DECENTRALIZED AND HIERARCHICAL DISCOVERY OF SOFTWARE APPLICATIONS IN THE ISHARE INTERNET SHARING SYSTEM

Xiaojuan Ren Zhelong Pan Rudolf Eigenmann Y. Charlie Hu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, 47907, USA
{**xren,zpan,eigenman,ychu**}**@purdue.edu**

Abstract

We present the design and evaluation of a fully decentralized software application discovery scheme – *iDiscover*, which is used in the iShare Internet-sharing system being built at Purdue University. The scheme employs a structured peer-to-peer (P2P) overlay routing mechanism and a hierarchical name space. The structured P2P routing mechanism is self-organizing and scalable. The hierarchical name space provides an effective way to describe software applications with their semantics. Measurements using a set of real software applications show that our resource discovery mechanism is efficient and scalable.

1 INTRODUCTION

Internet sharing systems – systems that harness worldwide computational resources – are among the most exciting projects. Software applications are important computational resources, shared by end users and developers from diverse communities. The complexity of today's software applications requires nontrivial development and maintenance efforts, which are beyond most end users' capability to handle. Therefore, facilities for online access to such applications through the Internet are likely to become important tools. Applications could be services running at specific hosts, or programs available for downloading and execution. Given a large amount of applications available online, end users face the challenge of resource discovery, i.e., how to find the applications which provide functionalities matching the goal of a task at hand.

While resource discovery has gained tremendous interest in both the Grids and P2P communities, little work has been targeted at discovering software applications. Grids resource discovery [4] concentrates on how to locate suitable machines for job executions and P2P applications [13] have primarily focused on sharing files and computer cycles. In this paper, we present an original approach to building end-user services for searching for software applications that would accomplish a given task. Our approach is comprised of two major components: a dynamic hierarchical name space, and P2P-based resource publication and discovery protocols.

The success of wide-area services for locating resources based on a hierarchical naming scheme (e.g. DNS) argues that hierarchical naming is a powerful model for locating information. Software applications can be naturally mapped to a hierarchical name space, because they can be classified into categories based on their functionalities. Applications' functionalities are described by meta-data, which form a tree representing the hierarchical name space. To avoid the centrality of traditional hierarchical services, meta-data are published and inserted into a structured P2P overlay, by hashing path names in the tree. This leads to the fact that meta-data with similar semantics are likely to be co-located in the P2P overlay, thus reducing the search cost (in terms of the number of nodes searched) for a given query.

Our system, based on the hierarchical name space and structured P2P overlay, is called *iDiscover*. In *iDiscover*, resource descriptions are specified in the Resource Description Framework (RDF) [1], leveraging its flexibility and semantic-rich data model. Meta-data are distilled from resource descriptors and mapped to peer nodes based on hashing. New hierarchical levels are created on the fly when publishing meta-data. Because peers unexpectedly leave and join, a replication algorithm is designed to maintain published meta-data. End users can search for applications with certain functionalities. There may be multiple application instances matching a user's search. We propose a proximity-aware resource cache to find the nearest application resource to the searching node, as well as to reduce search latency.

The contributions of *iDiscover* include:

- It supports the publication and discovery of both existing and new software applications. There are no assumptions about software interfaces or languages. No extra programming effort or static configuration is required.

- It provides an interactive and incremental query model.

With this model, users can issue queries incrementally, without specifying all search criteria.

- It employs RDF as the description language, which provides the potential for semantic interoperability among software applications.

- It is efficient and reliable. The proximity-aware caching reduces discovery latency and finds the nearest resource to a request node. The replications provide fault tolerance under quick membership changes.

We have implemented the proposed approach in iShare, an Internet sharing system being developed at Purdue University. iShare builds on the early experience of the Purdue University Network Computing Hubs [3, 10]. With its computational application focus, iShare aims to improve collaborations between application developers and end-users.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 gives a brief overview of the iShare system. Section 4 describes the design and implementation of *iDiscover*. Simulation results are given in Section 5.

2 RELATED WORK

Resource discovery is a research effort with a long history. To our knowledge, our paper is the first to explore hierarchical naming and P2P routing to discover software application resources. Here, we represent related work in three categories: decentralized resource discovery with structured P2P overlays, directory-style service discovery, and hierarchical naming based global discovery.

