

Scalable Grid Service Discovery Based on UDDI*

Sujata Banerjee[§], Sujoy Basu[§], Shishir Garg[€], Sukesh Garg[€], Sung-Ju Lee[§], Pramila Mullan[€], Puneet Sharma[§]

[§]HP Labs
1501 Page Mill Road
Palo Alto, CA, 94304 USA
+1-650-857-2137

[€]France Telecom R&D Division
801 Gateway Blvd, # 500
South San Francisco, CA, 94080 USA
+1 650 -875-1500

{sujata.banerjee,sujoy.basu,sungju.lee,puneet.sharma}@hp.com

{shishir.garg,sukesh.garg,pramila.mullan}@francetelecom.com

ABSTRACT

Efficient discovery of grid services is essential for the success of grid computing. The standardization of grids based on web services has resulted in the need for scalable web service discovery mechanisms to be deployed in grids. Even though UDDI has been the de facto industry standard for web-services discovery, imposed requirements of tight-replication among registries and lack of autonomous control has severely hindered its widespread deployment and usage. With the advent of grid computing the scalability issue of UDDI will become a roadblock that will prevent its deployment in grids. In this paper we present our distributed web-service discovery architecture, called DUDE (Distributed UDDI Deployment Engine). DUDE leverages DHT (Distributed Hash Tables) as a rendezvous mechanism between multiple UDDI registries. DUDE enables consumers to query multiple registries, still at the same time allowing organizations to have autonomous control over their registries. Based on preliminary prototype on PlanetLab, we believe that DUDE architecture can support effective distribution of UDDI registries thereby making UDDI more robust and also addressing its scaling issues. Furthermore, The DUDE architecture for scalable distribution can be applied beyond UDDI to any Grid Service Discovery mechanism.

Categories and Subject Descriptors

C2.4 [Distributed Systems]

General Terms

Design, Experimentation, Standardization.

Keywords

UDDI, DHT, Web services, Grid computing, MDS, discovery.

1. INTRODUCTION

Efficient discovery of grid services is essential for the success of grid computing. The standardization of grids based on web services has resulted in the need for scalable web service

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

**MGC '05, November 28- December 2, 2005 Grenoble, France
Copyright 2005 ACM 1-59593-269- 0/05/11... \$5.00**

discovery mechanisms to be deployed in grids. Grid discovery services provide the ability to monitor and discover resources and services on grids. They provide the ability to query and subscribe to resource/service information. In addition, threshold traps might be required to indicate specific change in existing conditions. The state of the data needs to be maintained in a soft state so that the most recent information is always available. The information gathered needs to be provided to variety of systems for the purpose of either utilizing the grid or providing summary information. However, the fundamental problem is the need to be scalable to handle huge amounts of data from multiple sources.

The web services community has addressed the need for service discovery, before grids were anticipated, via an industry standard called UDDI. However, even though UDDI has been the de facto industry standard for web-services discovery, imposed requirements of tight-replication among registries and lack of autonomous control, among other things has severely hindered its widespread deployment and usage [7]. With the advent of grid computing the scalability issue with UDDI will become a roadblock that will prevent its deployment in grids.

This paper tackles the scalability issue and a way to find services across multiple registries in UDDI by developing a distributed web services discovery architecture. Distributing UDDI functionality can be achieved in multiple ways and perhaps using different distributed computing infrastructure/platforms (e.g., CORBA, DCE, etc.). In this paper we explore how Distributed Hash Table (DHT) technology can be leveraged to develop a scalable distributed web services discovery architecture. A DHT is a peer-to-peer (P2P) distributed system that forms a structured overlay allowing more efficient routing than the underlying network. This crucial design choice is motivated by two factors. The first motivating factor is the inherent simplicity of the put/get abstraction that DHTs provide, which makes it easy to rapidly build applications on top of DHTs. We recognize that having just this abstraction may not suffice for all distributed applications, but for the objective at hand, works very well as will become clear later. Other distributed computing platforms/middleware while providing more functionality have much higher overhead and complexity. The second motivating factor stems from the fact that DHTs are relatively new tool for building distributed applications and we would like to test its potential by applying it to the problem of distributing UDDI.

In the next section, we provide a brief overview of grid information services, UDDI and its limitations, which is followed by an overview of DHTs in Section 3. Section 4 describes our proposed architecture with details on use cases. In Section 5, we

* Authors are listed in alphabetical order.

describe our current implementation, followed by our findings in Section 6. Section 7 discusses the related work in this area and Section 8 contains our concluding remarks.

