Volume 26, issue 3, March 2010
ISSN 0167-739X
ELSEVIER
THE INTERNATIONAL JOURNAL OF
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
GRID COMPUTING AND eSCIENCE
Editor-in-Chief:
Peter Sloot
Advisory Editors:
Carl Kesselman
Hai Zhuge
Rajkumar Buyya
Marian Bubak
Available online at www.sciencedirect.com
ScienceDirect

Future Generation Computer Systems 26 (2010) 361–373

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Grid resource discovery based on semantically linked virtual organizations

Juan Li

Computer Science Department, North Dakota State University, United States

ARTICLE INFO

Article history:
Received 19 January 2009
Received in revised form
5 July 2009
Accepted 24 July 2009
Available online 12 August 2009

Keywords:
Semantic Link Network
Grid
Discovery

ABSTRACT

Locating desirable resources and information from a large-scale grid is challenging due to the considerable diversity, large number, dynamic behavior, and geographical distribution of the resources. In this paper, we propose an efficient discovery framework which organizes a grid network by a semantically linked overlay representing the semantic relationships between grid participants. Specifically, we use a semantics-aware topology construction method to group similar nodes to form a semantic small-world. With the small-world topology constructed, resource-discovery queries will be propagated only between semantically related nodes, which greatly improves the efficiency and accuracy of resource discovery in grids. Moreover, we propose a novel algorithm for efficient resource information integration and searching over the semantic small-worlds. Our experiments with simulations substantiate that this framework significantly improve the search expressiveness, efficiency, scalability, and precision.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Grid computing is a virtualized distributed computing environment aimed at enabling the sharing of geographically distributed resources. Grid resources have traditionally consisted of dedicated super-computers, clusters, or storage units. With the present ubiquitous network connections and the growing computational and storage capabilities of modern everyday-use computers, more resources such as PCs, 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 resources in the grid, effective resource discovery mechanisms are required. However, resource discovery in large-scale semantic grids is very challenging due to the potentially 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.

The provision of an *information service* [1], 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 [2]. For example, classical grids always use centralized or static hierarchical models to discover resources. The Globus Toolkit [3] is a famous example. Globus users can get a node's resource information by directly querying a server application running on that node, or querying dedicated information servers that retrieve and publish the resource information of the organization. Although interactions between these information servers are supported, the general-purpose decentralized service discovery mechanism is still absent. 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., [4,5]). 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 semantic 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 work needs to be done 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.

This paper seeks to provide a general solution to the above-mentioned problem. It proposes a framework to share and discover resources on an unprecedented scale, and for geographically distributed groups to work together in ways that were previously impossible. To get enlightened by the recently proposed Semantic Link Network (SLN) [6,7], we propose a distributed semantics-based discovery framework. We make use of SLN model to enable rich semantic representation, reasoning, execution, and to construct a "semantic small-world" structure—OntoSum. OntoSum is based on the observation that query transferring in social networks

E-mail address: j.li@ndsu.edu.

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.07.011

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. Peers in OntoSum use their ontology summary to represent their expertise; they learn and store knowledge about other peers with a view to their potential for answering prospective queries. This way, the network topology is reconfigured with respect to peers' semantic properties. This reconfiguration partitions the large unorganized search space into multiple well-organized semantically related sub-spaces, which 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 resource-integrating and routing algorithm RDV (representing for Resource Distance-Vector-based), in which resource semantic metadata is decomposed into different coarse-grained elements, and then these elements are indexed with different schemes to improve scalability. We evaluate the performance of our system with extensive simulation experiments, the results of which confirm the effectiveness of the design.

The remainder of this paper is organized as follows: Section 2 presents the concept, property and construction of a semantic small-world architecture—OntoSum. Section 3 explains how resource discovery is performed in OntoSum. Section 4 proposes a comprehensive semantics-based query routing algorithm, RDV, which works as an improved routing algorithm to forward queries inside OntoSum clusters. Simulation experimental results are given in Section 5. Related work and concluding remarks are provided in Sections 6 and 7, respectively.

2. Semantic small-world

A widely-held belief pertaining to social networks is that any two people in the world are connected via a chain of six acquaintances (*six-degrees of separation*) [8]. The famous Milgram's experiments illustrated that individuals with only a local knowledge of the network (i.e., their immediate acquaintances) may successfully construct acquaintance chains of short length, leading to networks with "small-world" characteristics. Small-world networks exhibit special properties, namely, a small average diameter and a high degree of clustering. A small diameter corresponds to a small separation between peers, while a high clustering signals tight communities. Small world graphs contain inherent community structure, where similar nodes are grouped together in some meaningful way. Intuitively, a network satisfying the small-world properties would allow peers to reach each other via short paths while maximizing the efficiency of communication within the clustered communities.

We draw inspiration from small-world networks 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, i.e., there is a high degree of semantic similarity between peers within the clustered community; this would allow queries to be quickly propagated 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 [9]: 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.

2.1. Semantic basics

A major focus of our discovery solution is to provide an intelligent semantic search to overcome the problem of traditional keyword-based search. We employ ontology domain knowledge and SLN to assist in the search process, so that queries can be properly interpreted according to their meanings in a specific domain with the inherent relations between concepts also being considered.

2.1.1. Ontology-based metadata representation

Metadata, the data about data, is a crucial element of a discovery infrastructure. Effective metadata requires shared representations of knowledge as the basic vocabulary from which metadata statements can be asserted. An ontology, "a shared and common understanding of a domain" [10], is precisely intended to convey that kind of shared understanding. Therefore, we use ontologies to represent resource metadata semantics. An ontological representation defines concepts and relationships. To cope with the openness and extensibility requirements, we adopt two W3C recommendations: the Resource Description Framework (RDF) [11] and the Web Ontology Language (OWL) [12] as our ontology language. We concentrate on RDF's property of making statements about resources in the form of *subject–predicate–object* expressions, called *triples* in RDF terminology. The *subject* denotes the resource which has a Universal Resource Identifier (URI). The *predicate* denotes traits or aspects of the resource and expresses a relationship between the *subject* and the *object*. The *object* is the actual value, which can either be a resource or a literal. The concept of triple is very important in our work, because our metadata indexing scheme is based on this triple representation.