Structured P2P systems [15, 16, 19] provide name-to-location mapping services via distributed hashing techniques. These location services are novel in their wide-area scalability and self organization. However, they are insufficient when users are searching for a resource matching multiple required properties. One way to solve the problem is to allow resources to reside in a multi-dimensional space. For example, XenoSearch [18] utilizes Pastry [16] for locating resources. Information about resources is partitioned between nodes. Queries for specific resources are directed to the node responsible for the partition. Another example is the work by Schmidt and Parashar [17] which employs Space Filling Curves to map multi-dimensional index space to nodes in a structured P2P overlay network built on Chord [19]. The use of multi-dimensional search to match multiple required resource attributes provides an efficient means for locating resources. However, it requires complex algorithms to map multi-dimensional index spaces to peer nodes. By contrast, our *iDiscover* method adopts a hierarchical name space and exploits the simple name-to-location mapping by hashing partial paths in the hierarchical structure.

Jini [22], SLP [6], and SDS [7] are examples of directory-style, worldwide service discovery systems. Service registration servers are organized into multiple shared directories. Service announcers and queriers dynamically discover some server in a directory and interact with the entire system through them. This differs from our P2P-based architecture, which has no fixed servers in the entire system.

DNS [11] and Globe [20] utilize the hierarchical structure inherent in their unique service names. In the former case, names are mapped to addresses; in the latter, object identifiers are mapped to the object broker that manages them. All these two mappings are somehow static. The hierarchical name space we exploit is more flexible because its structure is adaptive to arbitrary application classifications. Leveraging the distributed hashing, resource mapping in the hierarchical space is dynamic and random.

3 ISHARE SYSTEM OVERVIEW

3.1 Motivation

iShare is an Internet sharing system with a decentralized and self-organizing architecture. It aims to provide flexibility in dynamically announcing, discovering and composing computational resources. Meanwhile, it embraces existing standards and technology, such as NSF's *Middleware* systems [12]. The main components in iShare include end user functionalities, iShare protocol plug-ins, service host protections and decentralized resource management. Detailed description of iShare [9] is beyond the scope of this paper. Here, we only focus on the resource management component.

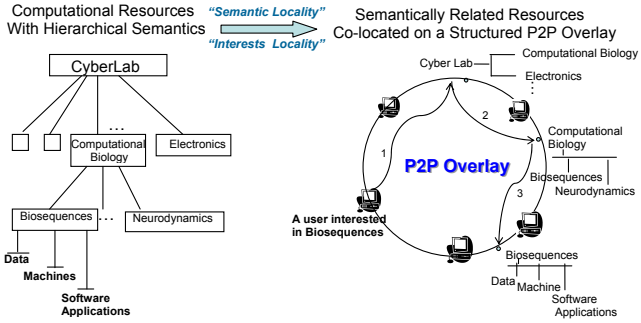

Figure 1: Architectural overview. The big circle represents a P2P ring, with arrows indicating P2P routing messages to discover resources.

One observation behind iShare's resource management is that computational resources can be hierarchically categorized into *CyberLabs*, as shown in Figure 1. The resources

in one lab are semantically related in their functionalities. We call this *semantic locality*. Another observation is *interests locality*: most often, a user is only interested in few labs and tends to share resources in those labs. To exploit these localities, iShare distributes resources within a hierarchical name space to a P2P overlay. An item in the hierarchical space is mapped to a peer node based on the hashing value of the item's path name. A child's path name, instead of its mapped node's id, is kept in the parent's repository. Therefore, users can incrementally search for interested resources without specifying all the search criteria.

3.2 End User Functions

Resource management in iShare provides the basic support to many end-user functions. Two essential functions are publishing available resources and searching for resources that match specific user needs.