2. BACKGROUND

2.1 Grid Service Discovery

Grid computing is based on standards which use web services technology. In the architecture presented in [6], the service discovery function is assigned to a specialized Grid service called *Registry*. The implementation of the web service version of the Monitoring and Discovery Service (WS MDS), also known as the MDS4 component of the Globus Toolkit version 4 (GT4), includes such a registry in the form of the Index service Resource and service properties are collected and indexed by this service. Its basic function makes it similar to UDDI registry. To attain scalability, Index services from different Globus containers can register with each other in a hierarchical fashion to aggregate data. This approach for attaining scalability works best in hierarchical Virtual Organizations (VO), and expanding a search to find sufficient number of matches involves traversing the hierarchy. Specifically, this approach is not a good match for systems that try to exploit the convergence of grid and peer-to-peer computing [5].

2.2 UDDI

Beyond grid computing, the problem of service discovery needs to be addressed more generally in the web services community. Again, scalability is a major concern since millions of buyers looking for specific services need to find all the potential sellers of the service who can meet their needs. Although there are different ways of doing this, the web services standards committees address this requirement through a specification called UDDI (Universal Description, Discovery, and Integration). A UDDI registry enables a business to enter three types of information in a UDDI registry – white pages, yellow pages and green pages. UDDI's intent is to function as a registry for services just as the yellow pages is a registry for businesses. Just like in Yellow pages, companies register themselves and their services under different categories. In UDDI, White Pages are a listing of the business entities. Green pages represent the technical information that is necessary to invoke a given service. Thus, by browsing a UDDI registry, a developer should be able to locate a service and a company and find out how to invoke the service.

When UDDI was initially offered, it provided a lot of potential. However, today we find that UDDI has not been widely deployed in the Internet. In fact, the only known uses of UDDI are what are known as private UDDI registries within an enterprise's boundaries. The readers can refer to [7] for a recent article that discusses the shortcomings of UDDI and the properties of an ideal service registry. Improvement of the UDDI standard is continuing in full force and UDDI version 3 (V3) was recently approved as an OASIS Standard. However, UDDI today has issues that have not been addressed, such as scalability and autonomy of individual registries.

UDDI V3 provides larger support for multi-registry environments based on portability of keys. By allowing keys to be re-registered in multiple registries, the ability to link registries in various topologies is effectively enabled. However, no normative description of these topologies is provided in the UDDI

specification at this point. The improvements within UDDI V3 that allow support for multi-registry environments are significant and open the possibility for additional research around how multi-registry environments may be deployed. A recommended deployment scenario proposed by the UDDI V3.0.2 Specification is to use the UDDI Business Registries as root registries, and it is possible to enable this using our solution.

2.3 Distributed Hash Tables

A Distributed Hash Table (DHT) is a peer-to-peer (P2P) distributed system that forms a structured overlay allowing more efficient routing than the underlying network. It maintains a collection of **key-value pairs** on the nodes participating in this graph structure. For our deployment, a key is the hash of a keyword from a service name or description. There will be multiple values for this key, one for each service containing the keyword. Just like any other hash table data structure, it provides a simple interface consisting of put() and get() operations. This has to be done with robustness because of the transient nature of nodes in P2P systems. The value stored in the DHT can be any object or a copy or reference to it. The DHT keys are obtained from a large identifier space. A hash function, such as MD5 or SHA-1, is applied to an object name to obtain its DHT key. Nodes in a DHT are also mapped into the same identifier space by applying the hash function to their identifier, such as IP address and port number, or public key. The identifier space is assigned to the nodes in a distributed and deterministic fashion, so that routing and lookup can be performed efficiently. The nodes of a DHT maintain links to some of the other nodes in the DHT. The pattern of these links is known as the DHT's geometry. For example, in the Bamboo DHT [11], and in the Pastry DHT [8] on which Bamboo is based, nodes maintain links to neighboring nodes and to other distant nodes found in a routing table. The routing table entry at row i and column j , denoted $R_i[j]$, is another node whose identifier matches its own in first i digits, and whose $(i + 1)$ st digit is j . The routing table allows efficient overlay routing. Bamboo, like all DHTs, specifies algorithms to be followed when a node joins the overlay network, or when a node fails or leaves the network. The geometry must be maintained even when this rate is high. To attain consistent routing or lookup, a DHT key must be routed to the node with the numerically closest identifier. For details of how the routing tables are constructed and maintained, the reader is referred to [8, 11].

