Semantics-Based Resource Discovery in Large-Scale Grids

Juan Li[1], Samee Ullah Khan[1,*], and Nasir Ghani[2]
[1]North Dakota State University
{j.li, samee.khan}ndsu.edu
[2]University of New Mexico
nghani@ece.unm.edu

Abstract

With the present ubiquitous network connections and the growing computational and storage capabilities of modern everyday-use computers, more resources, such as PCs, handheld devices (e.g., PDAs and sensors), applications, and services are on grid networks. Grid is expected to evolve from a computing and data management facility to a pervasive, world-wide resource-sharing infrastructure. To fully utilize the wide range of grid resources, effective resource discovery mechanisms are required. However, resource discovery in a global-scale grid is challenging due to the considerable diversity, large number, dynamic behavior, and geographical distribution of the resources. The resource discovery technology required to achieve the ambitious global grid vision is still in its infancy, and existing applications have difficulties in achieving both rich searchability and good scalability. In this chapter, we investigate the resource discovery problem for open-networked global-scale grids. In particular, we propose a distributed semantics-based discovery framework. Moreover, we show how this framework can be used to address the discovery problem in such grids and improve three aspects of performance: expressiveness, scalability, and efficiency.

1. Introduction

The new generation of grids enable the sharing of a wide variety of resources, including hardware, software packages, knowledge information, licenses, specialized devices, and other grid services [1]. These resources are geographically distributed and owned by different organizations. The fact that users typically have little or no knowledge of the resources contributed by other participants in the grid poses a significant obstacle to their use. For this reason, resource discovery is a vital part of a grid system, and an efficient resource discovery infrastructure is crucial to make the distributed resource information available to users in a timely and reliable manner. However, resource discovery in large-scale grids is very challenging due to the potential large number of resources, and their diverse, distributed, and dynamic nature. In addition, it is equally difficult to integrate the information sources with a heterogeneous representation format.

* Corresponding author.

The provisioning of an *information service*, as currently envisaged by the grid community, is a first step towards the discovery of distributed resources. However, a large part of these efforts have been focused on "getting it to work," without directly addressing issues of scalability, reliability, and information quality [1]. For example, classical grids, such as the Globus Toolkit [2], always use centralized or static hierarchical models to discover resources. To discover resources in a more dynamic, large-scale, and distributed environment, peer-to-peer (P2P) techniques have been used in resent research (e.g., [3] and [4]). P2P systems offer many benefits, such as adaptation, self-organization, fault-tolerance, and load-balancing, but they also present several challenges that remain obstacles to their widespread acceptance and usage in grids: First, current P2P systems offer limited data management facilities. In most cases, searching information relies on simple identifiers or Information Retrieval (IR)-style string matching. This limitation is acceptable for file-sharing applications, but in order to support complex resource discovery in grids we need richer facilities for exchanging, querying and integrating structured and semi-structured data. Second, most P2P systems specialize in a single functionality, for example, music sharing. More efforts are needed to support the sharing of varieties of resources in grids. Moreover, designing a good search mechanism is difficult in P2P systems because of the scale of the system and the unreliability of individual peers.

An effective grid resource discovery mechanism should support expressive query language. Most of the existing search systems use simple keyword-based lookups, which limit the searchability of the system. Our proposed framework improves search expressiveness from two directions: First, it uses a semantic metadata scheme to provide users with a rich and flexible representation mechanism, to enable effective descriptions of desired resource properties and query requirements. Second, we employ ontological domain knowledge to assist in the search process. The system is thus able to understand the semantics of query requests according to their meanings in a specific domain; this procedure helps the system to locate only semantically related results.

The more expressive the resource description and query request, the more difficult it is to design a scalable and efficient search mechanism. We ensure scalability by reconfiguring the network with respect to shared ontologies. This reconfiguration partitions the large unorganized search space into multiple well-organized semantically related sub-spaces that we call semantic virtual organizations. Semantic virtual organizations help to discriminatively distribute resource information and queries to related nodes, thus reducing the search space and improving scalability. To further improve the efficiency of searching the virtual organizations, we propose a semantics-based searching mechanism OntoSum. OntoSum utilizes the famous "small-world" theory to reorganize the virtual organization so that query can be answered efficiently by forwarding between neighboring nodes.

The remainder of this chapter is organized as follows: Section 2 provides a general overview of related work in resource discovery in grids. Section 3 describes our approach to the construction of semantic virtual organizations (VOs). Section 4 presents OntoSum – a framework for semantic resource discovery in virtual organizations. Extensive evaluations are detailed in Section 5. Finally, Section 6 presents our conclusions, a discussion of limitations, and suggestions for future work.

2. Related Work

Traditionally, a Grid Information Service is mainly based on a centralized or hierarchical model. In the Globus Toolkit 2 [2], the Monitoring and Discovery Service (MDS) [5] provides access to static and dynamic information about resources. MDS is based on the Lightweight Directory Access Protocol (LDAP) [6], and consists of two components: Grid Index Information Services (GIIS) and Grid Resource Information Service (GRIS). The resource information is obtained by the information provider and is passed on to GRIS. GRIS registers its local information with the GIIS, which registers with another GIIS, and so on. MDS clients can get the resource information directly from GRIS (for local resources) and/or a GIIS (for grid-wide resources). The MDS hierarchy mechanism is similar to DNS. GRIS and GIIS, at lower layers of the hierarchy, register with the GIIS at upper layers, realizing the global indexing and discovery. Globus Toolkit versions 3, 4 and 5 [2] provide a service-oriented information service, i.e., the Index Service. The Index Service leverages service data defined in the Open Grid Services Architecture (OGSA) [7] specification to provide services. All services are described in a standardized XML schema, called Elements of Service Data (SDEs). The Index Service provides high-level API functionalities to register, aggregate, and query SDEs. Users can get a node's resource information by either directly querying a server application running on that node, or querying dedicated information servers that retrieve and publish the resource information of the organization. Techniques for associating information servers, and to construct an efficient, scalable network of directory servers, are left unspecified.

Other grid applications proposed similar information services. For example, Condor's Matchmaker [8] uses a centralized mechanism to locate desirable resources. Each node in the Condor system advertises its resources and reports resource status to a central manager. The central manager then matches resource requesters' queries with resource providers' advertisements. In another example, Legion [9] takes an object-oriented approach to resource management. It uses *Collections* to search and locate resources in the grid. When a user requests a resource, Legion will query resource information in multiple Collections; if it finds several such resources, Legion's resource scheduler will randomly choose one of them.

For small-to-medium scale grids, these centralized or static hierarchical solutions work fine. However, for large, up to global-scale grids, these approaches are not efficient and do not scale. Additionally, even for smaller grids, a centralized solution will always be a performance bottleneck and a single point of failure. Presently, grids have moved from the obscurely academic to the highly popular. As the size of the grid grows from tens to thousands or even millions of nodes, the traditional server-based grid information service will not scale well. As a remedy, some researchers (e.g., [10]) advocate the use of P2P techniques for implementing scalable grid systems.

P2P systems offer many benefits over the traditional client-server model, including better scalability, automatic management, fault-tolerance, and load-balancing. Therefore, we use P2P as our underlying communication structure. At the same time, P2P systems also present several challenges that preclude their widespread acceptance and usage in grids. For example, current P2P systems often lack the ability to deploy production-quality services, persistent and multipurpose service infrastructure, complex

services, robustness, performance, and security. Thus, one of the tasks of this chapter is to overcome these problems and make P2P systems better serve grid needs.

3. Virtual Organization Formation

3.1 Overview

If not properly organized, searching an large-scale grid for quality resources is like looking for a needle in a haystack – we have too large of a space to explore. Therefore, as the first step of our discovery scheme, we organize and reduce the huge chaotic search space into multiple semantics-based sub-spaces. Participants in each sub-space share similar semantic interests, forming semantics-based Virtual Organizations (VO). Searching can then be performed on VOs, and queries can be quickly propagated to many appropriate members in the VO. This procedure results in a higher precision and recall of search results.

3.2 Ontological Directories

To organize different interests and to facilitate the construction of VOs, we propose an abstract generic ontological model that guides users in determining the desired ontological properties and choosing the "right" VOs to join. The ontology model defines most general categories of existence (e.g., existing item, spatial region, dependent part), which essentially form a hierarchy, within which each entry corresponds to a categorical domain. Here we provide a formal definition of this ontology model, which we call the ontology directory.

DEFINITION 1: An ontology directory is a system $D=(L,H,r)$, which consists of:

- *A lexicon: The lexicon L contains a set of natural language terms.*
- *A hierarchy H: Terms in L are taxonomically related by the directed, acyclic, transitive, reflexive relation H.* ($H \subset L \times L$);
- *A root term* $r \in L$. *For all* $l \in L$, *it holds:* $H(l,R)$.

The ontology directory essentially defines a hierarchy, within which each node corresponds to a lexicon or a categorical term. It is almost a rooted tree structure, with rare nodes having multiple parents. The subordination relationship between nodes is interpreted as the involvement (topic/subtopic) relationship, while the layers of nodes correspond to intuitively perceived levels of abstractness of topics. Each node is described by primitives that are generic concepts that may include other concepts. An example of a primitive is *computer* that includes *software*, *hardware*, *networks*, and so forth. The hierarchical relationship, also called the IS-A relationship, is transitive. That is, whatever holds for a more general concept also holds for a more specific concept, e.g., *music* is a type of *art*.