In our system the ontology knowledge is represented by OWL-DL and is separated into two parts: the terminological box (T-Box) and the assertion box (A-Box) as defined in the description logic terminology. The purpose of distinguishing between the T-Box and A-Box is to enable different coarse-grained indexing based on these two cases. The T-Box is a finite set of terminological axioms, which includes all axioms for concept definition and descriptions of domain structure. The A-Box is a finite set of assertional axioms, which includes a set of axioms for the descriptions of concrete data and relations. Separating the T-Box and A-Box enables different coarse-grained knowledge indexing, thus increasing the scalability of the system.

2.1.2. Semantic links

For many reasons, different people and organizations tend to use different ontologies. Therefore, we have to deal with situations where various local ontologies developed independently are required to be integrated as means for extracting information from the local ones. Semantic links provides a layer from which several ontologies could be accessed and hence information could be exchanged in a semantically sound manner. Semantic links relates the entities of two ontologies that share the same domain of discourse in such a way that the logical structure and the intended interpretations of the ontologies are respected. We adopt the semantic links defined by Zhuge in [13]. In particular, a semantic link between two ontologies can be one of the following types:

1. *Equal-to*, denoted by $Pi{-}equ \rightarrow Pj$, says that Pi is semantically equal to Pj. The equal-to link is reflexive, symmetric, and transitive.

2. *Similar-to*, denoted by $Pi - (sim, sd) \rightarrow Pj$, says that Pi is semantically similar to Pj to the degree sd.

3. *Reference*, denoted by $Pi - ref \ \rightarrow \ Pj$, says that Pi refers semantically to Pj.

4. *Implication*, denoted by $Pi - imp \ \rightarrow \ Pj$, says that Pi semantically implies Pj. The implication link is transitive and can

```
/* This algorithm generates the refined Ontology Signature Set
OSS for an ontology O */

createOss(Ontology O)
{
   OSS={};
   for each c ∈{concepts of ontology O}
      $p_c$ is parent concept of c
      add c, $p_c$ to OSS
      for each $S_c$∈{senses of c}
          $H_c$={hypernyms of $S_c$}
          for each $Sp_c$∈{senses of $p_c$)
            if $H_c$ • $Sp_c$ !=null
               add $S_c$,$Sp_c$ to OSS
```

Fig. 1. A refined algorithm to generate the Ontology Signature Set.

help the reasoning mechanism to find new semantic implication relationships.

5. *Subtype*, denoted by $Pi - st \rightarrow Pj$, says that *Pj* is semantically a part of *Pi*. The subtype link is transitive.

6. *Sequential*, denoted by $Pi - seq \rightarrow Pj$, says that the content of *Pj* is the successor of the content of *Pi* in a context.

Reasoning rules based on the semantic links are defined in [13].

2.2. Semantic similarity

There has been extensive research [14–16] focusing on measuring the semantic similarity between two objects in the field of information retrieval and information integration. However their methods are very complicated and computationally intensive. In this paper, we propose a simple method to compute the semantic similarity between two peers.

2.2.1. Ontology signature set (OSS)

To measure the semantic similarity between peers, we need to extract each peer's semantic characteristics. The representation of these characteristics should be lightweight, so that they can be efficiently exchanged between peers and the similarity based on these characteristics can be easily computed. As mentioned, the T-Box part of an 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. A naïve approach is to 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. This summarization is simple and concise, but on the other hand, it is not precise; it ignores the inherent relationships between T-Box concepts and thus damages the semantic meaning of each concept. A semantic meaning may be represented by different labels in different ontologies, while it is also possible that the same literal label in different ontologies means totally different things. Therefore, two semantically equivalent ontologies may have totally different OSSs, while two similar OSSs may represent two completely different ontologies. Ontology comparison based on primitive OSSs may not yield satisfying results.

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 [17], to extend the OSS to include words which are semantically related to the concepts from the original set. WordNet maps word forms in word senses using the syntactic category as a parameter. Words of the same syntactic category that can be used to express the same meaning are grouped into a single synonym set, called *synset*. An intuitive idea of extending an OSS is to extend each concept with its synset, i.e., 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, we also extend OSS concepts with their hypernyms (is_a relationship).

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. Having unrelated senses in the OSS will diminish the accuracy of measuring the ontology difference and incur higher computation cost for set operations. Therefore, we have to prune the expanded OSS to exclude those unrelated terms. We utilize relations between the concepts in an ontology to further refine the semantic meaning of a particular concept. Only words with the most appropriate senses are added to the OSS. 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. The refined algorithm to generate the OSS is illustrated with the pseudocode in Fig. 1.

The algorithm in Fig. 1 creates 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. The complexity of this algorithm is $N \times a^2$ where N is the number of T-Box concepts of an ontology, a is a constant representing the average number of senses of a concept in WordNet. The major cost of this algorithm is for storing and searching the WordNet dictionary.

2.2.2. 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" [18], which is evaluated by set operations and is in agreement with an information-theoretic definition of similarity [19]. Our similarity function is based on the normalization of Tversky's model to give a numeric measurement of ontology similarity.

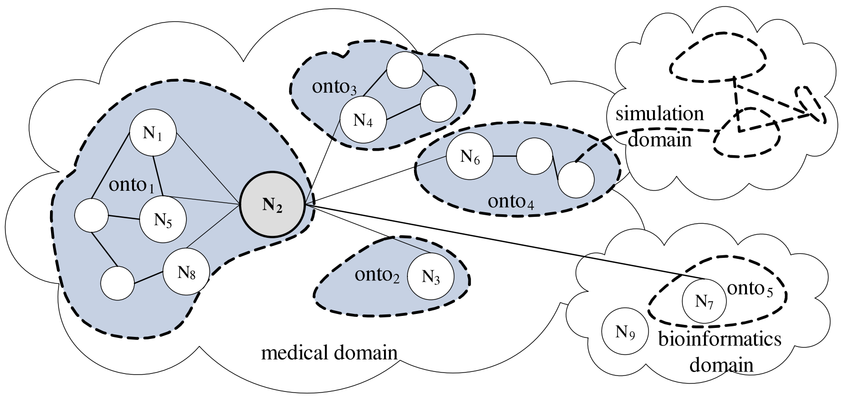


Fig. 2. A sample network topology.

Definition 1. 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:

$$sim(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cap S(B)| + \alpha|S(A) - S(B)| + \beta|S(B) - S(A)|}. \quad (1)$$

In the above equations, "∩" denotes set intersection, "–" is set difference, while "| |" 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. With the similarity measure specified, we have the following definition:

Definition 2. Two nodes, node A and node B are said to be semantically equivalent if their semantic similarity measure, $sim(A, B)$ equals 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 ($0 < t \leq 1$). Node A is semantically unrelated to node B if $sim(A, B) < t$.