Figure 1 presents a simple example for searching for a software application. A user interested in Biosequences joins iShare, trying to search for a software tool to do sequence analysis. With no prior knowledge about potential applications to accomplish the task, the user starts the search by sending a query to the global root "CyberLab". The query returns a list of labs, including "Computational Biology", semantically related to "sequence analysis". Then the user continues searching in "Computational Biology" and finds the "Biosequences" lab. Probably, she may find the software tool to do sequence analysis in this lab. To exploit the *interests locality*, recently-used meta-data are cached to reduce discovery latency. If the user searches for data sets of biosequence patterns in the future, she can start searching in "Biosequences" directly.

After discovering the tool, the user can put it into different labs (e.g. "Biopatterns") in the local resource cache by creating links to the original "Biosequences" lab. This simple mechanism provides flexibility and accommodates the fact that application providers and end-users may have different preferences in categorizing resources.

3.3 Resource Publication

To specify resource semantics, iShare adopts an expressive resource description language – Resource Description Framework. Resource providers create resource descriptors by an interactive tool. A resource descriptor is translated to low-level meta-data, which are used for publishing the resource. Figure 2 presents the meta-data and messages created for publishing a software tool named "GeneParser". The tool is used to parse a DNA sequence into introns and exons. It resides in the "Biosequences" lab under the "Computational Biology" category.

Multiple "iSharePublish" messages are created based on the meta-data: one message to check or create the category (key = "CyberLab", content = category); one message for each category to check or create the lab (key = category, content = lab); and two messages to create an instance of the application (key1 = category//lab//software, content1 = application//provider//version; key2 = content1, content2 = short description of the application). A message is routed to the destination node whose Id is numerically closest to the hashing value of the message's key. To achieve fault tolerance, each publish operation creates a few replicas. We will talk about this in details in Section 4.2.2.

Message	Key	Content
1	CyberLab	Computational Biology
2	Computational Biology	Biosequences
3	Computational Biology //Biosequences//Software	GeneParser//bio. abc.edu// 1.0
4	GeneParser//bio.abc.edu// 1.0	Short Description

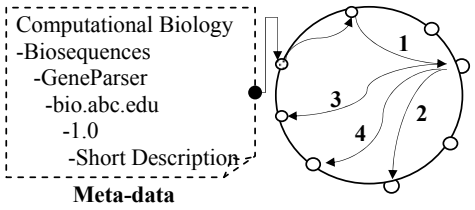

Figure 2: An example for publishing GeneParser. (category=Computational Biology; lab=Biosequences; name=GeneParser; provider=bio.abc.edu; version=1.0)

4 IDISCOVER ARCHITECTURE

We have developed a prototype implementation of iShare, which includes the decentralized and hierarchical software application discover scheme, *iDiscover*. In the rest of the paper, we start with a brief review of two *iDiscover* building blocks.

4.1 Building Blocks

4.1.1 Structured P2P Overlay Networks

Structured P2P overlay networks such as CAN [15], Chord [19] and Pastry [16] effectively implement scalable *distributed hash tables* (DHTs), where each node in the network has a unique node identifier (`nodeId`) and each data item stored in the network has a unique key. The `nodeId`s and keys live in the same name space, and each key is mapped to a unique node in the network. Thus DHTs allow data to be inserted without knowing where it will be stored, and requests for data to be routed without requiring any knowledge of where the corresponding data items are

stored. While any of the DHTs can be used to build *iDiscover*, we use Pastry as an example in this paper. In the following, we briefly explain the DHT mapping in Pastry.

Pastry [16, 2] is a scalable, fault resilient, and self-organizing P2P substrate. Each Pastry node has a unique, uniform, and randomly assigned `nodeId` in a circular 128-bit identifier space. Given a message and an associated 128-bit key, Pastry reliably routes the message to the live node whose `nodeId` is numerically closest to the key.

In Pastry, each node maintains a routing table that consists of rows of other nodes' `nodeId`s which share different prefixes with the current node's `nodeId`. In addition, each node also maintains a leaf set, which consists of l nodes with `nodeId`s that are numerically closest to the present node's `nodeId`. The leaf set ensures reliable message delivery and is used to store replicas of application objects. Pastry routing is prefix-based. At each routing step, a node seeks to forward the message to a node whose `nodeId` shares with the key a prefix that is at least one digit longer than the current node's shared prefix. A more detailed description of Pastry can be found in [2, 16].