The ontology model allows users to choose the right VO to join, detect new trends, or find useful information they did not realize was available. Our ontology directory is different from those global web directories, such as Google directory, Yahoo directory, and DMOZ [11], because it is not predefined, but created and extended automatically with network growth and the evolution of the ontology. Moreover,

the ontology directory loosely defines domain categories; it does not expect different communities of users to conform to the same ontology to describe their resources and interests. Therefore, it is based on multiple ontologies as opposed to a global ontology.

To implement the ontology directory in a decentralized manner, we propose an efficient and scalable distributed hash table (DHT) structure [12] to index and lookup the hierarchical taxonomy. To index and retrieve the hierarchical ontology directory with a flat DHT structure, we extend the basic DHT API. The directory path starting from the root is used to represent the ontology domain (e.g., /computer science/systems/network). One domain corresponding to a particular VO should include contact information for peers in this VO. A direct indexing scheme is to index the full directory path as a key, and users can locate a VO by providing the full directory path. However, unlike navigating in a UNIX file system, users rarely input an absolute directory path, but rather browse directories level by level and select the more interesting one at each level. Therefore, it is necessary to provide users an ontology browsing interface. Moreover, to automatically locate related VOs for nodes, we extract key concepts from the joining nodes' ontology and then use them as keys to locate the right directory domain. Therefore, we should also provide a keyword-based lookup interface.

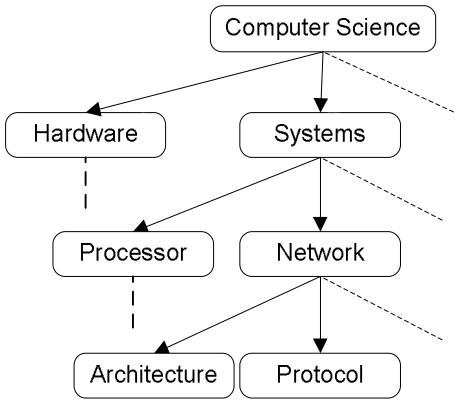


Figure 1. Fragment of an ontological directory model

Consider the ontology model in Figure 1. It consists of taxonomy paths:

/computer science
/computer science/systems
/computer science/hardware
/computer science/systems/network
/computer science/systems/processor
/computer science/systems/network/architecture
/computer science/systems/network/protocol

Some domains may relate to keywords, for example:
Keywords: *cluster, grid, P2P,* are related to taxonomy */computer science/system/network/architecture*
Keywords: *protocol, TCP, IP* are related to taxonomy */computer science/system/network/protocol*

For each path and keyword, a hash value (key) is computed in Pastry [13], a DHT implementation, using an SHA-1 algorithm. Table 1 shows keys for taxonomy paths and keywords of the model. To make the example simple, we use a 4-digits (8 bits) identifier space; however, in reality a much larger identifier space is used, such as 160 or 128 bits. Each key is assigned to a node, which is the nearest node to the key in the key-space. For example, as listed in Table 1, the hashed key of directory path */computer science/system* is *0230,* and the key is stored at node *0213* as shown in Figure 2, because node *0213*'s id is closest to the key. Each owner node of a directory key maintains a Least Recently Used (LRU) cache storing contact information of peers that are interested in this directory. To implement the directory browser's functionality, an overlay node that is in charge of a directory entry also stores information about that directory's direct children. When the user chooses one directory, Pastry routes to that directory entry and retrieves child directory information, allowing the directory to be extended dynamically while browsing. An overlay node also stores keywords that are hashed to it and links the keywords with related ontology domains. Figure 2 shows how the directory model above is stored into an example Pastry network.

Table 1. Hash keys of models in Figure 1 in a sample 4-digit identifier space

Hash key	**Directory path**
1211	/computer science
0230	/computer science/systems
3211	/computer science/hardware
2011	/computer science/systems/network
1000	/computer science/systems/processor
1013	/computer science/systems/network/architecture
0012	/computer science/systems/network/protocol
2111	Protocol
0211	TCP
1201	IP
2003	Cluster
0012	Grid
0032	P2P

Because nodes might fail and network connections might break, the ontology model stored on its corresponding overlay nodes are replicated on its neighbors in the Pastry identifier space. This can be done by setting the replica factor *f*. Whenever a node receives a directory storing request, it will not only store the directory locally but also store it to its *f* immediate leaf nodes. If any node fails or its connection breaks, its leaf neighbors will detect it by using the keep-alive messages.

3.3 Ontology Directory Lookup and VO Register

We provide three kinds of lookup interfaces for users: (a) exact lookups, (b) browser-based lookups, and (c) keyword-based lookups. A node can use these three interfaces to locate VOs they are interested in and join these VOs.

Exact Lookups: This is the simplest form of a lookup. This type of query contains the complete directory path of the interest domain, for example *"/computer science/system/network/architecture"*. This complete directory path is hashed to a key and then a corresponding lookup of the hashed key on the Pastry overlay is executed.

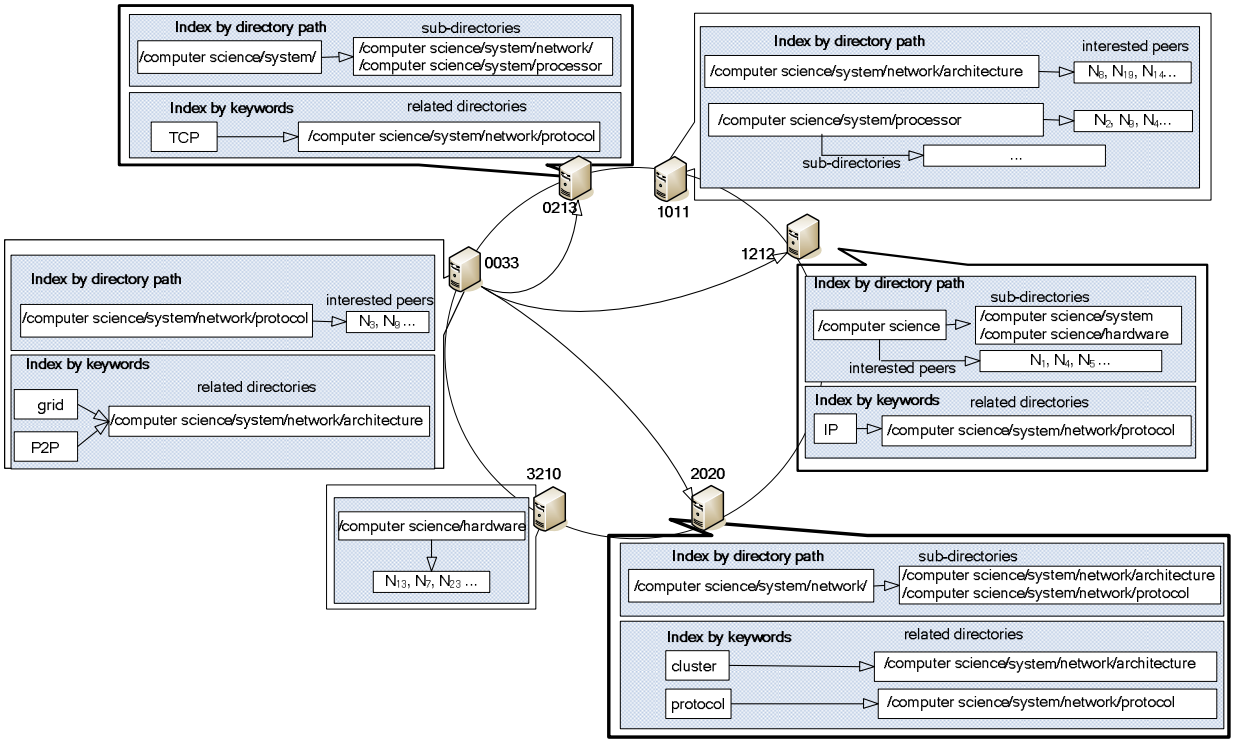


Figure 2. Storing the ontology model into a Pastry network
of 6 nodes in an example 8-bit identifier space

Browser-based Lookups: In this case, users do not need to remember the directory path to locate the directory domain of interest. Instead, they can navigate from the root of each hierarchy down to the leaves to reach the directory of interest. A user first uses the root ID as the key to locate the node storing the root of the ontology model. Because a node storing a directory entry also storing the next level children, then users can dynamically expand a directory tree node to browse its child branches. After the user chooses an interested branch, the directory path of that branch is used as a key to lookup the next level directory. In this way, the tree is expanded until users find the desired directory entries. In reality, the root and top level categories are widely cached in most of the nodes in the network; therefore they can be quickly located without going through the overlay network.

Keyword-based Lookups: Users can also specify one or more key concepts of their local ontology and use the concepts as keys to lookup the corresponding directory in the overlay. Because overlay nodes in charge of the keywords keep links to the corresponding directory entries, a keyword-based lookup can be converted into an exact lookup. When a user provides multiple keywords, each of them may correspond to multiple directories,; therefore, the intersection (or union) of all directories related to these keywords is returned to the user. Domain ontologies and/or external generic ontologies like WordNet [14] can be used for keyword semantic query expansion or keyword conceptual indexing in order to improve retrieval performance.