2.3. Small-world topology adaptation

In Kleinberg's small-world experiment [9], 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: (1) zero-distance neighbor (or semantically equivalent neighbor), if $sim(A, B) = 1$, (2) short-distance neighbor (or semantically related neighbor) if $sim(A, B) \geq t$ ($0 < t < 1$ is A's semantic threshold), (3) 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 the domain, nodes may create smaller clusters if they share the same ontology schema. Fig. 2 shows a high level view of a sample network topology. All peers in the medical domain are interested in information related to medicine. They may be interested in different aspects of the medical resources, and they may use different ontologies to describe their resources. They connect with each other through short-distance links. Inside the medical domain, nodes further organize themselves to finer-grained clusters based on their ontologies. For example, nodes N_1, N_2, N_5, and N_8 use the same ontology, $onto_1$ (e.g., a medical ontology, SNOMED-RT [20]), thus they are zero-distance neighbors and form the same-ontology cluster. In the rest of this paper, we use the term "domain" to represent a group of clusters sharing similar ontological topics, and use the term "cluster" to denote the ontologically equivalent cluster. Clusters and domains do not have fixed boundaries; they are formed by randomly connecting relevant nodes.

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 a finer-grained knowledge of neighbors semantically closer to it, but a 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.

2.3.1. Inter-cluster routing table

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 and forwards the neighbor-discovery query to the network through its bootstrapping neighbors. We can use popular P2P bootstrapping mechanisms, such as Web Cache, previously stored list of active users, proxy servers, or local network broadcasting, to determine the initial 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. To reconcile the semantic differences between clusters, inter-ontology mappings (as defined in Section 2.1.2) 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 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

```
/* When a node N receives a neighbor-discovery query Q issued by a
new joining node X, N calls this function to process the query*/

process_neighbor_discovery_query (query Q)
{
1.  if Q has been received before, discard it, return
2.  compute the semantic similarity between X and N, sim(X,N)
3.  if (sim (X,N) =1)
4.      send a reply indicating N is X's zero-distance neighbor
        the reply also contains N's zero-distance neighbours
5.  if (threshold ≤sim(X,N) < 1)
6.      send a reply indicating N is X's short-distance neighbor
7.  if (TTL does not expire)
8.      for each neighbor N_j in N's inter-cluster table
9.          compute the semantic similarity sim(X, N_j)
10.         if (sim(X, N_j) ≥ threshold)
11.             forward Q to N_j
12.     if no N_j found
13.         forward Q to N's long distance neighbors
}
```

Fig. 3. The algorithm of neighbor-discovery query.

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.

2.3.2. *Neighbor discovery query*

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. The detailed query processing algorithm is illustrated in Fig. 3.

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. Queries are translated whenever passing along short-distance links.

3. 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. Since 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. Since 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 among 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. To further improve the performance of intra-cluster searching, we propose an efficient intra-cluster routing algorithm which is presented in the next section.

4. RDV routing

To efficiently forward resource discovery queries inside the cluster, we propose the Resource Distance Vector (RDV) routing algorithm. The main idea of this algorithm is to build and integrate each node's ontological instance summaries. When processing a query, the summaries are used in a pre-processing step to find peers that are likely to provide relevant answers to the query. The RDV algorithm can be used independently as a semantics-based routing algorithm in a network with a fixed ontology schema.

4.1. *Triple filters*

Compared with the whole network, the size of a cluster is relatively small. Therefore, it is possible to index more detailed ontology information into the intra-cluster routing table. Unlike the inter-cluster routing tables which store abstract T-box knowledge, the intra-cluster routing table records detailed A-Box knowledge from neighbors inside the same cluster (i.e., zero-distance neighbor). In the rest of this section, we use the term "neighbor" to represent a zero-distance neighbor. Every peer maintains a resource index table, and peers exchange their indices. Queries can then be distributed by relaying based on these indices. However, the instance-level indexing can be expensive due to the large number of instances. To reduce the overhead of propagating the index information, we propose a lightweight indexing summarization scheme based on a concise data structure — Bloom filter [21].

We extend the classical Bloom filter to a structure called a *triple filter*. As mentioned, the building block of RDF statements is a triple including a *subject*, a *predicate*, and an *object*. Any RDF statement can be represented by a sequence of triples. A triple filter includes three different Bloom filters: the *subject filter*, the *predicate filter*, and the *object filter*. These three filters work together to represent the RDF triples and answer triple membership queries. To store an A-Box RDF statement, the statement is first decomposed to sequence of triples and these triples in turn can be mapped to the corresponding triple filters. The sizes of the subject filter, the predicate filter, and the object filter may be different, so is the number of hash functions used on these filters. Normally, the object filter has a larger size and uses more hash functions, while the predicate filter has a smaller size and uses fewer hashes, because a particular ontology usually has more distinct objects than distinct predicates. To identify the existence of a triple, three parts of the triple are mapped to the corresponding filters. If all of them are found in the triple filter, we conjecture that the queried triple exists. Every node maintains a local triple filter and several aggregated neighbor triple filters. These filters form a routing table that directs query forwarding.

4.2. RDV routing table

We construct the RDV routing table (RDVT) based on the triple filters. Specifically, each node in the network keeps a modified triple filter for every neighbor (adjacent node) in the overlay topology. A neighbor filter is created by merging filters of all nodes *d* hops away from that neighbor; therefore it keeps track of resources reachable via *d* hops through the overlay network starting with that neighbor. We add distance information to the triple filter, so that we can not only know how far away the resource is located, but also control how far a node can "see" its neighborhood; together with the neighbor summaries, we can determine where to forward a particular query. In a routing table, each entry in the triple filter is not a single bit but rather a small counter. Initially, all entries are set to *infinity* (represented by a special number). When a local resource is inserted, the corresponding counters are set to 0, meaning the distance is 0, representing a local resource. When the summary is propagated to another node, the counters corresponding to each resource are incremented. To control the false positives caused by Bloom filter aggregation, we set the maximum value of the counter, which we call the *radius*. The *radius* limits the number of hops the resource information can be propagated. After a series of propagations, if a resource is propagated to a node which is more than *radius* away, then its entries in the RDVT are set to *infinity* (not available). Because of the small-world theory, nodes are connected with a small number of hops. Therefore, a small *radius* works for our system. As revealed by our experiments, 3 bits per counter should suffice.