4.1.2 Resource Description Framework

Resource description is an essential ingredient of resource management. iShare's Resource Description Form (iRDF) builds on W3C's Resource Description Framework (RDF) [21] as the semantic description language.

RDF is an XML-based language for describing information contained in a Web resource. One of the goals of RDF is to specify semantics for data in a standardized interoperable manner [1]. Compared to XML, RDF provides a richer data model for describing objects. The relationships of two objects can be arbitrarily created and stored separately from the two objects. This nature of RDF is very suitable for the dynamically changing, shared nature of distributed resources [1]. Thus, RDF has steadily increased in adoption as a resource description language for web applications and web data.

4.2 Design and Implementation

In this section, we describe a number of aspects how the operations defined in Section 3 are implemented over Sun's Java SDK 1.4.1 and FreePastry [5].

4.2.1 Semantic Description with RDF

We employ RDF to describe software applications and define the generic semantics for an application, which includes four major parts: resource attributes, command interface, access controls and documentation. Figure 3 shows the RDF document created for a software tool named "GeneParser".

```
<?xml version="1.0"?>
……
<i:IShareAppRes rdf:ID="GeneParser">
   <i:res_name>GeneParser</i:res_name>
   <i:res_provider rdf:parseType="Resource">
       <i:prov_org>bio.abc.edu</i:prov_org>
       ……
   </i:res_provider>
   <i:app_category>Computational
         Biology//Biosequences</i:app_category>
   <i:app_manpageurl>http://……</i:app_manpageurl>
   ……
   <i:app_access rdf:parseType="Resource">
       <i:acc_operation>download</i:acc_operation>
       ……
   </i:app_access>
   <i:app_hardware rdf:parseType="Resource">
       <i:hw_arch>any</i:hw_arch>
       …….
   </i:app_hardware>
   <i:app_arguments rdf:parseType="Resource">
       <i:arg_name>precede sequence file name</i:arg_name>
       <i:arg_type>input file</i:arg_type>
       ……
   </i:app_arguments>
   ……
```

Figure 3: Example iRDF document.

We use the ICS-FORTH Validating RDF Parser (VRP) [8] to parse RDF documents. The parser creates an array of RDF sentences. Each sentence contains three parts: *subject*, *predicate*, and *object*. A semantic processor is implemented to build RDF sub-nodes from RDF sentences. Each sub-node relates to one semantic property of an application resource. The type of the sub-node is decided by the *predicate* and its identity is decided by the *object*. After semantic processing, a RDF source node is created, with all the sub-nodes as its fields. This node contains all the semantic information for the described application resource.

4.2.2 Fault Tolerance

To achieve reliability, inserted meta-data are replicated across multiple nodes. The number (K) of replicas for each resource is chosen by the provider. The system keeps the invariant that K copies of the inserted meta-data are maintained on the nodes with the K closest `nodeId`s to the key (hashing value) of the meta-data. Because `nodeId`s are randomly assigned, with high probability, the set of nodes over which the meta-data are replicated are diverse in terms of geographic location and ownership.

The invariant of K replicas is maintained as nodes join and leave the iShare overlay network. When a new node arrives, it initializes its state tables, and then informs other nodes of its presence, which may lead to changes in some nodes' leaf sets. The change in a node's leaf set will trigger replica adjustments to maintain the invariant. An iShare node is considered failed when its immediate neighbors in the `nodeId` space can no longer communicate with the node. The two immediate neighbors remove the failed node in the leaf sets and initiate the replica adjustment. The detailed operations to keep the invariant are described below:

- When a new resource is published, the node whose `nodeId` is closest to the key of resource meta-data holds the *primary replica* for this resource, as shown in Figure 4(a). This node will choose among Pastry leaf set to get K replica candidates.

- When a new node joins, only those nodes whose leaf sets change are affected. So, if its leaf set changes, a *primary replica* holder will find out whether the *primary replica* or other replicas should be migrated to the newly joined node. In the first case, the K replicas may be migrated to keep the invariant. The two cases are shown in Figure 4(b) and 4(c) respectively.