VO Register: Because an overlay node in charge of an ontology directory also keeps a cache storing information about nodes interested in that ontology directory, a querying node can get contacts of others sharing the same interest through this overlay node. The new node can then join the VO by connecting to those contacts. At the same time, if its ontology matches the ontology of the VO, this new node can register with the VO by adding itself to the cache of the directory overlay node; therefore, in the future, others can find it. A node with multiple interests can register with multiple VOs. There are several special cases for a node's registration: (a) If a new registering node cannot get enough contacts from the interested domain (i.e., the VO is very small), it explicitly routes to the upper- and/or lower- level categories to register and get more contacts. (b) If a node cannot find suitable categories satisfying its interest (i.e., it is the first node registering this interest), it will try to add this category by applying from an authoritative organization.

Directory Overlay Maintenance: The directory overlay nodes are also user nodes. We utilize the heterogeneity of grid nodes, and promote those stable and powerful ones to join the directory overlay. Excluding ephemeral nodes from the directory overlay avoids unnecessary maintenance costs. The maintenance of the directory overlay mainly includes adding new directory entries. We assume deleting and updating do not occur frequently. When a new joining node cannot find its category of interest, it may try to apply to create a new category. If the application is approved by the authoritative organizations in the grid, the node will create this category by hashing the directory path to an overlay node and informing the parent node to add this entry. Then it hashes each of its main key concepts in the ontology to the overlay network. A node joins the directory overlay only when three conditions are satisfied: (a) It satisfies the capacity requirements or it is powerful enough. (b) It is stable for a threshold time period. (c) The directory load balancing algorithm (which will be explained in the subsequent section) requires it to do so.

4. Semantics-based Resource Discovery in Virtual Organizations

4.1 Overview

The ontology-based model facilitates nodes in forming virtual organizations (VOs). The next task is to efficiently share and search inside VOs. Searching and sharing within VOs is still very challenging, since the heterogeneous, distributed, dynamic, and large-scale properties of the problem still exist. This section proposes an infrastructure named OntoSum for efficiently sharing and discovering resources inside VOs.

OntoSum is inspired by a widely-held belief of "small-world" pertaining to social networks, in which that any two people in the world are connected via a chain of six acquaintances (*six-degrees of separation*) [15]. OntoSum is based on the observation that query transferring in social networks is made possible by locally available knowledge about acquaintances. Because of the similarity between grid networks and social networks and the fact that human users of grid networks direct grid nodes' links, we argue that grid networks can also utilize this phenomenon to discover resources.

We draw inspiration from small-world networks [36] and organize nodes in our system to form a small-world topology, particularly from a semantic perspective. Our objective is to make the system's dynamic topology match the semantic clustering of peers. That is to say that there is a high degree of semantic similarity between peers within the clustered community. This would allow queries to quickly propagate among relevant peers as soon as one of them is reached. To construct the semantic small world network depicted above, we follow the idea of the Kleinberg experiment [16]: each node keeps many close neighbors (short-range contacts), as well as a small number of distant neighbors (long-range contacts). The distance metric in our system is determined by nodes' semantic similarity. With the semantics-based small-world constructed, a query can be efficiently resolved in the semantic cluster neighborhood through short semantic paths.

4.2 Semantic Similarity

There has been extensive research [17, 18] focusing on measuring the semantic similarity between two objects in the field of information retrieval and information integration. However, their methods are very comprehensive and computationally intensive. In this work, we propose a simple method to compute the semantic similarity between two peers.

4.2.1 Ontology Signature Set (OSS)

To measure the semantic similarity between peers, we need to extract each peer's semantic characteristics. The T-Box [19] part of ontology defines high-level concepts and their relationships like the schema of a database. It is a good abstraction of the ontology's semantics and structure. Therefore, our semantic property representation is based on T-Box knowledge. We can use keywords of a node's T-Box ontology as its ontology summary. For each node, we extract the class and property labels from its T-Box ontology, and put them into a set. This set is called this node's Ontology Signature Set (OSS). We can measure the similarity of two ontologies by comparing the elements of their OSSs. With the OSS, we summarize a node's ontology properties as a set of keywords. This summarization is simple and concise, but on the other hand, it is not precise. That is, it ignores the inherent relationships between T-Box concepts and thus damages the semantic meaning of each concept.

One improvement is to extend each concept with its semantic meanings, so that semantically related concepts would have overlaps. Based on this intuition, we use the lexical database, WorldNet [14], to extend the OSS to include words which are semantically related to the concepts from the original set. An intuitive idea of extending an OSS is to extend each concept with its synset (or its synonyms). Given a primitive OSS consisting of a number of ontology concept labels, we lookup each concept in the WordNet lexicon and extend each concept with its synonyms in the synset. In this way, two semantically

related ontologies would have common WordNet terms in their extended OSSs. Besides synonyms, WordNet also includes other lexical semantic relations, such as *is-a*, *kind-of*, *part-of*. Among these relations, *is-a* (represented by hyponym/hypernym in WordNet) is the most important relationship; it explains a concept by a more general concept. Therefore, we also extend OSS concepts with their hypernyms.

After extension, an OSS may get a large number of synonyms for each concept. However, not all of these synonyms should be included in the set, because each concept may have many senses (meanings), and not all of them are related to the ontology context. A problem causing the ambiguity of concepts in OSS is that the extension does not make use of any relations in the ontology. Since the dominant semantic relation in an ontology is the subsumption relation (super-class, the converse of is-a, is-subtype-of, or is-subclass-of), in this development phase of our system, we use the subsumption relation and the sense disambiguation information provided by WordNet to refine OSSs. It is based on a principle that a concept's semantic meaning should be consistent with its super-class's meaning. We use this principle to remove those inconsistent meanings.

We create the refined OSS by adding the appropriate sense set of each ontology concept based on the *sub-class/super-class* relationships between the parent concepts and child concepts. For every concept in an ontology, we check each of its senses; if a sense's hypernym has an overlap with this concept's parent's senses, then we add this sense and the overlapped parent's sense to the OSS set. In this way, we can refine the OSS and reduce imprecision. Besides the *is-a* relation, ontologies often include additional types of domain-specific relationships that further refine the semantics they model.

4.2.2 Peer Semantic Similarity

To compare two ontologies, we define an ontology similarity function based on the refined OSS. The definition is based on Tversky's "Ratio Model" [20], which is evaluated by set operations and is in agreement with an information-theoretic definition of similarity [21]. Our similarity function is based on the normalization of Tversky's model to give a numeric measurement of ontology similarity.

DEFINITION 2: *Assume A and B are two peers, and their extended Ontology Signature Sets are S(A) and S(B) respectively. The semantic similarity between peer A and peer B is defined as:*

$$sim(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cap S(B)| + \alpha |S(A) - S(B)| + \beta |S(B) - S(A)|}$$

In the above equations, "∩" denotes set intersection, "–" is set difference, while the notation "| |" represents set cardinality, "α" and "β" are parameters that provide for differences in focus on the different components. The similarity *sim*, between *A* and *B*, is defined in terms of the semantic concepts common to OSS of *A* and *B*: $S(A) \cap S(B)$, the concepts that are distinctive to *A*: $S(A) - S(B)$, and the features that are distinctive to *B*: $S(B) - S(A)$. The parameters α and β are non-negative, determining the relative weights of these two components. The similarity depends not only on the proportion of features common to the two ontologies but also on their unique features and the relative importance varies with

the parameters α and β. These parameters allow the model some flexibility, because it can decide whether common or distinctive features have more influence. Note that with this definition, similarity is not a symmetric relation, i.e., "how similar is *A* to *B*" may give a different answer than "how similar is *B* to *A*". Employing such an asymmetric measurement reflects human judgment: sometimes, we say one object is similar to another one, but not conversely. With the similarity measure specified, we have the following definition:

DEFINITION 3: *Two nodes, node A and node B are said to be semantically equivalent if their semantic similarity measure, sim(A,B) equals to 1 (implying sim(B,A)=1 as well). Node A is said to be semantically related to node B, if sim(A,B) exceeds the user-defined similarity threshold t (0<t≤1). Node A is semantically unrelated to node B if sim(A,B)<t.*

4.2.3 An Illustrative Example

We use an example to further illustrate how to use the refined OSS and similarity function to measure the semantic similarity between two peers. Figure 3 shows two partial ontology definitions about automobiles. Detailed ontology definitions are omitted here. Table 2 and Table 3 list the ontology concepts and their synonyms and hypernyms from all senses extracted from WordNet.

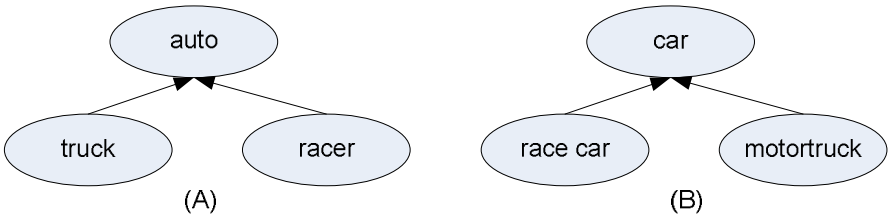


Figure 3. Parts of two ontologies

The primitive OSSs of these two ontologies are:

S_A = *{auto, truck, racer}*
S_B = *{car, race car, motortruck}*

These two sets share no common terms, and literally they are totally different. If the similarity function is applied to these two sets, the result is 0, meaning they are totally unrelated. Table 2 and Table 3 illustrate how to extend the OSSs with right WordNet senses.