Fig. 4 shows part of the network with the associated RDVT for each node. For brevity, only one of the three filters is shown here. Each element in the filter is associated with a distance number: the minimum distance to a matching resource. The first row of the RDVT is the local filter containing local resource index. For example, node *A*'s local filter contains a local resource a, which is mapped to two positions (2, 4) in the filter. We set the distance number of a local resource as 0. The rest of the rows represent resources accessible from neighbors. For example, in Fig. 4(a), *A*'s second row contains resources that can be reached through the neighbor *B* (e.g., resource b(4, 0) with 1 hop).

When a node joins the cluster, it should construct its routing table, RDVT. Neighbors of this new node should update their RDVTs to reflect the joining of this new node. Fig. 4 illustrates the RDVT updating process when a new node *C* joins the network. Node *C* joins the network by connecting to an existing node *A* in the network. After the connection is established, node *C* sends its resource indices to *A*. Similarly, *A* should inform *C* of all the resources *A* has knowledge of. Specifically, *A* merges its local and neighbor vectors into one vector and sends it to *C*. The merged vector of *A* represents resources accessible from *A* and their shortest distances to *A*. The merging process guarantees that the nearer resource information always gets higher priority in the filter, because a position occupied by a nearer resource would never be overwritten by a further resource. This property in turn guarantees that increasing *radius* will not bring more false positives for a limited sized triple filter, which will be proved by our experiments as well. *A* does not need to send more information as

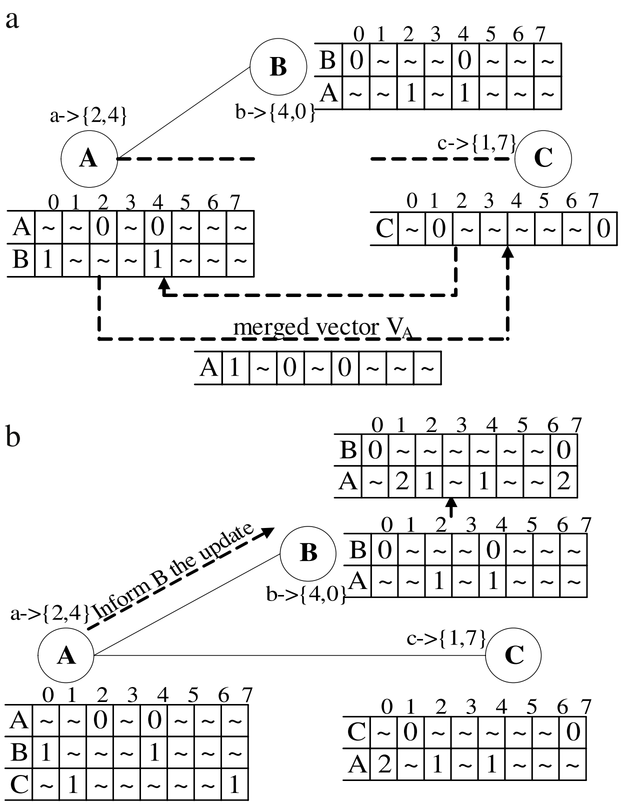


Fig. 4. Construction of the routing table.

C does not need to know the precise location of these resources, but only that they can be accessed through *A*. After *C* receives the merged vector from *A*, it adds 1 hop to each element of the vector, and adds an additional row in its RDVT (as shown in Fig. 4(b)). After *A* receives *C*'s resource information and updates its routing table, it informs its neighbors (in this case, node *B*) of the update. In this way, nodes can construct and update their RDVTs.

A node's routing table should be updated when the resource information changes. When new resources are added to a node, this node calculates the changed positions in its own filter (1st row in its routing table) and the merged filter. It then sends these positions out to each neighbor. The deleting process is more complex. Because of the overlapping of different resources, deleting cannot be performed by simply setting the related hash positions to *infinity*. We can solve the problem by using the counting Bloom filter proposed by Fan et al. [9], or using the timing-based deletion approach [10]. A resource update can be implemented as a deletion followed by an addition. Each node sends updates to and receives updates from its directly connected neighbors. To reduce the overhead of transmitting routing information, a soft-state update mechanism is used, in which routing information is exchanged periodically.

4.3. Query forwarding

Based on the routing table RDVT, we propose a so-called resource-distance-vector (RDV) routing algorithm. It uses a distance vector approach to route the query to the nearest matching nodes. The traditional distance vector approach is not scalable for locating unique nodes in a large network, but this modified version is extremely well suited for our resource discovery problem.

When a node receives a query, it converts the query into a triple sequence and matches the sequence in the RDVT. Besides matching local resources, the query is also forwarded to the "right"

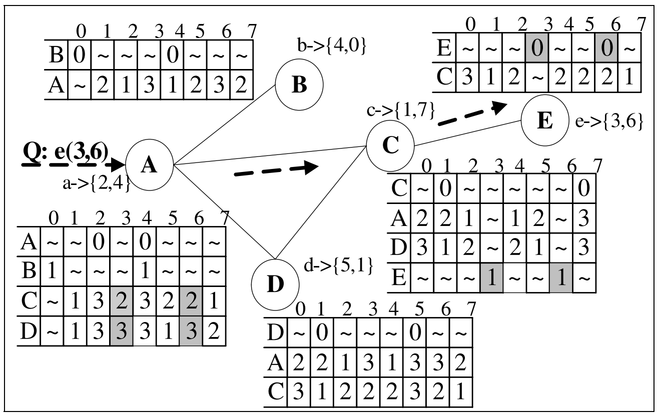


Fig. 5. RDV query routing.

neighbors. A query may be transferred several hops until arriving at the matching node or the query TTL expires. Fig. 5 illustrates a query routing example. We only show one of the three triple vectors. For simplicity, the query has only one constraint. The *radius* is set to 3, so nodes are only aware of resources within 3 hops. In this example, node A receives a query for resource e (which is mapped to two positions: 3 and 6 in the filter). It checks its routing table and finds two matches: through C with 2 hops ($C_3 = 2, C_6 = 2$) and through D with 3 hops ($D_3 = 3, D_6 = 3$). Since the shortest distance to the resource is 2 through neighbor C, the query is forwarded to C. Similarly, C forwards the query to E. E finds a match in its local vector, and then it checks the RDF database against the original query.