3. PROPOSED ARCHITECTURE OF DHT BASED UDDI REGISTRY HIERARCHIES

As mentioned earlier, we propose to build a distributed UDDI system on top of a DHT infrastructure. This choice is primarily motivated by the simplicity of the put/get abstraction that DHTs provide, which is powerful enough for the task at hand, especially since we plan to validate our approach with an implementation running on PlanetLab [9]. A secondary motivation is to understand deployment issues with DHT based systems. Several applications have been built as overlays using DHTs, such as distributed file storage, databases, publish-subscribe systems and content distribution networks. In our case, we are building a DHT based overlay network of UDDI registries, where the DHT acts as a rendezvous network that connects multiple registries. In the grid computing scenario, an overlay network of multiple UDDI registries seems to be an interesting alternative to the UDDI public

registries currently maintained by Microsoft, IBM, SAP and NTT. In addition, our aim is to not change any of the UDDI interfaces for clients as well as publishers.

Figure 1 highlights the proposed architecture for the DHT based UDDI Registry framework. UDDI nodes are replicated in a UDDI registry as per the current UDDI standard. However, each local registry has a local proxy registry that mediates between the local UDDI registry and the DHT Service. The DHT service is the glue that connects the Proxy Registries together and facilitates searching across registries.

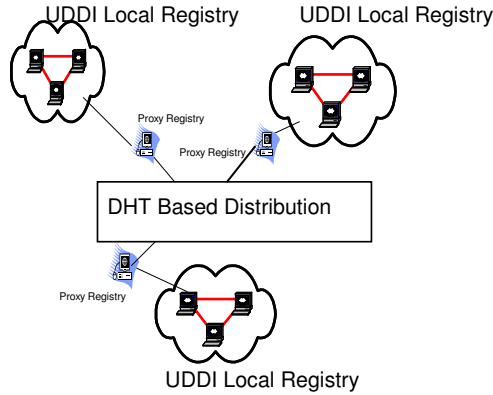


Figure 1: DUDE Architecture

Service information can be dispersed to several UDDI registries to promote scalability. The proxy registry publishes, performs queries and deletes information from the dispersed UDDI registries. However, the scope of the queries is limited to relevant registries. The DHT provides information about the relevant registries. The core idea in the architecture is to populate DHT nodes with the necessary information from the proxies which enables easy and ubiquitous searching when queries are made. When a new service is added to a registry, all potential search terms are hashed by the proxy and used as DHT keys to publish the service in the DHT. The value stored for this service uniquely identifies the service, and includes the URL of a registry and the unique UDDI key of the service in that registry. Similarly when queries arrive, they are parsed and a set of search terms are identified. These search terms are hashed and the values stored with those hash values are retrieved from the DHT. Note that a proxy does not need to know all DHT nodes; it needs to know just one DHT node (this is done as part of the bootstrapping process) and as described in Section 2.3, this DHT node can route the query as necessary to the other nodes on the DHT overlay. We describe three usage scenarios later that deal with adding a new local registry, inserting a new service, and querying for a service.

Furthermore, the DHT optimizes the UDDI query mechanism. This process becomes a lookup using a UDDI unique key rather than a query using a set of search parameters. This key and the URL of the registry are obtained by searching initially in the DHT. The DHT query can return multiple values for matching services, and in each of the matching registries, the proxy performs lookup operations.

The service name is used as a hash for inserting the service information. The service information contains the query URL and

unique UDDI key for the registry containing the service. There could be multiple registries associated with a given service. The service information conforms to the following schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="registries">
    <xs:annotation>
      <xs:documentation>Service Information</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="registry" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name"/>
              <xs:element name="key" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

There can be multiple proxy UDDI registries in this architecture. The advantage of this is to introduce distributed interactions between the UDDI clients and registries. Organization can also decide what information is available from the local registries by implementing policies at the proxy registry.

3.1 Sequence of Operations

In this section, we demonstrate what the sequence of operations should be for three crucial scenarios – adding a new local registry, inserting a new service and querying a service. Other operations like deleting a registry, deleting a service, etc. are similar and for the sake of brevity are omitted here.