By extending the two primitive OSSs*:* S_A *and* S_B, we get the extended OSSs: S_A*'* and S_B*'*:

S_A*'* = *{auto, car, automobile, machine, motorcar, truck, motortruck, racer, race car, racing car}*
S_B*'* = *{car, auto, automobile, machine, motorcar, racer, race car, racing car, truck, motortruck}*

Now we can see that these two sets share exactly the same semantic concepts! The similarity functions based on the extended OSSs are:

$$sim(A, B) = \frac{|S(A) \bigcap S(B)|}{|S(A) \bigcap S(B)| + \alpha |S(A) - S(B)| + \beta |S(B) - S(A)|} = \frac{11}{11 + 0\alpha + 0\beta} = 1$$

$$sim(B, A) = \frac{|S(A) \bigcap S(B)|}{|S(A) \bigcap S(B)| + \alpha |S(B) - S(A)| + \beta |S(A) - S(B)|} = \frac{11}{11 + 0\alpha + 0\beta} = 1$$

This means ontology A and ontology B are semantically equivalent. Note: the equivalent is independent of α and β. With the semantic similarity function defined, we can measure the semantic distance between nodes and reconfigure the network topology accordingly to form semantic small-worlds. The following section details a brief overview of our semantic small-world topology.

Table 2. WordNet senses and hypernyms for ontology A

Concept	Parent-concept	WordNet senses/synset	Hypernyms of senses in WordNet	Right sense?
auto		car, auto, automobile, machine, motorcar	motor vehicle, automotive vehicle	yes
truck	auto	truck, motortruck	motor vehicle, automotive vehicle	yes
		hand truck, truck	handcart, pushcart, cart, go-cart	no
racer	auto	race driver, automobile driver	driver	no
		racer, race car, racing car	car, auto, automobile, machine, motorcar	yes
		racer (an animal that races)	animal, animate being, beast, brute, creature, fauna	no
		racer (slender fast-moving North American snakes)	colubrid snake, colubrid	no

Table 3. WordNet senses and hypernyms for ontology B

Concept	Parent-concept	WordNet senses/synset	Hypernym of senses WordNet	Right sense?
car		auto, automobile, machine, motorcar	motor vehicle, automotive vehicle	yes
		railcar, railway car, railroad car	wheeled vehicle	no
		gondola	compartment	no
		cable car, car	compartment	no
race car	car	racer, race car, racing car	car, auto, automobile, machine, motorcar	yes
motortruck	car	truck, motortruck	motor vehicle, automotive vehicle	yes

4.3 Semantics-based Topology Adaptation

In Kleinberg's experiment [16], to form a network with small-world characteristics nodes keep many "local" contacts and one "remote" contact. Our semantic topology construction is based on this idea. In our system, a node distinguishes three kinds of neighbors based on their semantic similarity. A peer A's neighbor, B, can be one of these three types: (a) zero-distance neighbor (or semantically equivalent neighbor), if $sim(A,B)=1$, (b) short-distance neighbor (or semantically related neighbor) if $sim(A,B) \geq t$ ($0<t<1$ is A's semantic threshold), (c) long-distance neighbor (or semantically unrelated neighbor) if

sim(A,B)<t. A node always tries to find as many close neighbors as possible, but it also keeps some long distance neighbors to reach out to other ontological clusters.

Nodes in the system randomly connect to each other through these three types of neighbor links. They produce a semantically clustered small-world topology. The cluster structure is not flat but multi-layered: nodes with similar ontological topics (short-distance neighbors) form a domain. Inside each domain, nodes may create smaller clusters if they share the same ontology schema. Peers in our system may pose two kinds of queries, neighbor-discovery queries and resource-discovery queries. The neighbor-discovery query is used to construct the semantic small-world topology. When a new node joins the network, it issues neighbor-discovery queries to find semantically related neighbors, so that it can join their domains and clusters by connecting to them. The resource-discovery query is used to locate desirable resources in the network. Once the semantic topology has been created, resource discovery can be performed inside local clusters and domains. To efficiently resolve both queries, each node maintains finer-grained knowledge of neighbors semantically closer to it, but coarser-grained knowledge of neighbors further from it. This reflects the characteristic of our routing strategy, in which the query first walks around the network, and once it reaches the target cluster, it zooms in on that cluster and investigates its detailed ontology properties.

The construction of an ontology-based topology is a process of finding semantically related neighbors. A node joins the network by connecting to one or more bootstrapping neighbors. Then the joining node issues a neighbor-discovery query, and forwards the query to the network through its bootstrapping neighbors. The neighbor-discovery query routing is in fact a process of inter-cluster routing and is based on the inter-cluster routing table. A node's inter-cluster routing table stores the abstract semantic knowledge of its neighboring clusters. Specifically, it keeps contacts to those clusters – its short-distance and long-distance neighbors, their semantic similarities to this node, and their OSS mapped in a compressed Bloom filter [22]. To reconcile the semantic differences between clusters, inter-ontology mappings are also stored in the inter-cluster routing table. A query can then be forwarded to a neighbor after being translated according to the inter-ontology mapping. A neighbor-discovery query is mainly routed over clusters to quickly locate related clusters. A resource-discovery query is always forwarded inside the clusters because of the topology's semantic locality property.

To control the overhead of routing table maintenance, a soft-state update mechanism is used to keep the routing information up-to-date; nodes periodically probe their neighbors and propagate updated ontology information to them. At any given time, the resource routing information may potentially be stale or inconsistent, but in the long run they are good enough to direct query forwarding to the right peers.

A neighbor-discovery query message includes several parts: (a) the querying node's compressed OSSs, (b) a similarity threshold which is a criterion to determine if a node is semantically related to the query (optional), (c) a query Time To Live (TTL) to gauge how far the query should be propagated, (d) a list of clusters (represented by the ontology namespace of the cluster) the query has passed through, so that the query will not be forwarded to the same cluster again and again. When a node *N* receives a neighbor-

discovery query *Q* which tries to find neighbors for a new joining node *X*, *N* computes the semantic similarity between *X* and itself. If *N* is semantically related to *X*, *N* will send a *Neighbor Found* reply to *X*. If the query's TTL has not expired, *N* computes the semantic similarity between *X* and each of its neighbors, and forwards the query to semantically related neighbors. If no semantically related neighbors are found, the query will be forwarded to *N*'s long-distance neighbors.

A neighbor discovery query aims to locate short-distance and zero-distance neighbors for the querying node. Bootstrapping neighbors can be candidates for long-distance neighbors if they are not semantically related to the querying node. Information of short-distance and long-distance neighbors is used to construct a node's inter-cluster routing table. After a node finds its short-distance neighbors, it will contact them to map ontologies with them. Thereafter, queries are translated whenever passing along short-distance links.

4.4 Resource Discovery in OntoSum

With the semantic small-world topology constructed, resource discovery can be efficiently performed. In most cases, a resource discovery query can be answered within the querying node's local domain, because queries reflect the querying node's ontology interest, and semantically related nodes are within the neighborhood of the querying node. When a node issues (or receives) a query, it first chooses its zero-distance neighbors to forward the query inside the local cluster. Because they use the same ontology, the zero-distance neighbors are the best candidates to forward the query to. Another important step in query processing is to reformulate a peer's query over other peers on the available semantic paths. Starting from the querying peer, the query is reformulated over the querying peer's short-distance neighbors, then over their short-distance neighbors, and so on until the query TTL expires. Because of the small-world property, the query can get enough answers within a small number of hops with high probability. The query reformulation is according to the inter-ontology mappings. Because the ontology mapping between two clusters rarely maps all concepts in one cluster to all concepts in the other, mappings typically lose some information and can be partial or incomplete; the reformulated query may deviate from the original query's intention, and the query result should be evaluated at the querying node. Feedback on query results can be used to improve the quality of inter-ontology mappings. Moreover, nodes can learn from query results to update their neighbors. Therefore, when a node updates its semantic interests, the system is able to adjust that node's links accordingly.

Sometimes, users may want to locate resources in other semantic domains. In this case, they would first locate the related domain using the inter-cluster routing algorithm; then they can follow procedures just mentioned to process the query in that domain. The semantic domains and clusters reduce the search time and decrease the network traffic by minimizing the number of messages circulating within the domains and clusters. Inside the cluster, nodes randomly connect with their zero-distance neighbors sharing the same ontology schema. Queries looking for particular resources can be routed inside the cluster using flooding- or random-walk- based simple forwarding algorithms.

5. Evaluations

5.1 Evaluation with a Prototype System

We evaluate the operability of the presented architecture by implementing a prototype system. The system's builds on top of the available plug-ins of the Protégé and provides additional components for managing the distributed resource metadata. The OntoSum toolkit extends Protégé by adding two plug-in tabs: (a) the *ontology directory browser & VO register tab* (Figure 4) and (b) the *VO ontological query tab* (Figure 5).