- When a node leaves, the two immediate neighbors in the circular identifier space detect the departure. One of these two nodes with closer `nodeId` to the key will become the new *primary replica* holder. After that, a new copy of replica is created and saved at an appropriate node to keep the invariant. Figure 4(d) shows a simple example for node leaving.

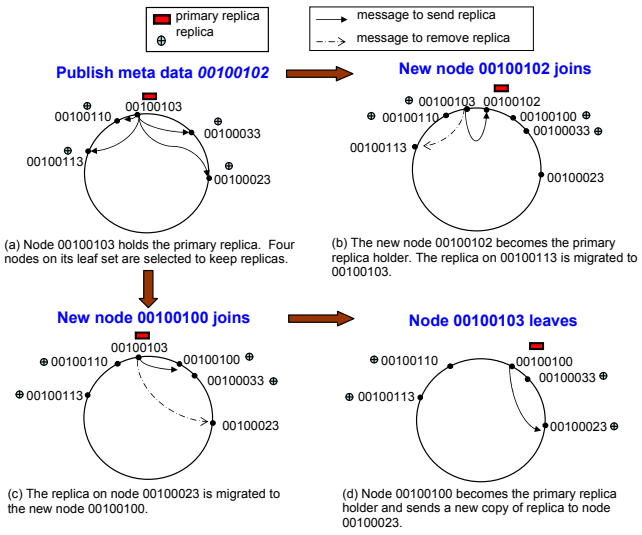

Figure 4: Examples for fault tolerance. To simplify the description, K is assumed to be 4. In practice, K could be any number chosen by the provider.

4.2.3 Proximity Aware Cache

To exploit *interests locality*, a local resource cache is designed to keep recently-used meta-data. Successful end-user operations of discovery and publishing will update the cache. When publishing a new resource, messages to check or create the hierarchical path will not be sent if the path is found in the local cache. When discovering a resource, matching cache items are returned immediately. A cached item is invalidated in three cases: (1) it expires after a live period; (2) any operation using the cached item fails; and (3) the user explicitly flushes the cache with a new discovery. We test the effect of caching on discovery latency and show the results in Section 5.

With different versions and providers, there could be multiple application instances matching a user's search. The cache keeps a sorted list of application instances with respect to proximity. Here, the proximity is defined as the actual network latency from the local node to a service host. The proximity is determined by pinging the actual service host for each application instance on the list. Such selection of nearby application services leads to saved bandwidth for data transfer and faster task completion.

5 EVALUATION

In the following, we report the results from simulating a large number of P2P nodes over the iShare testbed. We are interested in understanding response time per discovery and measure response time as the number of hops traversed for answering a request. We only focus on evaluating the effect of resource caching in this paper. Evaluation of fault tolerance is beyond the scope of this paper.

5.1 Methodology

We simulated our *iDiscover* design on a GT-ITM router network using the transit-stub model [23]. The size of the IP network is 1050 routers, 50 of which are used in transit domains and the remaining 1000 in stub domains. To test the scalability of *iDiscover*, we simulated several iShare testbeds with the number of nodes ranging from 500 to 10,000. We assume the iShare nodes are uniformly attached to the routers.

Our simulations use a collection of software tools and libraries located at the Pittsburgh Supercomputing Center (PSC) [14]. These are real scientific applications from 12 different science areas.

To measure the influence of user request patterns on discovery latency, we experimented with the naive *iDiscover* (without caching) under two different synthetic user request patterns: one that follows the Zipf law and another that follows a uniform distribution. The parameter in the Zipf-like query distribution is 1.20. Each discovery operation starts from searching the root "CyberLab" and finishes at getting a specific application instance.

To understand the effect of resource caching, we compared discovery response time and cache hit rate under different cache expiration times. We repeated each experiment multiple times and the results shown in the next section are the average values of our measurements.

5.2 Results

Two sets of simulation results are presented. First, we measured the discovery response time without caching. For the simulation, we published all the 500 applications and then tested for discovery under two different user query patterns. (1) For the uniform query pattern, each application was discovered once. (2) For the Zipf-like request model, the applications with rank i were discovered $500/i^{1.20}$ times. For each discovery operation, one node in the P2P overlay was randomly chosen to initiate the request. Figure 5 shows the average number of hops traversed for each discovery without caching. From the graph it can be seen that the discovery latency increases very slowly with the total number of nodes and the influence of different resource request models is insignificant. Compared with routing a basic message over Pastry, *iDiscover* takes 3-4 times longer. This follows from the simulated fully incremental query model. Four Pastry messages are required to search at the 4 hierarchical levels from the root "CyberLab" to a specific application instance. And the discovery acknowledgment adds one more hop for each search.