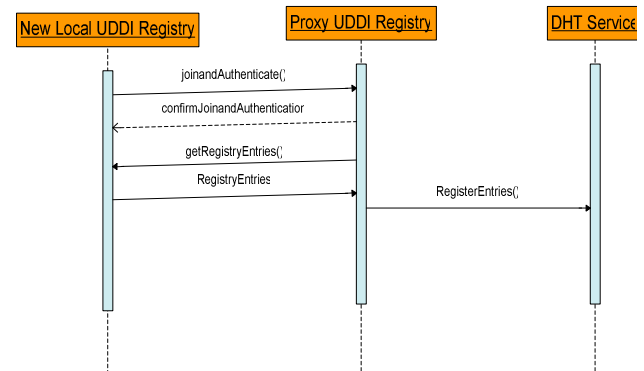


Figure 2: Sequence Diagram- Add New Local Registry

Add a New Local UDDI Registry

Figure 2 contains a sequence diagram illustrating how a new UDDI registry is added to the network of UDDI registries. The new registry registers itself with its proxy registry. The proxy registry in turn queries the new registry for all services that it has

stored in its databases and in turn registers each of those entries with the DHT.

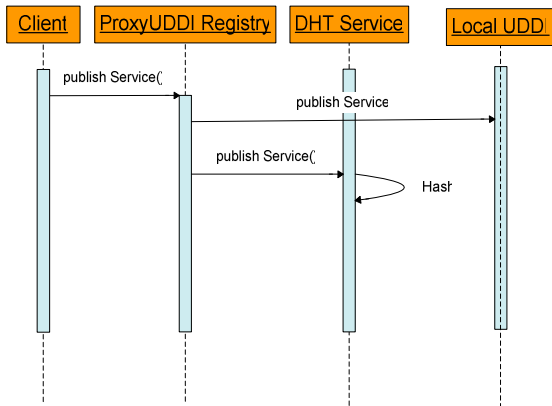


Figure 3: Sequence Diagram – Add New Service

Add a New Service

The use case diagram depicted in **Error! Reference source not found.** highlights how a client publishes a new service to the UDDI registry. In order to interact with the registry a client has to know how to contact its local proxy registry. It then publishes a service with the proxy registry which in turn publishes the service with the local UDDI registry and receives the UDDI key of the registry entry. Then new key-value pairs are published in the DHT, where each key is obtained by hashing a searchable keyword of the service and the value consists of the query URL of the registry and the UDDI key.

Query Service Sequence

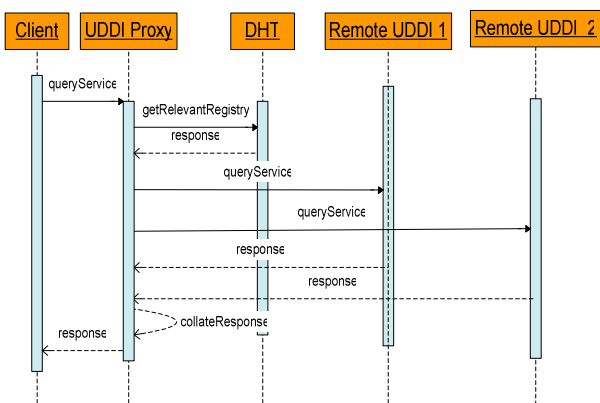


Figure 4: Sequence Diagram – Query for a Service

Query a Service

Figure 4 shows how a client queries the UDDI registry for a service. Once again, the client needs to know how to contact its local proxy registry and invokes the query service request. The proxy registry in turn contacts one of the DHT nodes to determine DHT queries using the search terms.

As explained earlier in the context of Figure 1, multiple values might be retrieved from the DHT. Each value includes the query URL of a registry, and the unique UDDI key of a matching

service in that registry. The proxy then contacts the matching registries and waits for the response of lookup operations using the corresponding UDDI keys. Upon receiving the responses, the proxy registry collates all responses and returns the aggregated set of services to the client.

We will now illustrate these operations using an example. Consider a client contacting its local proxy to publish a service called “Computer Accessories”. The proxy follows the steps in Figure 3 to add the service to UDDI 1 registry, and also publishes two entries in the DHT. The keys of these entries are obtained by hashing the words “computer” and “accessories” respectively. Both entries have the same value consisting of the query URL of this registry and the unique UDDI key returned by the registry for this service. Next we consider another client publishing a service called “Computer Repair” through its proxy to UDDI 2 registry. A similar process results in 2 more entries being added to the DHT. Recall that our DHT deployment can have multiple entries with the same key. If we follow the steps in Figure 4 for a client sending a query to its proxy using the word “computer”, we see that the DHT is queried with the hash of the word “computer” as key. This retrieves the query URL and respective UDDI keys of both services mentioned before in this example. The proxy can then do a simple lookup operation at both UDDI 1 and 2 registries. It is clear that as the number of UDDI registries and clients increases, this process of lookup at only relevant UDDI registries is more scalable than doing a full search using the word “computer” at all UDDI registries.