We installed the OntoSum Toolkit software on six WinXP computers and six Linux SUSE computers. Each physical node runs three copies of the software and simulates three virtual nodes, therefore we have in total thirty-six nodes in the system. The toolkit prototype shows that the proposed OntoSum strategy is applicable and effective to support distributed expressive discovery in distributed systems. However, because of the limited experimental environment (with only 12 physical nodes in a LAN), we leave the scalability evaluation to simulations presented in the next section.

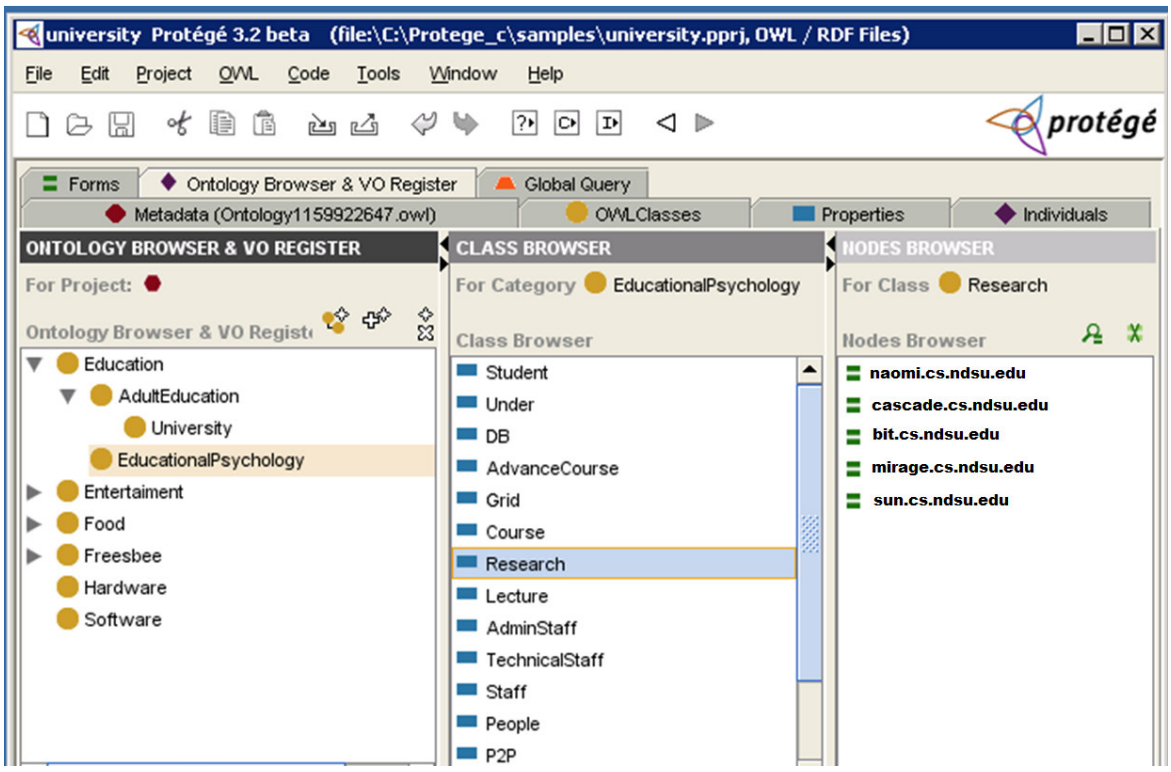


Figure 4. A screenshot of the directory browser & VO register tab

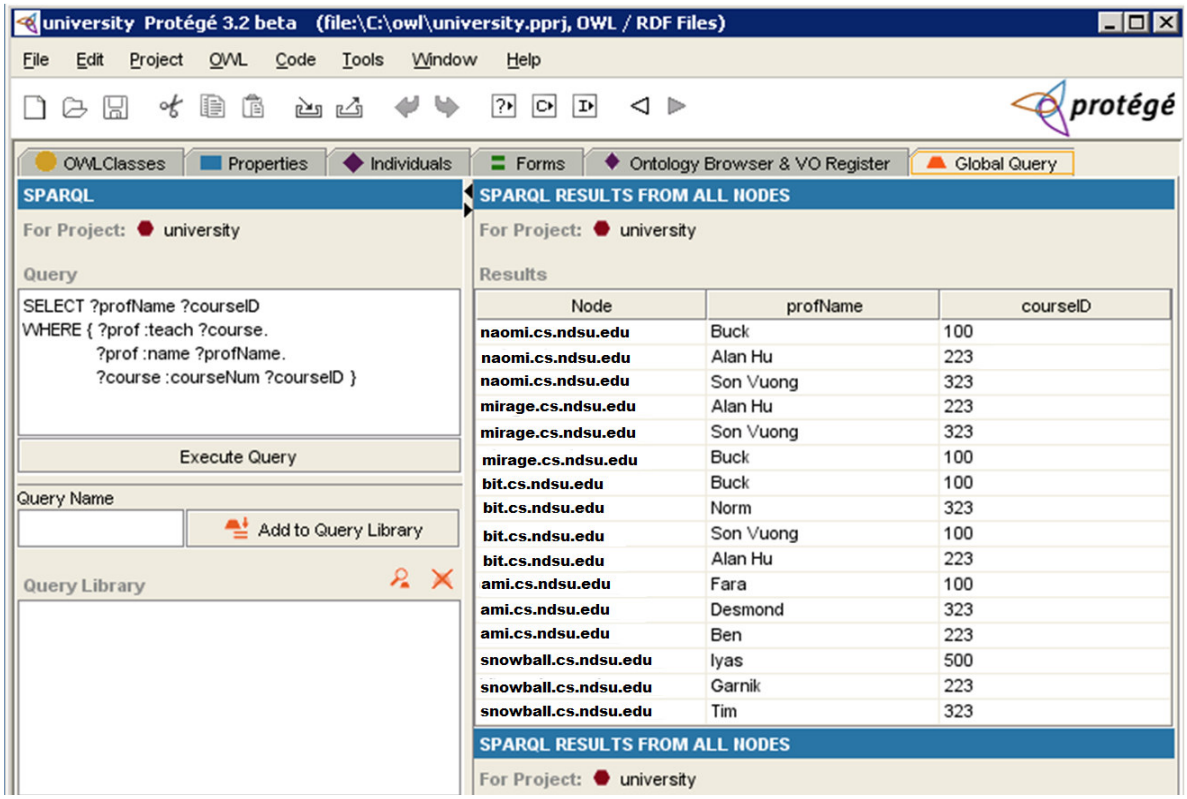


Figure 5. A screenshot of the query tab

5.2 Evaluation with Large-scale Simulations

5.2.1 Setup

The test data is artificially generated. The ontology schemas are generated first, and then individuals are created by instantiating classes. We assume for simulation purposes that ontologies and queries are associated with a specific domain, and all ontologies in the same domain have ontology mappings defined in advance. Queries were generated by randomly replacing parts of the created triples with variables. Single triple queries and conjunctive triple queries are used as the representative query format in this experiment.

The simulation is initialized by injecting nodes one by one into the network until a certain network size has been reached. The network topology created this way has power-law properties; nodes inserted earlier have more links than those inserted later. This property is consistent with the real world situation, in which nodes with longer session time have more neighbors. After the initial topology is created, a mixture of joins, leaves, and queries are injected into the network based on certain ratios. The proportion of join to leave operations is kept the same to maintain the network at approximately the same size. Inserted nodes start functioning without any prior knowledge.

For comparisons, we simulate our searching scheme OntoSum in conjunction with the learning-based ShortCut scheme [23] and a random-walk based simple Gnutella scheme [24]. The ShortCut approach is chosen as one comparison reference since it is simple yet effective, and many popular applications (e.g.,

[23], [25], [26], [27]) use this approach as their basic routing scheme. Moreover, it is comparable to our approach in the sense that it creates clusters on top of the unstructured network. The ShortCut approach relies on the presence of interest-based locality to create "shortcuts". Each peer builds a shortcut list of nodes that answered previous queries. To find content, a peer first queries the nodes on its shortcut list and only if unsuccessful, floods the query. This approach presents a promising reorganization method within unstructured P2P networks. Flooding-based Gnutella was chosen as another reference approach for its simplicity and prevalence, which, in fact, made it a widely used baseline for many of the previous research efforts.

The resource-discovery query is propagated exponentially, i.e., each node chooses a certain number of neighbors (called walkers) to forward the query. The neighbor-discovery query (for OntoSum only) is propagated linearly, i.e., only the node that issues the query forwards the query to a certain number of walkers, while all other nodes only forward the query to one neighbor. In the rest of the book chapter, we use the term "query" to refer to resource-discovery query.

The simulation parameters and their default values are listed in Table 4.

