '1' and '0'. A service session results in QoS level '1', if and only if both its local and network resource requirements are satisfied throughout the session; otherwise, the session results in QoS level '0'.

Framework/Domain	A	B	C	D	E	F
Basic	4.2	3.9	4.4	4.0	4.8	4.5
Enhanced	1.6	2.1	1.9	1.7	1.9	2.0

Table 1. Average Number of DSes Involved in a Query - in Each Level-2 Domain and under the Basic and Enhanced Framework, Respectively

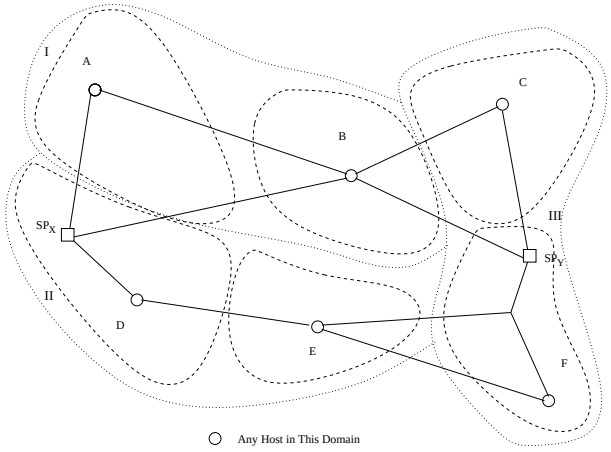

Figure 7. Simulated Scenario to Show Better Session QoS Achieved by the Enhanced Framework

We again compare the basic and the enhanced service discovery frameworks. The former is QoS-unaware; while the latter is with QoS feedback capability. Therefore, between the two replicated SPs, a DS in the basic framework just *randomly* selects one to return to each querying client; while a DS in the enhanced framework uses QoS feedbacks (in the form of average QoS levels) to select an SP. In our simulation, we adopt the following policy in selecting an SP: select SP_X with probability $Q_X/(Q_X + Q_Y)$; while select SP_Y with probability $Q_Y/(Q_X + Q_Y)$. Q_X (Q_Y) is the QoS feedback associated with the service ad for SP_X (SP_Y).

Table 2 shows the average QoS level of service sessions observed by clients in each level-2 domain, under the basic and enhanced service discovery framework, respectively. We notice that in every domain, the average session QoS achieved by the enhanced framework is better than the QoS achieved by the basic framework.

	A	B	C	D	E	F
Basic	0.72	0.71	0.73	0.73	0.71	0.70
Enhanced	0.83	0.84	0.85	0.84	0.85	0.85

Table 2. Average QoS Level of Service Sessions - in Each Level-2 Domain and under the Basic and Enhanced Framework, Respectively

Although it is still simple and initial, our simulation does show the effectiveness of the proposed enhancement to the basic service discovery framework. More complex and detailed performance analysis is our on-going work.

5 Related Work

Service discovery in a networked environment has been an active research topic in recent years. The IETF Service Location Protocol (SLP) [3] defines the basic architecture for all brokerage-based service discovery frameworks. The Discovery Agent (DA) in SLP correspond to the Discovery Server (DS) in our framework. In addition, using SLP glossary, we implicitly assume that in our framework, there is a User Agent (UA) running on each client, and there is a Service Agent (SA) running on each service provider (SP). However, SLP was originally designed for service discovery within one administrative domain, such as an intranet, rather than in a wide-area grid environment. There have been wide-area extensions to SLP [6, 2]. However, none of them address the issues of service ad caching and QoS feedback.

The hierarchy has been recognized as an efficient and scalable architecture for wide-area information distribution and discovery [2, 10, 11, 7]. We regard the Berkeley SDS [2] as our 'base' service discovery framework. Other frameworks [10, 11] also adopt a similar lossy aggregation approach to control higher-level discovery server load and bandwidth consumption due to service ad propagation. Again, none of these framework address the issues of QoS awareness and service ad caching in the hierarchy. On the other hand, hierarchical caching has been introduced for web content caching [1]. Interestingly, we find it also an effective technique in service ad caching; and even better, some of its disadvantages disappear in this context.