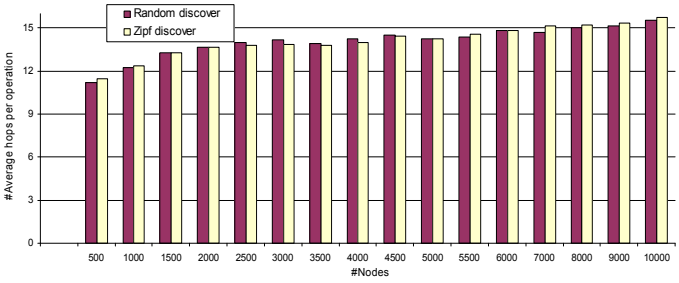

Figure 5: Average number of hops per discovery (no cache) as a function of the number of iShare nodes.

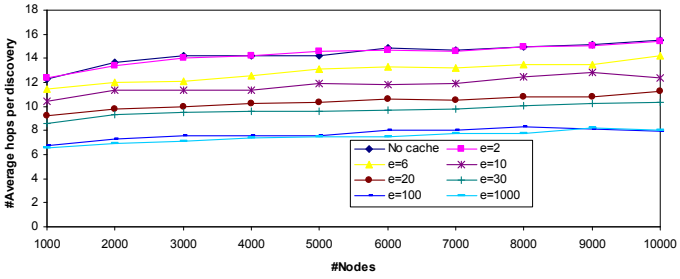

Figure 6: Average number of hops per discovery for the random request model. e is the normalized cache expiration time (cache expiration time/average request period).

In the second set of results, we measured the effect of caching with different normalized cache expiration time e (e = cache expiration time/average request period). In this set of experiments, we adopted the random request model and one out of five nodes in the P2P overlay was randomly chosen to initiate the search request. Figure 6 and Figure 7 plot discovery response time and cache hit rate with e ranging from 2 to 1,000. We see that with fair expiration time (e = 6), the response time is reduced by 10.37% on average compared to a discovery without local cache. As expected, increasing the cache expiration time results in lower discovery latency and higher cache hit rate. Ideally (cache will never expire), each discovery takes only 7 hops with the cache hit rate close to 50%. Figure 8 shows the results by examining the cache performance more closely. Figure 8(a) indicates that cache hit rate is not a function of system scale and Figure 8(b) shows that the hit rate increases exponentially and arrives its maximum value 51.7% at e = 173. We chose the cache expiration time e between 5 and 10 ($log(e)$ is between 0.7 and 1.0), which reduces the discovery response time by 10%-18% compared to that without caching, as shown in Figure 8(c). Figure 8(d) plots discovery latency decrease rate versus cache hit rate. It is close to a linear function with coefficient 1.0645.

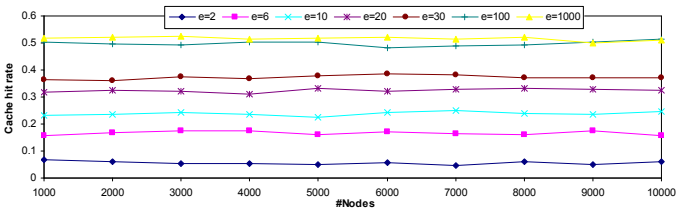

Figure 7: Cache hit rate for the random request model.