Table 4. Parameters used in the simulations

Parameter	**Range and default value**
network size	2^9~2^{15} default: 10,000
initial neighbors (node degree)	5
maximum neighbors	30
average node degree	14
TTL	1~20 default 9
resource-discovery query walkers	3 (propagate exponentially)
neighbor-discovery query walkers	2 (propagate lineally)
ontology domains	1~10 default: 8
ontology schemas per domain	1~10 default:8
distinct resources per domain	100
resources per node	1~10
die/leave probability per time slice per node	0-21%, 3% default
resource change probability per time slice per node	20%instance update, 2% schema update
query probability per time slice per node	5%
sample of nodes to compute diameter	5%

5.2.2 Results

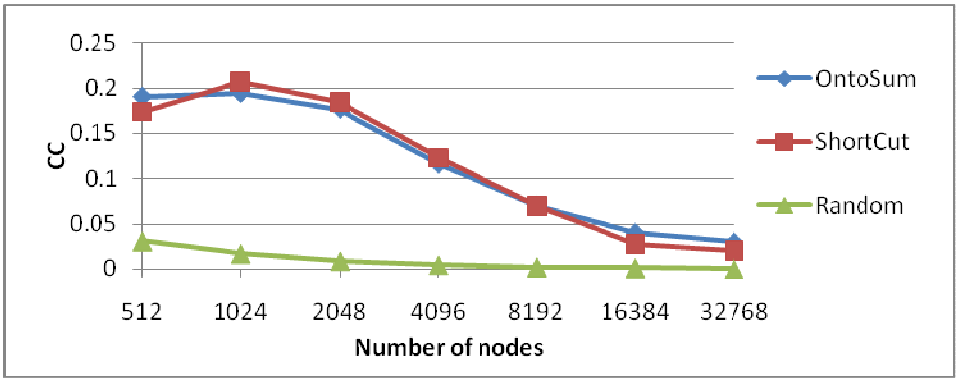
5.2.2.1 Emergence of the Small-world

We expect that the OntoSum semantic neighbor discovery scheme will transform the topology into a small-world network. To verify this transformation, we examine two network statistics, the *clustering*

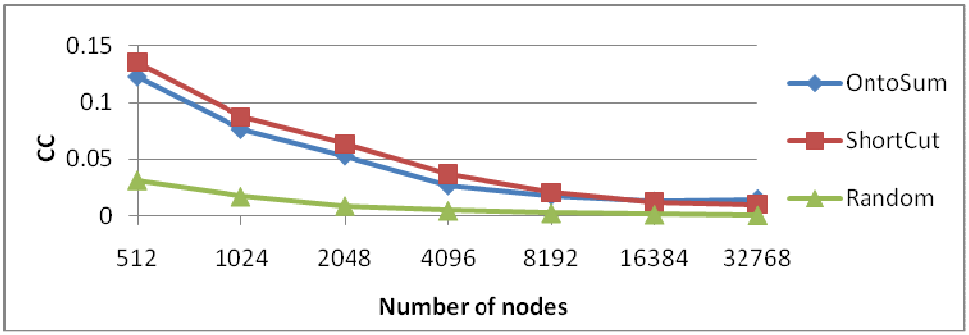
coefficient and the *average network path length*, as indicators of how closely the topology has approached a "small-world" topology. The *clustering coefficient* (*CC*) is a measure of how well connected a node's neighbors are with each other. The *CC* of a node is the ratio of the number of existing edges and the maximum number of possible edges connecting its neighbors. The average over all |V| nodes gives the *CC* of a graph. The *average path length (APL)* is defined as the average shortest path across all pairs of nodes. The *APL* corresponds to the degree of separation between peers. For a large graph, measuring distances between all node pairs is computationally expensive; therefore an accepted procedure is to measure it over a random sample of nodes [28]. In our experiment, we use a random sample of certain percent of the graph nodes. We use Dijkstra's algorithm to compute the shortest distance between pairs of nodes. In our simulated topology, we intentionally make the network strongly connected, so that any pairs of nodes have a directed path.

We performed experiments to measure OntoSum's *cluster coefficient* (*CC*) and *average path length (APL*). An interest-based ShortCut topology and a random power-law topology with the same average node degree are used as reference topologies. The former has been proved to be a small-world system [29]. For the ShortCut scheme, test results are collected after the system has had an extensive training process, i.e., nodes have learned as many ShortCuts as possible through query results and the system topology has become stable.

Figure 6 and Figure 7 show plots of the *clustering coefficient* and the *average path length* as a function of the number of nodes in the network. The system has two configurations; in Figure 6 (a) and 7 (a), nodes have more ontologies to choose from, while in Figure 6 (b) and 7 (b), nodes have fewer ontological domains. We observe that both the *clustering coefficient* and the *average path length* of OntoSum are very similar to those of ShortCut. The *clustering coefficients* of OntoSum and ShortCut are much larger than that of the random power-law network, while the *average path length* of OntoSum and ShortCut are almost the same as that of the random network. This indicates the emergence of a small-world network topology [28]. We must note that because all of the three topologies that are created by inserting nodes to the existing system, all topologies show the power-law property to some extent, and thus the *average path length* of all three topologies are smaller than a random network. This set of experiments verifies that firstly, well connected clusters exist in the OntoSum system; due to the semantic similarity definition, these clusters correspond to groups of users with shared ontological interests. Secondly, there is, on average, a short path between any two nodes in the system topology graph; therefore, queries with relatively small TTL would cover most of the network. Our later simulation experiments will verify this.

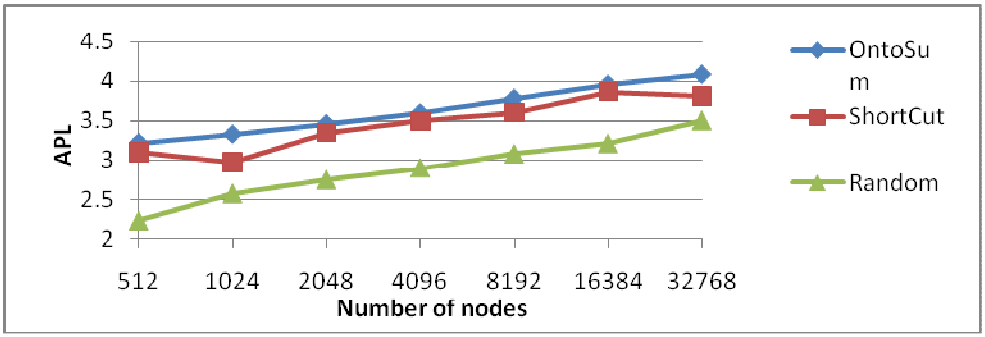


(a) With 10 ontological domains, and 5-10 ontologies per domain

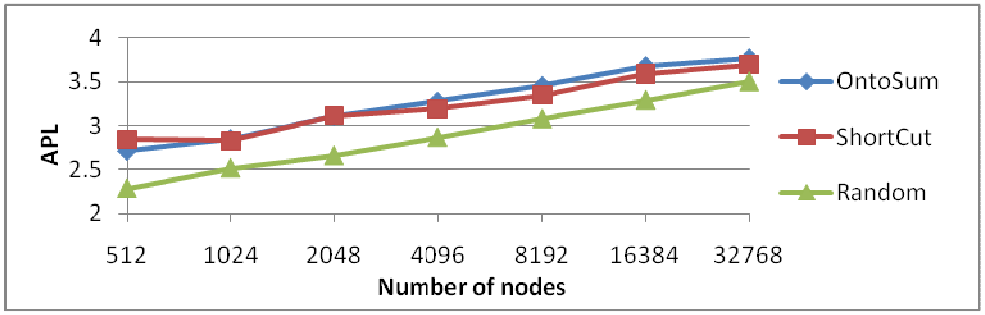


(b) With 4 ontological domains, and 2-4 ontologies per domain

Figure 6. Comparison of clustering coefficient



(a) With 10 ontological domains, and 5-10 ontologies per domain



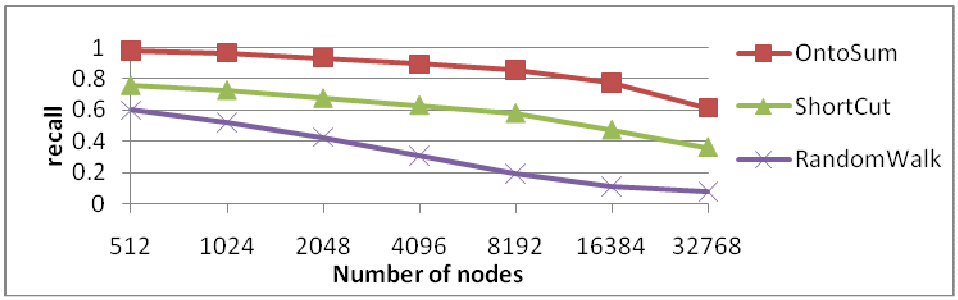
(b) With 4 ontological domains, and 2-4 ontologies per domain