Application-level anycast [8, 13] is also related to QoS-aware service discovery. It involves the selection of replicated SPs based on their current service quality or capacity. However, existing application-level anycast systems [8, 13] focus more on the *selection* policy and mechanism, rather than on the more active *discovery* capability. As a result, the QoS feedbacks will only benefit clients within the same domain. On the contrary, our enhanced service discovery framework further propagates the service ads with QoS feedbacks - both vertically and horizontally, so that a wider range of clients can benefit in both query responsiveness and session QoS.

Finally, a network-level anycast scheme is proposed in [4]. It is more discovery-oriented comparing with its application-level counterparts. However, the layer it works at intrinsically limits the effectiveness in bringing good end-to-end application-level QoS to clients.

6 Conclusion

In this paper, we study the issue of wide-area service discovery in emerging service grids. Based on an existing wide-area service discovery framework, we propose our enhancement of service query responsiveness and QoS awareness. On the client side, we propose the generation of QoS feedbacks about completed service sessions; on the Discovery Server side, we propose the caching and propagation of service ads with QoS feedbacks. The enhanced service discovery framework is scalable with respect to the numbers of service providers, discovery servers, and service queries. Furthermore, it achieves shorter query response time, lower processing overhead, and better session QoS.

Current service discovery systems focus on the distribution and matching of service information and service queries, with less emphasis on domain-dependent session QoS. On the other hand, current application-level anycast systems focus on QoS-aware selection of replicated service providers, whose locations are assumed to be known. Our work addresses *both* problems, and provides an integrated solution.

References

[1] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarchical Internet Object Cache. *Proceedings of USENIX Annual Technical Conference*, 1996.

[2] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. *Proceedings of ACM MobiCom'99*, Sept. 1999.

[3] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. *IETF RFC-2165*, Nov. 1998.

[4] D. Katabi and J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). *Proceedings of ACM SIGCOMM 2000*, Aug. 2000.

[5] B. Li, W. Jeon, W. Kalter, K. Nahrstedt, and J. Seo. Adaptive Middleware Architecture for a Distributed Omni-Directional Visual Tracking System. *Proceedings of SPIE/ACM MMCN 2000*, Jan. 2000.

[6] J. Rosenberg and H. Schulzrinne. Internet Telephony Gateway Location. *Proceedings of IEEE Infocom '98*, Mar. 1998.

[7] M. Sheldon, A. Duda, R. Weiss, and D. Gifford. Discover: a Resource Discovery System Based on Content Routing. *Computer Networks and ISDN Systems*, pages 953–972, 1995.

[8] M. Stemm, S. Seshan, and R. Katz. A Network Measurement Architecture for Adaptive Applications. *Proceedings of IEEE Infocom 2000*, Mar. 2000.

[9] R. Stevens. *Unix Network Programming, Prentice Hall*, 1:672–685, 1998.

[10] N. Sturtevant, N. Tang, and L. Zhang. The Information Discovery Graph: Towards a Scalable Multimedia Resource Directory. *Proceedings of IEEE Workshop on Internet Applications '99*, Aug. 1999.

[11] R. van Renesse. Scalable and Secure Resource Location. *Proceedings of IEEE Hawaii International Conference on System Sciences*, Jan. 2000.

[12] D. Xu, D. Wichadakul, and K. Nahrstedt. Resource-Aware Confi guration of Ubiquitous Multimedia Service. *Proceedings of IEEE ICME 2000*, Aug. 2000.

[13] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-Level Anycasting: a Server Selection Architecture and Use in a Replicated Web Service. *IEEE/ACM Transactions on Networking*, Aug. 2000.