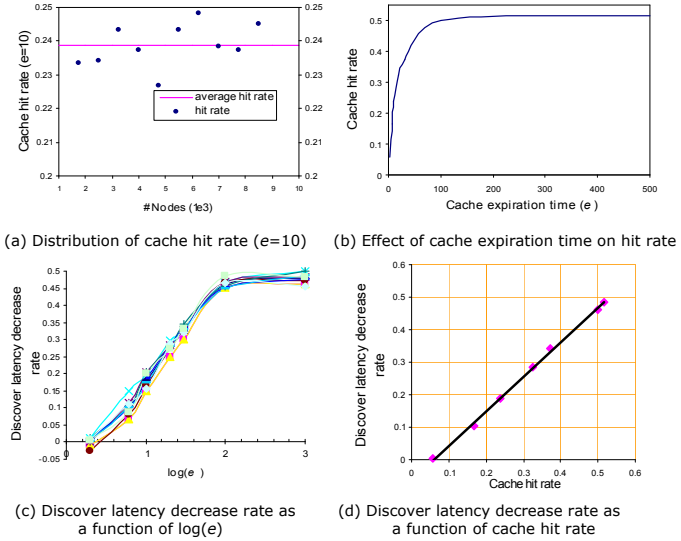

Figure 8: Cache hit rate and discover latency decrease rate.

6 CONCLUSION AND FUTURE WORK

In this paper we have introduced a decentralized and hierarchical mechanism to publish and discover software applications. Software applications are mapped to a hierarchical structure based on their functionality semantics. The hierarchical structure is distributed onto a structured P2P overlay by hashing partial path names. Thus, semantically related resources tend to be co-located on the same peer node, which reduces the discovery latency. To exploit *interests locality*, a local resource cache is designed and evaluated.

This work was presented with the aim to develop a resource management system for the iShare Internet sharing system. The resource management aims to provide flexibility in dynamically announcing, discovering, and composing computational resources.

We plan to extend the current work in a number of directions. First, our evaluation of the mechanism presented here is obtained through simulation. In ongoing research, we are deploying the system in a large-scale distributed computing environment within iShare. Second, we expect that the study of resource usage patterns will lead to significant improvement in the caching performance. Finally, as part of the iShare project, we are exploring security and resource composition issues.

7 ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under Grants No. 9986020-EIA, 9974976-EIA, 0226851-EIA, and the CAREER award ACI-0238379.

8 REFERENCES

[1] K. S. Candan, H. Liu, and R. Suvarna. Resource description framework: Metadata and its applications. *ACM SIGKDD Explorations Newsletter*, 3:6–19, 2001.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. Technical report, Technical report MSR-TR-2002-82, 2002.

[3] J. A. B. Fortes, N. H. Kapadia, R. Eigenmann, et al. On the use of simulation and parallelization tools in computer architecture and programming courses. In *Proceeding of 2000 ASEE Annual Conference & Exposition*, June 2000.

[4] I. Foster (Ed.) and C. Kesselman (Ed.). *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[5] FreePastry. `http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry`.

[6] E. Guttman, C. Perkins, J. Veizades, et al. Service location protocol, version 2, November 1998.

[7] T. D. Hodes, S. E. Czerwinski, et al. An architecture for secure wide-area service discovery. *Journal on Wireless Networks*, 8(2-3), March 2002.

[8] ICS-FORTH Validating RDF Parser (VRP). `http://athena.ics.forth.gr:9090/RDF/`.

[9] iShare Project. `http://paramount.www.ecn.purdue.edu/ParaMount/`.

[10] N. H. Kapadia and J. A. B. Fortes. PUNCH: An architecture for web-enabled wide-area network-computing. *Cluster Computing*, 2:153–164, 1999.

[11] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Proceedings of SIGCOMM '88*, August 1988.

[12] NSF Middleware Initiative. `http://www.nsf-middleware.org`.

[13] A. Oram (Ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, 2001.

[14] Pittsburgh Supercomputing Center. `http://www.psc.edu/general/software/`.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proc. SIGCOMM'01*, pages 161–172, San Diego, CA, 2001.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[17] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proc. HPDC'03*, Seattle, WA, June 2003.

[18] D. Spence and T. Harris. XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform. In *Proc. HPDC'03*, Seattle, WA, June 2003.

[19] I. Stoica, R. Morris, D. Karger, et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, San Diego, California, August 2001.

[20] M. van Steen, F. Hauck, et al. Locating objects in wide-area systems. *IEEE Communications Magazine*, January 1998.

[21] Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/rdf-concepts/`.

[22] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.

[23] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. IEEE INFOCOM*, March 1996.