Figure 7. Comparison of average path length

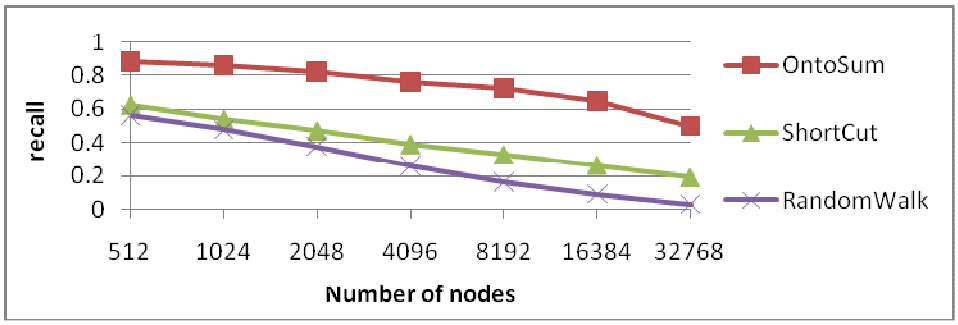
5.2.2.2 Scalability and Efficiency

We examine the system performance in three different aspects, namely routing scalability, efficiency, and accuracy by executing the experiment in different network configurations. The performance is measured using the metric of recall rate, which is defined as the number of results returned divided by the number of results actually available in the network. For comparison, we also implement the learning-based ShortCut algorithm and random-walk based Gnutella algorithm. For the ShortCut approach, we collect query results after sufficient learning has been done. To simulate dynamic factors, in each time slice every node has a *5* percent probability to issue a query, and a *2* percent probability to leave the system. The probability of new nodes with new resources joining the system is the same as the probability of a node leaving. First, we vary the number of nodes from 2^9 to 2^{15} to test the scalability of the routing scheme. The results are listed in Figure 8. As we expected, OntoSum gets higher recall in all these different sized networks and in both static and dynamic environments. In addition, OntoSum's recall decreases less with the increase in network size. Figure 9 illustrates the system efficiency by showing the relationship between query recall rate and query TTL. With a small TTL, OntoSum gets a higher recall rate than the other two algorithms. This means that OntoSum resolves queries faster than the others. In Figure 10, we show the effect of dispatching a different number of walkers to search the network. We can see that with the same TTL, OntoSum locates more results with fewer walkers. This indicates that OntoSum routing is more accurate and can always find the right node to forward the query to. From Figures 8, 9 and 10, we also notice that OntoSum performs better than ShortCut and random-walk in both static and moderately dynamic environments.

As expected, our OntoSum searching scheme performs well as measured by recall rate in both static and dynamic networks. OntoSum's small-world topology effectively reduces the search space, and its ontology summary guides the query in the right direction. Therefore, OntoSum can locate results faster and more accurately. This explains why OntoSum scales to large network size and why it achieves higher recall with shorter TTL and fewer walkers. Besides all these reasons, another factor contributing OntoSum's overall better recall rate is that OntoSum is able to locate semantically related results that cannot be located by the ShortCut and random-walk. Because of the semantic heterogeneity of our experimental setup, relevant resources may be represented with different ontologies. OntoSum may use its ontology signature set to find semantically related nodes and use the mapping defined to translate the query. Therefore, it can locate most of the relevant results. However, for ShortCut and random-walk, they have no way to find semantically related resources. Therefore, they can only locate resources represented in the same ontology as the ontology of the querying node.

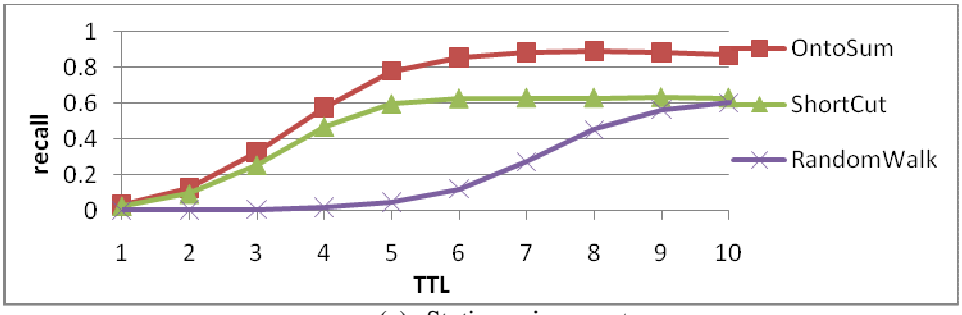


(a) Static environment

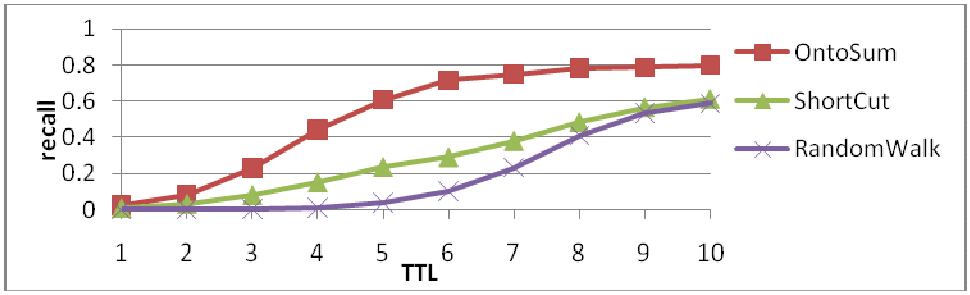


(b) Dynamic environment

Figure 8. Recall rate vs. network size

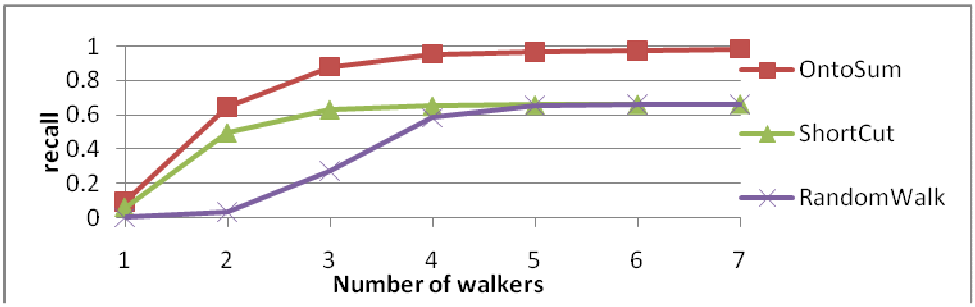


(a) Static environment

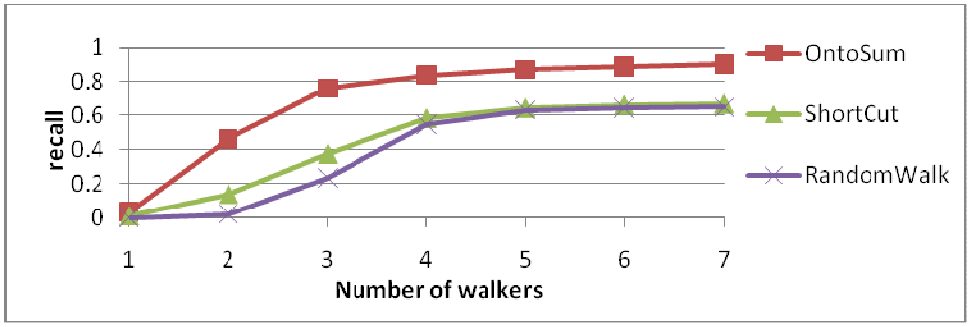


(b) Dynamic environment

Figure 9. Recall rate vs. TTL (with # walkers=3)



(a) Static environment



(b) Dynamic environment

Figure 10. Recall rate vs. walkers (with TTL=5)

5.2.2.3 Overhead and Adaptability to Dynamics

System overhead has a close relation with the system dynamics, as a system must maintain consistent information about peers in the system in order to operate most effectively. Therefore, we measure the system dynamics together with the overhead. To evaluate the adaptability to different levels of dynamics, we measure the system overhead under different levels of peer "churn rate" and "update rate", referring to the rate of peers leaving/joining the system and the rate of resource updates. Experiments in this section are performed on a 10,000-node network. The churn rate is represented as the probability for a node to die/leave the system in unit time slice; to maintain the constant number of network size we also insert an equal number of new nodes into the system. The update rate is the probability for a node to update its resource information in a time slice.

The experiment shown in Figure 11 gives an overview of how dynamics affect the system performance. Specifically, it shows the query recall rate under different dynamic configurations. In the experiment, we increase the dynamics by increasing the churn rate. From the figure, we find that OntoSum performs similarly to the ShortCut algorithm which is proved to be resilient to churn [30]. When peers join or leave frequently, the performance of ShortCut and OntoSum deteriorate gracefully. Churn does not affect the two schemes dramatically. Their unstructured random topologies provide multiple routes to a destination thus increasing the system resilience. In the worst case, they degrade to random-walk.

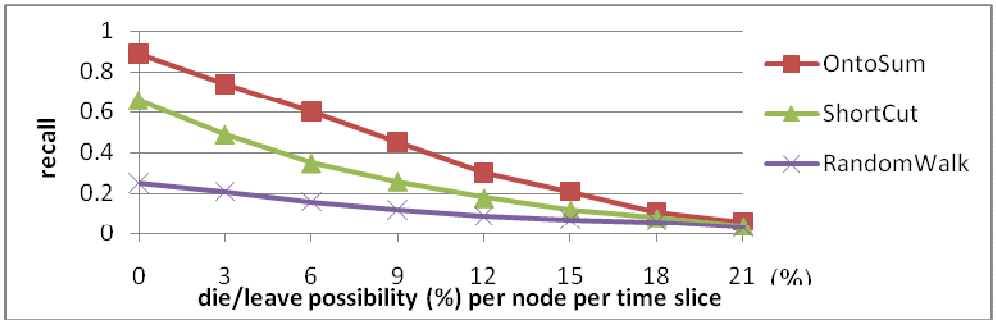


Figure 11. Recall vs. churn rate

Figure 12 shows the accumulated bandwidth overhead of finding 10000 results under different churn rates. We use a soft state approach to update the routing table: the routing table is updated periodically instead of in real time. From the figure, we can see that in most situations OntoSum produces much less overhead then the other two methods. But when the system is very dynamic, such as when the dying probability is beyond 20%, OntoSum produces much more overhead. When the system is very dynamic, the neighborhood relationship changes frequently, and OntoSum creates great amounts of overhead maintaining its routing table. Even worse, the overwhelming maintenance overhead does not bring much benefit in this situation, because the newly constructed topology will change quickly. Luckily, churn of the nature described above rarely happens in reality [31], and we can see from Figure 12 that with this churn rate, ShortCut degrades to random-walk. The high overhead problem of OntoSum in very dynamic environments can be solved by a simple solution: when the network is very dynamic, the system can give up the ontology-based topology construction and routing and resort to basic Gnutella random-walk as the solution.