Our routing algorithm works fine with networks containing cycles. Because of cycles, a node may receive a query multiple times. To avoid processing queries more than once, every query has a unique query ID and every node keeps a list of recently received query IDs. If a query has been received before, it will be discarded. Another benefit of recording the query is that it ensures the query does not hit the same false positive twice.

4.4. Heuristic jump and caching

By setting a *radius*, we limit the distance a node's resource information can travel. This reduces false positives, but at the same time, this causes a node not to have global knowledge of the network but have only a local view of the neighborhood. Because of this, a node may not find enough matches from its RDVT to forward queries. A naïve solution is to forward the query to some random neighbors even if they have no match hoping that these neighbors can find matches from their neighborhood. This method is inefficient since a node's neighbor has a neighborhood which largely overlaps its own. If the requested resources are scarce in the local area, forwarding the query to another neighbor in this area will not substantially increase the chance of resolving a query. To address this problem, we introduce a forwarding method called the "heuristic jump."

This method allows the system to keep additional long-distance links as an addendum to the RDVT. When the RDVT cannot resolve the query, the query will "jump" to remote nodes the links point to. To discover these long-distance links, the system employs an aggressive caching technique. After finding the result of a query, the result travels along the reverse path to the requester. Whenever it is passed through a node, it is cached in that location. Every internal node caches the query, the destination node, and the distance to that node. We use caching to not only eliminate the need to forward a query which may be resolved locally, but also to use this cached information as links for future long-distance jumps. During the query-forwarding process, when a node cannot find enough matches in its routing table, it chooses appropriate long-distance links from its cache and forwards the query accordingly. This expedites the searching process by jumping over barren areas. Candidate long-distance nodes should be located outside the neighborhood area; i.e., the distance should be greater than *radius*. In our heuristic, we also consider other metrics. For example, the query might "jump" to nodes that answered more previous queries, or to nodes that answered similar queries.

4.5. False positive

RDV routing assumes if all three parts of a triple can be found in the triple filter, then the query can be satisfied by the filter. However, this conjecture may be false, because (1) the Bloom filter structure itself contains false positive; (2) even when all parts of a triple are found in the filter, these parts may belong to different resource instances. We argue that these false positives do not dramatically deteriorate the system's performance or affect the system's fidelity. This is because of the following reasons: (1) The inherent false positive of Bloom filter is $(1 - -e^{-kn/m})^k$ where k is the number of independent hash functions used and m is the size of the filter. As shown in [21], this is trivial and can be ignored when k and m are chosen properly. (2) Users pose queries according to their need and common sense, rather than randomly picking terms from the three domains of the triple. Therefore, aggregation of resource information would not cause too much false in reality. If aggregation does cause false positives, these false positives will reveal themselves after several hops. Moreover, all queries will finally be evaluated at the local resource database, therefore the false positive will not affect the system fidelity. We try to control the false positives caused by the aggregation of resource information with a set of strategies. First we try to reduce the degree of resource aggregation by limiting the resources to be stored in the triple filter to those belonging to 0-neighbors of the node only. Moreover, we also set radius to further limit the aggregation. Our experiments illustrate that false positives do not significantly deteriorate the system's performance.

5. Experiment

In this section, we will explain the experiment setup, and then present the simulation results.

5.1. Setup

As it is difficult to find representative real world ontology data, we have chosen to generate test data artificially. Our data does not claim to model real data, but shall rather provide reasonable approximation to evaluate the performance of the system. We use a dictionary as the vocabulary source of ontological data. The overlapping of ontologies can be controlled by adjusting the dictionary size: When the vocabulary size is smaller, the ontologies created have a higher degree of overlapping, and vice versa. Ontology data can be characterized by many factors such as the number of classes, properties, and individuals; thus we have generated the test data in multiple steps. The algorithm starts with generating the ontology schema. Each schema includes the definition of a number of classes and properties. The classes and properties may form a multilevel hierarchy. Then the classes are instantiated by creating a number of individuals of the classes. To generate an RDF instance triple t, we first randomly choose an instance of a class C among the classes to be the subject: *sub(t)*. A property p of C is chosen as the predicate *pre(t)*, and a value from the range of p to be the object: *obj(t)*. If the range of the selected property p are instances of a class C', then *obj(t)*is a resource; otherwise, it is a literal.

We assume each node uses 1 to 3 ontologies. Each ontology includes at most 10 classes. The number of properties that each class has is at most $k = 3$. The number of instances of each class at each peer is less than 10. Finally, the number of triple patterns in each query we create is either 1 or 3. In our experiment, we do not do knowledge reasoning. In other words, we do not augment the RDF graph by inference (forward chaining). 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.

The queries are generated by randomly replacing parts of the created triples with variables. For our experiments, we use single-triple-queries and conjunctive-triple-queries. To create the conjunctive-queries, we randomly choose a property p_1 of class C_1. Property p_1 leads us to a class C_2 which is the range of p_1. Then we randomly choose a property p_2 of class C_2. This procedure is repeated until the range or the property is a literal value or we have created n ($n \leq 3$) triple patterns.

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 [22] and a random-walk based simple Gnutella scheme [23]. The ShortCut approach is chosen as one comparison reference since it is simple yet effective, and many popular applications (e.g., [22,24–26]) 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 previous research efforts. We tested two versions of OntoSum, OntoSum_0 and OntoSum_1. The former has no intra-cluster RDV routing table and uses random-walk to forward queries inside the cluster; while the latter uses the RDV routing scheme (with *radius 1*) to forward queries inside a cluster. The reason to use 1 as the value of *radius* in this experiment is to save memory storage to support large-scale test. When *radius* is 1, the RDV table does not need to maintain distance information and it is simplified as a Bloom filter bitmap. We use the function defined in Eq. (1) to measure the semantic similarity between OntoSum peers. We set the default similarity threshold t as 0.6.

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 paper, we use the term "query" to refer to resource-discovery query.

The simulation parameters and their default values are listed in Table 1.

Table 1
Parameters used in the simulations.