4. IMPLEMENTATION

In this section, we describe our implementation which is currently deployed on PlanetLab [9]. PlanetLab is an open, globally distributed platform for developing, deploying, and accessing network services. It currently has 527 machines, hosted by 249 sites, spanning over 25 countries. PlanetLab machines are hosted by research/academic institutions as well as industrial companies. France Telecom and HP are two of the major industry supporters for PlanetLab. Every PlanetLab host machine is connected to the Internet and runs a common software package including a Linux based operating system that supports server virtualization. Thus the users can develop and experiment with new services under real-world conditions. The advantage of using PlanetLab is that we can test the DUDE architecture under real-world conditions with a large scale geographically dispersed node base.

Due to the availability of jUDDI, an open source UDDI V2 registry (<http://www.juddi.org>) and a lack of existing readily available UDDI V3 registry, a decision to use UDDI V2 was made. The standardization of UDDI V3 is recent and we intend to extend this work to support UDDI V3 and subsequent versions in the future. The proxy registry is implemented by modifying the jUDDI source to enable publishing, querying and deleting service information from a DHT. Furthermore, it also allows querying multiple registries and collating the response using UDDI4j [13].

For the DHT implementation, we use the Bamboo DHT code [11]. The Bamboo DHT allows multiple proxy registries to publish and delete service information from their respective UDDI registries, as well as to query for services from all the registries. The proxy uses the service name as input to the DHT’s hash

function to get the DHT key. The value that is stored in the DHT using this key is the URI of the registry along with the UDDI key of the service. This ensures that when the proxy registry queries for services with a certain name, it gets back the URI and UDDI keys for matching entries. Using these returned results, the proxy can do fast lookup operations at the respective UDDI registries. The UDDI keys make it unnecessary to repeat the search at the UDDI registries with the service name.

We have so far described the process of exact match on service name. However there are additional types of search that must be supported. Firstly, the search requested could be case-insensitive. To support that, the proxy registry has to publish the same service once using the name exactly as entered in the UDDI registry, and once with the name converted to all lower-case letters. To do a case-insensitive search, the proxy registry simply has to convert the query string into lower-case letters. Secondly, the user could query based on the prefix of a service name. Indeed, this is the default behavior of search in UDDI. In other words, a wildcard is implicit at the end of the service name being searched. To support this efficiently in the DHT, our proxy registries have to take prefixes of the service name of varying length and publish the URI and UDDI key multiple times, once using each prefix. For example, the prefix sizes chosen in one deployment might be 5, 10, 15 and 20 characters. If a search for the first 12 characters of a service name is submitted, the proxy registry will query the DHT with the first 10 characters of the search string, and then refine the search result to ensure that the match extends to the 12th character. If the search string has less than 5 characters, and the search is for a prefix rather than an exact match, the DHT cannot be of any help, unless every service is published in the DHT with prefix of length 0. Using this null prefix will send a copy of every advertised service to the DHT node to which the hash of the null prefix maps. Since this can lead to load-imbalance, a better solution might be to use the DHT only to get a list of all UDDI registries, and send the search to all of them in the locations to be searched. Thirdly, the service name being searched can be a regular expression, such as one with embedded wildcard characters. For example, a search for “Garden%*s*” should match both “Garden Supplies” and “Gardening Tools”. This will be treated similarly to the previous case as the DHT has to be queried with the longest available prefix. The results returned have to be refined to ensure that the regular expression matches.