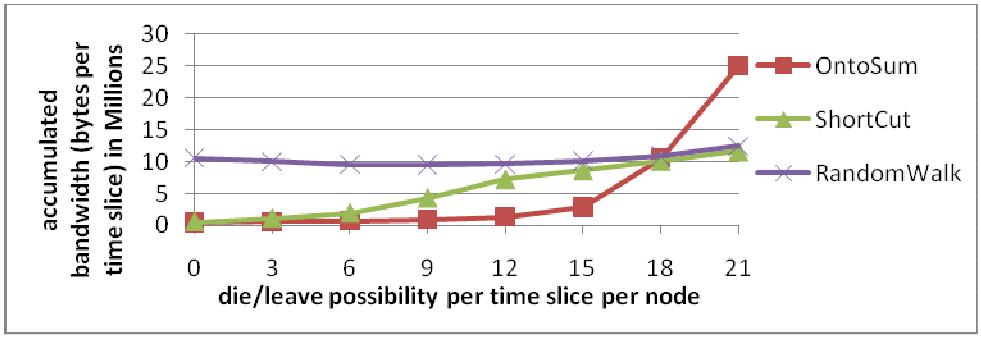


Figure 12. System overhead vs. churn rate

6. Conclusions and Future Directions

This book chapter focused on the resource discovery in a large-scale grid environment. We demonstrated that it is possible to meet both the scalability and searchability challenges faced by the resource discovery problem in this target environment. To accomplish the aforementioned, we designed, implemented, and evaluated distributed discovery systems that are fully decentralized, scalable to the number of users and resources involved, adaptive to heterogeneous resource representations, and capable of handling complex queries.

Important problems in large-scale resource discovery remain unsolved. We identify several limitations of our work and research directions for future work. Some of the research directions are a natural continuation of this chapter - others are more general problems in resource discovery.

Our search system focuses on relatively static resource information; however, sometimes we need to consider very dynamic information. For example, in computational grids, a scheduler may need to find available computational resources with both relatively static requirements, such as system architecture,

OS version, and access policy, and more dynamic requirements, such as instantaneous load and predictions of future availability.

In OntoSum, finding semantically related neighbors is accomplished according to their semantic similarity, which is defined by comparing the extended Ontology Signature Set. This simple similarity can be improved by considering other factors such as nodes' ontological structure, definitions of concepts, and instances of classes.

In our current system, query results are returned to requesters without using any ranking mechanisms. There are many techniques for ranking entities on the Web, for example PageRank 32] and HITS [33], on XML documents [34], and on the Semantic Web [35]. However, these techniques cannot be used directly to rank our search results because of the different problem nature. We plan to investigate the result-ranking problem, so that query results can be ordered based on relevance and importance for users. The ranking problem involves a rich blend of semantic and information-theoretic techniques. The ordering of the results should be able to vary according to user need.

References

[1] H. Casanova, "Distributed Computing Research Issues in Grid", Computing, typescript, Univ. of California, San Diego, 2002.

[2] Globus Toolkit: http://www.globus.org/toolkit/.

[3] M. Cai, M. Frank, J. Chen and P. Szekely, "MAAN: A Multi-Attribute Addressable Network for Grid Information Services." The 4th International Workshop on Grid Computing, 2003.

[4] A. Iamnitchi, I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," in Proceeding of the 2nd IEEE/ACM International Workshop on Grid Computing 2001, Denver, 2001.

[5] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. "A Directory Service for Configuring High-Performance Distributed Computations." In Proceedings of the 6th IEEE Symp. on High-Performance Distributed Computing, pages 365–375. IEEE Computer Society, 1997.

[6] W. Smith and D. Gunter. "Simple LDAP Schemas for Grid Monitoring." Global Grid Forum, GWD-Perf-13-1, June 2001.

[7] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. "The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration." Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[8] R. Raman, M. Livny, M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing". In Proceeding of IEEE Intel. Symp. On High Performance Distributed Computing, Chicago, USA, 1998.

[9] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. “The Legion resource management system.” In Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, San Juan, Puerto Rico, 1999.

[10] Aloisio, G.; M. Cafaro; S. Fiore; M. Mirto; and S. Vadacca. “GRelC Data Gather Service: a Step Towards P2P Production Grids”, In Proceedings of 22nd ACM Symposium on Applied Computing, 2009.

[11] DMOZ Open Directory Project. http://www.dmoz.org/

[12] DHT: http://en.wikipedia.org/wiki/Distributed_hash_table

[13] A. Rowstron and P. Druschel. “Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems,” In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Middleware, 2001.

[14] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross and K. J. Miller. “Introduction to WordNet: an on-line lexical database.” International Journal of Lexicography, 3(4):235-312. 1993.

[15] A. L. Barabási, “Linked: How Everything is Connected to Everything Else and What It Means for Business, Science, and Everyday Life.” New York, Plume, 2003.

[16] J. Kleinberg, “Navigation in a small world,” Nature, no. 406, p. 845, 2000.

[17] M. A. Rodriguez, M. J. Egenhofer, "Determining Semantic Similarity Among Entity Classes from Different Ontologies". IEEE Transactions on Knowledge and Data Engineering, VOL. 15, NO. 2, MARCH/APRIL, 2003.

[18] J. Lee, M. Kim, and Y. Lee, “Information Retrieval Based on Conceptual Distance in IS-A Hierarchies,” J. Documentation, vol. 49, pp. 188-207, 1993.

[19] OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-features/

[20] A. Tversky. “Features of similarity.” Psychological Review, 84(4):327–352, 1977.

[21] D. Lin. “An information-theoretic definition of similarity.” In Proceeding of the 15th International Conf. on Machine Learning, pages 296–304. San Francisco, CA, 1998.

[22] B. Bloom. “Space/time tradeoffs in hash coding with allowable errors”. Communications of the ACM, pages 13(7):422-426, July 1970.

[23] X. Tempich, S. Staab, A. Wranik, “REMINDIN: semantic query routing in peer-to-peer networks based on social metaphors” in Proceedings of the International World Wide Web Conference (WWW), New York, USA, 2004.

[24] Gnutella website. http://gnutella.wego.com/

[25] K. Sripanidkulchai, B. Maggs, and H. Zhang. “Efficient content location using interest-based locality in peer-to-peer systems,” In Proceedings of the INFOCOM’03, 2003.

[26] S. Castano, A. Ferrara, S. Montanelli, and D. Zucchelli. “Helios: a general framework for ontology-based knowledge sharing and evolution in P2P systems.” In Proceeding of IEEE DEXA WEBS 2003 Workshop, Prague, Czech Republic, September 2003.

[27] A. Castano, S. Ferrara, S. Montanelli, E. Pagani, G. Rossi, "Ontology addressable contents in P2P networks." in Proceedings of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing, 2003.

[28] D. Watts and S. Strogatz. "Collective dynamics of "small-world" networks." - Nature, 1998

[29] A. Iamnitchi, M. Ripeanu, and I. Foster. Small-world filesharing communities. In Proceedings of the Infocom, Hong Kong, China, 2004.

[30] X. Tempich, S. Staab, A. Wranik, "REMINDIN: semantic query routing in peer-to-peer networks based on social metaphors" in Proceedings of the International World Wide Web Conference (WWW), New York, USA, 2004.

[31] D. Stutzbach and R. Rejaie. "Understanding churn in peer-to-peer networks." In Proceeding of the Internet Measurement Conference (IMC), Oct. 2006.

[32] S. Brin, L. Page, "The anatomy of a large-scale hypertextual Web search engine." In Proceeding Of WWW1998, pages 107-117. Brisbane, Australia, 1998.

[33] J. Kleinberg, "Athorative sources in a hyperlinked environment." J. ACM, 48:604-632, 1999.

[34] V. Hristidis, Y. Papakonstantinou, A. Balmin, "Keyword Proximity Search on XML Graphs." In Proceedings of the IEEE ICDE, 2003.

[35] A. Sheth, Aleman-Meza, B., Arpinar, I. B., Halaschek, C., Ramakrishnan, C., Bertram, C., Warke, Y., Avant, D., Arpinar, F. S., Anyanwu, K., Kochut, K. "Semantic Association Identification and Knowledge Discovery for National Security Applications." Journal of Database Management, 16 (1), pp. 33-53, Jan-Mar 2005.

[36] A. L. Barabási, "Linked: How Everything is Connected to Everything Else and What It Means for Business, Science, and Everyday Life." New York, Plume, 2003.