Parameter	Range and default value
Network size	$2^9 \sim 2^{15}$ default: 10,000
Initial neighbors (node degree)	5
Maximum neighbors	30
Average node degree	14
TTL	$1 \sim 20$ default 9
Resource-discovery query walkers	3 (propagate exponentially)
Neighbor-discovery query walkers	2 (propagate lineally)
Ontology domains	$1 \sim 10$ default: 8
Ontology schemas per domain	$1 \sim 10$ default:8
Distinct resources per domain	100
Resources per node	$1 \sim 10$
RDV table radius	1
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%
RDVT update frequency	every 5 time slices
Sample of nodes to compute diameter	5%

5.2. Results

In this part, we present the experimental results which demonstrate the performance of our searching scheme.

5.2.1. Emergence of the small-world

As discussed, the topology of the peer network is a crucial factor determining the efficiency of the search system. 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. According to one commonly used formula for computing the *clustering coefficient* of a graph Eq. (2), the *clustering coefficient* 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 *cluttering coefficient* of a graph Eq. (3).

$$CCv = \frac{\text{\# of edges between } v\text{'s neighbors}}{\text{maximum \# of possible edges between } v\text{'s neighbors}} \quad (2)$$

$$CC = \frac{1}{|V|} \sum_{v} CCv. \quad (3)$$

The *average path length (APL)* is defined as the average shortest path across all pairs of nodes Eq. (4). 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 [27]. 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 pair of nodes has a directed path.

$$APL = \frac{\sum_{ij} l_{i,j}}{|V| \cdot (|V| - 1)}. \quad (4)$$

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

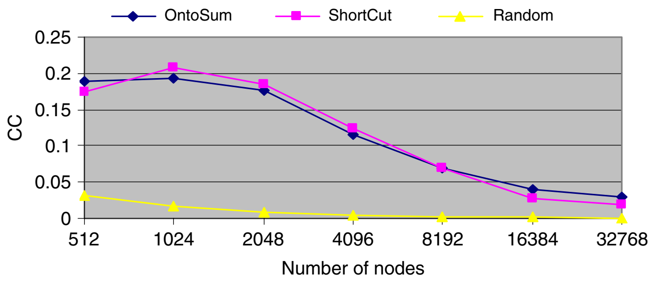


Fig. 6. Comparison of clustering coefficient.

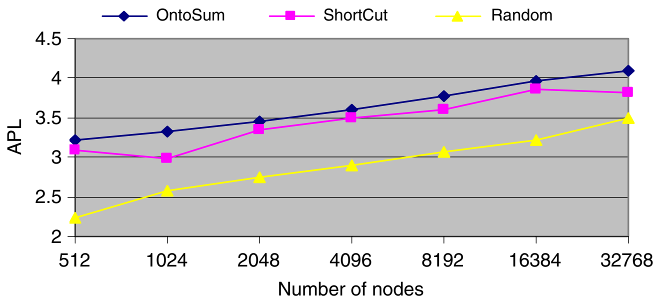


Fig. 7. Comparison of average path length.

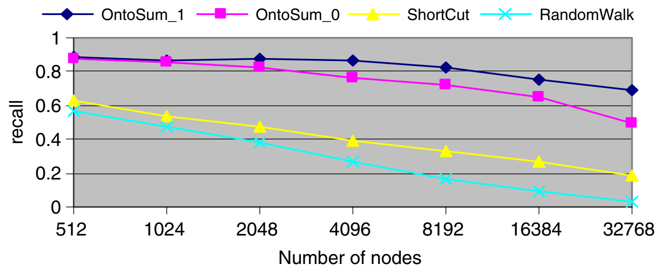


Fig. 8. Recall rate vs. network size.

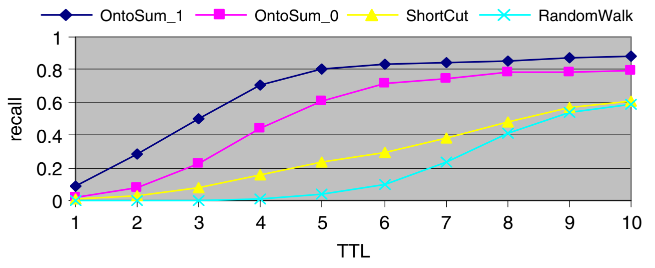


Fig. 9. Recall rate vs. TTL (with # walkers = 3).

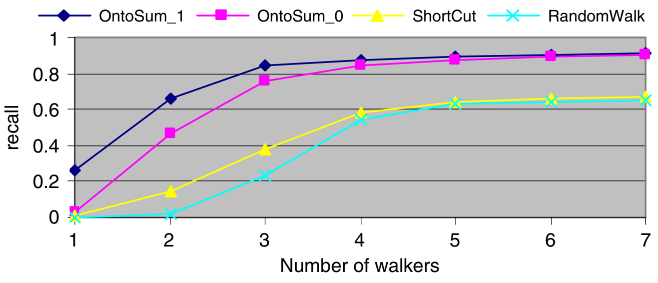


Fig. 10. Recall rate vs. walkers (with TTL = 5).

former has been proved to be a small-world system [28]. 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.

Figs. 6 and 7 show plots of the *clustering coefficient* and the *average path length* as a function of the number of nodes in the network. 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 [27]. Note: Because all of the three topologies 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.

5.2.2. *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. To simulate dynamic factors, in each time slice every node has a 5% probability to issue a query, and a 2% 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 Fig. 8. As we expected, both versions of OntoSum get higher recall in all these different sized networks. In addition, OntoSum's recall decreases less with the increase in network size. Fig. 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 Fig. 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.

As expected, our OntoSum searching scheme performs well as measured by the recall rate. 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.

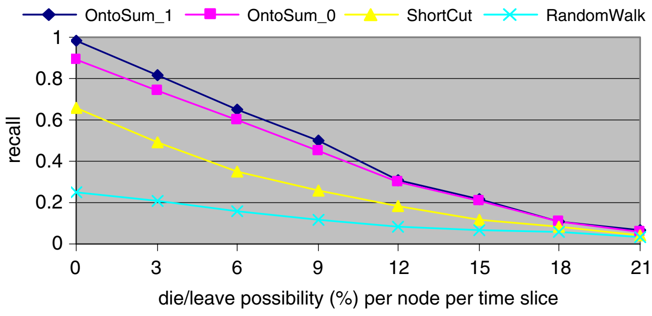


Fig. 11. Recall vs. churn rate.