Figure 5 shows the network diagram for our implementation. There are two proxy UDDI and juddi registry pairs. Consider a client which contacts the UDDI proxy on `grouse.hpl.hp.com`. The proxy does a lookup of the DHT using the query string or a prefix. This involves contacting one of the DHT nodes, such as `pl1-br-3.hpl.hp.com`, which serves as the gateway to the DHT for `grouse.hpl.hp.com`, based on the latter’s configuration file. The DHT node may then route the query to one of the other DHT nodes which is responsible for the DHT key that the query string maps to. The results of the DHT lookup return to `pl1-br-3.hpl.hp.com`, which forwards them to `grouse.hpl.hp.com`. The results may include a few services from each of the juddi registries. So the proxy registry performs the lookup operations at both `planetlab1` and `planetlab2.rdfrcancetelecom.com` for their respective entries listed in the search results. The responses to these lookups are collated by the proxy registry and returned to the client.

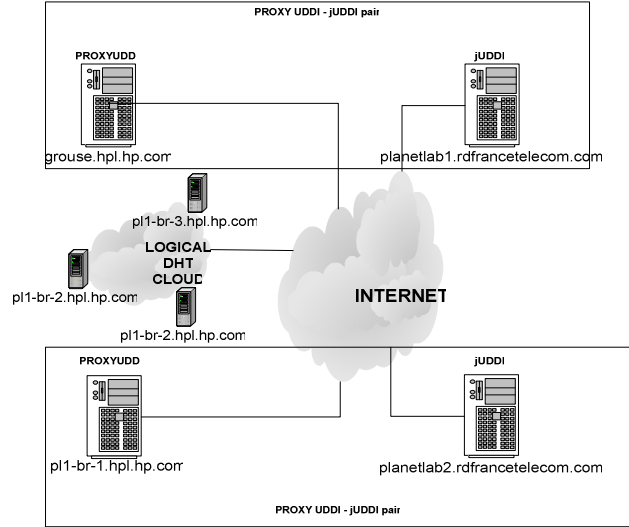


Figure 5 Network Diagram

5. RELATED WORK

A framework for QoS-based service discovery in grids has been proposed in [18]. UDDIe, an extended UDDI registry for publishing and discovering services based on QoS parameters, is proposed in [19]. Our work is complementary since we focus on how to federate the UDDI registries and address the scalability issue with UDDI. The DUDE proxy can publish the service properties supported by UDDIe in the DHT and support range queries using techniques proposed for such queries on DHTs. Then we can deliver the scalability benefits of our current solution to both UDDI and UDDIe registries. Discovering services meeting QoS and price requirements has been studied in the context of a grid economy, so that grid schedulers can use various market models such as commodity markets and auctions. The Grid Market Directory [20] was proposed for this purpose.

In [12], the authors present an ontology-based matchmaker. Resource and request descriptions are expressed in RDF Schema, a semantic markup language. Matchmaking rules are expressed in TRIPLE, a language based on Horn Logic. Although our current implementation focuses on UDDI version 2, in future we will consider semantic extensions to UDDI, WS-Discovery [16] and other Grid computing standards such as Monitoring and Discovery Service (MDS) [10]. So the simplest extension of our work could involve using the DHT to do an initial syntax-based search to identify the local registries that need to be contacted. Then the Proxy Registry can contact these registries, which do semantic matchmaking to identify their matches, which are then merged at the Proxy Registry and returned to the client.

The convergence of grid and P2P computing has been explored in [5]. GridVine [2] builds a logical semantic overlay on top of a physical layer consisting of P-Grid [1], a structured overlay based on distributed search tree that uses prefix-based routing and changes the overlay paths as part of the network maintenance protocol to adapt to load in different parts of the key-space. A federated UDDI service [4] has been built on top of the PlanetP [3] publish-subscribe system for unstructured P2P communities. The focus of this work has been on the manageability of the federated service. The UDDI service is treated as an application

service to be managed in their framework. So they do not address the issue of scalability in UDDI, and instead use simple replication. In [21], the authors describe a UDDI extension (UX) system that launches a federated query only if locally found results are not adequate. While the UX Server is positioned as an intermediary similarly to the UDDI Proxy described in our DUDE framework, it focuses more on the QoS framework and does not attempt to implement a seamless federation mechanism such as our DHT based approach. In [22] D2HT describes a discovery framework built on top of DHT. However, we have chosen to use UDDI on top of DHT. D2HT have used (Agent Management System) AMS/ (Directory Facilitator) DF on top of DHT.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have described a distributed architecture to support large scale discovery of web-services. Our architecture will enable organizations to maintain autonomous control over their UDDI registries and at the same time allowing clients to query multiple registries simultaneously. The clients are oblivious to the transparent proxy approach we have adopted and get richer and more complete response to their queries. Based on initial prototype testing, we believe that DUDE architecture can support effective distribution of UDDI registries thereby making UDDI more robust and also addressing its scaling issues. The paper has solved the scalability issues with UDDI but does not preclude the application of this approach to other service discovery mechanisms. An example of another service discovery mechanism that could benefit from such an approach is Globus Toolkit's MDS. Furthermore, we plan to investigate other aspects of grid service discovery that extend this work. Some of these aspects include the ability to subscribe to resource/service information, the ability to maintain soft states and the ability to provide a variety of views for various different purposes. In addition, we plan to revisit the service APIs for a Grid Service Discovery solution leveraging the available solutions and specifications as well as the work presented in this paper.