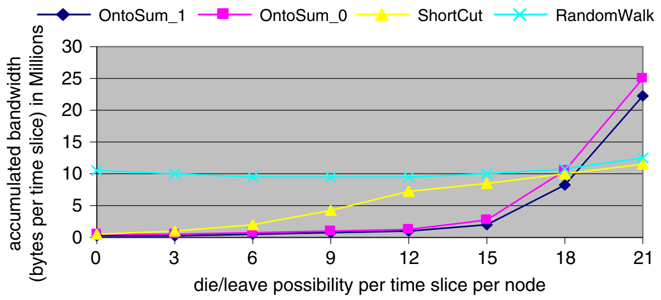


Fig. 12. System overhead vs. churn rate.

5.2.3. *Overhead and adaptability to dynamics*

The good recall performance of OntoSum does not come for free. Generally speaking, the more efficient the query searching is, the more the system has to pay for maintaining the system structure or indexing the resource information, i.e., there is a tradeoff between query efficiency and maintenance overhead. Unlike ShortCut and random-walk approaches, which only create query propagating overhead, OntoSum also creates overhead for maintaining the inter-cluster and intra-cluster routing table. We expect the extra overhead is reasonable and the saving from query cost exceeds the extra maintenance cost. To verify this, we examine the system's overhead in terms of accumulated bandwidth and compare it with that of ShortCut and random-walk. 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.

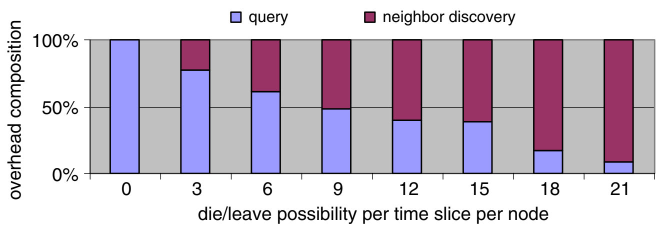
The experiment shown in Fig. 11 gives an overview of how dynamics affect the system performance. Specifically, it shows the query recall rate under different dynamic configurations. In the experiment, we increase the dynamics by increasing the churn rate. From the figure, we find that OntoSum performs similarly to the ShortCut algorithm which is proved to be resilient to churn [22]. When peers join or leave frequently, the performance of ShortCut and OntoSum deteriorate gracefully. Churn does not affect the two schemes dramatically because both algorithms do not depend on a strict structure to perform routing as DHTs do. Their unstructured random topologies provide multiple routes to a destination thus increasing the system resilience. In the worst case, they degrade to random-walk. Another observation is that when the system is more dynamic, OntoSum_1 degrades to OntoSum_0. This is easy to understand because when the system is more dynamic, the resource information in the RDV table is not accurate. In the worst case, using the RDV table to forward the query is like randomly choosing a neighbor to forward to.

Fig. 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, and that OntoSum_1 is even better than OntoSum_0. 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 [29], and we can see from Fig. 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 a solution.

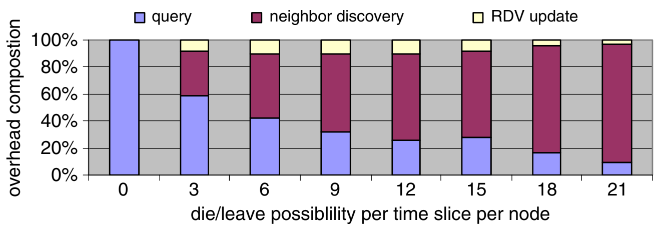
With the same configuration as the experiment in Fig. 12, Fig. 13 illustrates the overhead composition of each routing approach. Most of the overhead of ShortCut and random-walk is caused by query forwarding, and a little overhead is caused by finding neighbors when new nodes are inserted into the system. For OntoSum, the neighbor discovery overhead accounts for a higher proportion of the overhead when the system is more dynamic.

We also performed a set of experiments to evaluate the system overhead under various resource update rates. The resource update rate is represented by the probability of a resource change per node per time slice. There are two types of updates: one is called instance update and the other is called the ontology (schema) update. In an instance update, a node keeps its original ontology schema and only updates instances of that ontology. In this case, each node changes 20% of its resources per update. An ontology update, on the other hand, changes the ontology schema and of course all the related instances. This kind of change is a dramatic change: it means that the node totally changes its interest, and the practical effect is the same as inserting a new node. The rate of these two types of changes is set to 10:1. The RDV routing table is updated periodically every five time slices in the simulation.

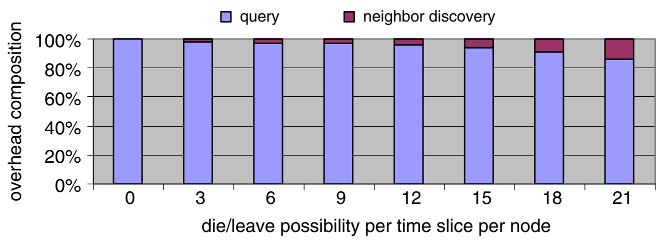
Fig. 14 illustrates the relationship between system overhead and resource update frequency. It is clear that the rate of resource updates has a bigger impact on OntoSum_1 than on the other algorithms. Because OntoSum_1 has to update its RDV routing table to reflect the changing resources, this unavoidably causes more overhead when resource update is frequent. When the resource update is so frequent that the routing table update cannot catch up to the resource update, the information in the routing table cannot represent the real resource distribution, and maintaining the routing table becomes useless. Therefore, in a very dynamic environment, we recommend the system stop intra-cluster routing table updates, and turn to OntoSum_0 as the routing scheme. Because most of the resource updates are instance-level updates, OntoSum_0 and ShortCut do not need to change their neighborhood too much, and consequently they do not see much overhead. If there are more ontology-level updates, it is like inserting more new nodes, and the result would be similar to the result shown in Fig. 12. Fig. 15 illustrates the overhead composition of OntoSum_1



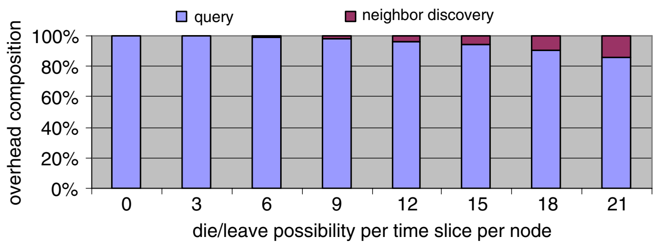
(a) OntoSum_0 overhead composition.



(b) OntoSum_1 overhead composition.