7. REFERENCES

- [1] "P-grid: A self-organizing structured p2p system". K. Aberer, P. Cudr_e-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Puceva, and R. Schmidt. ACM SIGMOD Record, 32(3), 2003.
- [2] "GridVine: Building Internet-Scale Semantic Overlay Networks" Karl Aberer, Philippe Cudre-Mauroux, Manfred Hauswirth, and Tim van Pelt. Proceedings, 3rd ISWC 2004, Hiroshima, Japan.
- [3] "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities". F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. In Proceedings of 12th Intl Symposium on HPDC, June 2003.
- [4] "Self-Managing Federated Services". Francisco Matias Cuenca-Acuna and Thu D. Nguyen. In Proceedings of 23rd IEEE International SRDS, 2004, Florianopolis, BRAZIL.
- [5] "On Death, Taxes, and the Convergence of P2P and Grid Computing". Ian Foster and Adriana Iamnitchi. In Proceedings of the 2nd IPTPS 2003.
- [6] "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", I. Foster, C. Kesselman, J. M. Nick and S. Tuecke. Presented to OGSIG WG, Global Grid Forum, June 22, 2002. Available at <http://www.globus.org/alliance/publications/papers.php>
- [7] "Was the Universal Service Registry a Dream?", Fred Hartman and Harris Reynolds, In the Web Services Journal, Dec 2, 2004.
- [8] "Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems". A. Rowstron and P. Druschel. In Proc. of IFIP/ACM Middleware, Nov. 2001
- [9] <http://www.planet-lab.org>
- [10] "Grid information services for distributed resource sharing". K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Proceedings of the IEEE HPDC-10, 2001.
- [11] "Handling churn in a DHT". S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Proceedings of the USENIX Annual Technical Conference, June 2004.
- [12] "Ontology-based Resource Matching in the Grid – The Grid Meets the Semantic Web", Hongsuda Tangmunarunkit, Stefan Decker, Carl Kesselman. In Proceedings of the Second ISWC (2003), Miami, Florida.
- [13] UDDI4j Java Class Library: <http://www-124.ibm.com/developerworks/oss/uddi4j/>
- [14] UDDI V2 specification: Available at <http://uddi.org/>
- [15] UDDI V3.0.2 specification: <http://uddi.org/>
- [16] Web Services Dynamic Discovery (WS-Discovery) Specification, February 2004. <http://msdn.microsoft.com/ws/2004/02/discovery>
- [17] Information Services (MDS): Key Concepts. <http://www.globus.org/toolkit/docs/4.0/info/key/>
- [18] "G- QoSM: Grid Service Discovery using QoS Properties", R J. Al-Ali, O.F. Rana, D.W. Walker, S. Jha and S. Sohail. Journal of Computing and Informatics (Special issue on Grid Computing), Ed: Domenico LaForenza, Vol. 21, No. 4, pp. 363-382, 2002.
- [19] "UDDIe: An Extended Registry for Web Services", A. ShaikhAli, O.F. Rana, R. Al-Ali and D.W. Walker, Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference, Florida, US, January 2003. IEEE Computer Society Press.
- [20] "A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services", J. Yu, S. Venugopal and R. Buyya, Journal of Supercomputing, Kluwer Academic Publishers, USA, 2005.
- [21] Chen Zhou, Liang-Tien Chia, Bilhanan Silverajan, Bu-Sung Lee: UX - An Architecture Providing QoS-Aware and Federated Support for UDDI. ICWS 2003: 171-176.
- [22] Kee-Hyun Choi, Ho-Jin Shin, Dong-Ryeol Shin, Service Discovery Supporting Open Scalability Using FIPA-Compliant Agent Platform for Ubiquitous Networks, Lecture Notes in Computer Science, Volume 3482, Jan 2005.