(c) Shortcut overhead composition.



(d) Random-walk overhead composition.

Fig. 13. Overhead composition vs. churn rate.

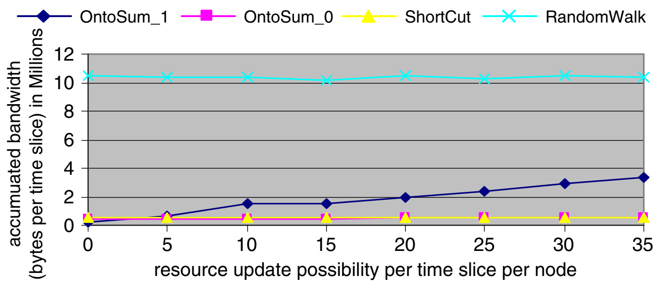


Fig. 14. System overhead vs. resource update rate.

and Onto_Sum_0. Because resource changes do not affect Short-cut and random-walk much, we do not plot their results.

We should understand that the overhead is application dependent. It depends on the quantity of resources, the routing table update rate, as well as factors like the compression rate of RDV routing table. In our experiments our resource size is between 10 to 200 instances per node. We set OntoSum's RDV routing *radius* as 1, which means that inter-cluster routing tables are only exchanged between direct neighbors. We do not compress the RDV table during transferring. If a system has more resources than this configuration, the system will see more overhead and vice versa. We can see there is a tradeoff between query efficiency and indexing overhead. The application should choose a suitable OntoSum version for its particular purpose.

6. Related work

This section gives an overview of technologies and research related to this paper.

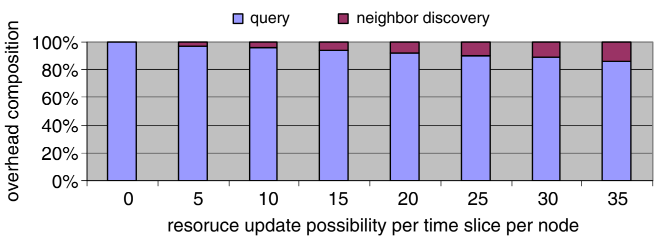
Semantic Web and Semantic Link Network

Proposed by Tim Berners-Lee, inventor of the Web and HTML, the Semantic Web is an evolving extension of the World Wide Web (WWW) in which Web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily. Applications have used the standard RDF language [11] to describe data. Ontology languages, such as DAML+OIL [11] and OWL [12] built on top of RDF, allow describing relations between resources, thus defining a more abstract and expressive resource sharing environment. The Semantic Link Network [6,7,30–32] was proposed by Zhuge et al. as a semantic data model for organizing various web resources. It extends the web's hyperlink to semantic link which links various resources and knowledge and helps humans to understand, learn and discover in the world. SLN has been used to improve the efficiency of query routing in P2P network [13], and it has been adopted as one of the major mechanisms of organizing resources for the knowledge grid [33]. Our proposed system, OntoSum, was inspired by the Semantic Web and the SLN technologies. It harnessed the power of these technologies to aid in resource information representation, retrieval and aggregation over large distributed grids. Therefore, these two technologies are the foundation of OntoSum.

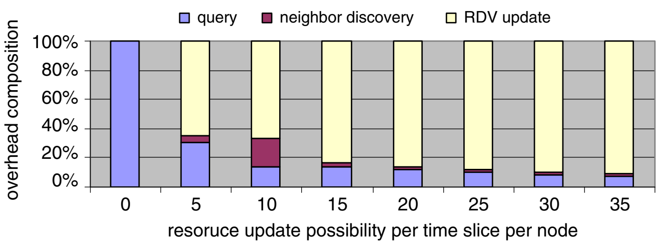
P2P-based semantic search

P2P technology has been used to improve the scalability and efficiency of the semantic searching.

Edutella [34] is a P2P network for searching semantic web metadata. Each Edutella peer can make its metadata information available as a set of RDF statements. The distributed individual RDF peers register the queries they may be asked through the



(a) OntoSum_0 overhead composition.



(b) OntoSum_1 overhead composition.

Fig. 15. Overhead composition vs. resource update rate.

query service, and queries are sent through the Edutella network to the subset of peers who have registered with the service to be interested in this kind of query. To forward queries between nodes, Edutella uses JXTA to broadcast queries to a HyperCup topology. Similarly InforQuilt [35] and Piazza [36] also uses broadcast/flooding to search semantic metadata. The simple P2P broadcast structure used by these systems makes them very difficult to scale to large-scale networks. Our system solves this problem by topology adaptation and semantics-based routing.

Projects, such as RDFPeer [37] and OntoGrid [38], attempt applying DHT techniques to the retrieval of the ontology encoded knowledge. The basic idea is to map each keyword of a semantic entity to a key. For example, RDFPeer [37] indexes each RDF triple to support semantic RDF query. A query with multiple keywords then uses the DHT to lookup each keyword and returns the intersection. However, DHTs have difficulty in supporting other semantically richer queries, such as wildcard queries, fuzzy queries, and proximity queries. In addition, most DHT-based applications require all peers in the system sharing a uniform ontology schema, which is impractical in reality. These limitations restrict the deployment of DHTs to semantic data discovery. Like DHT-based approaches, our OntoSum system resolves the scalability issue; but unlike DHTs that cannot support complex queries, our system has no limitation for query format and can support arbitrary types of queries.

Semantic clustering

Recently, there has appeared the idea of grouping nodes with similar contents together to facilitate searching [39–42]. The latest super-peer-based Edutella [41] and Semantic Overlay Network (SON) [43] rely on centralized server or super-peers to cluster contents and nodes. However, super-peers may be potential bottlenecks of the system, and efficient communication mechanism between super-peers is absent in these systems. Preliminary work in [40] proposes to cluster nodes with similar interest together, without discussing how to define the interest similarity amongst peers and how to form clusters. [42] relies on periodic message exchanges amongst peers to keep track of other peers with similar documents, which incurs very high message overhead. Semantic Small Word (SSW) positions peers and data objects in the semantic space, so that peers with similar data objects form into clusters. It then applies a dimension reduction technique on top of the DHT to realize a semantics-based search. In SSW, semantics of data objects is represented by a multi-attribute vector, but not Semantic Web-based data. Applications such as REMINDIN [22], Helios [25], and Bibster [26] add semantic short-cuts to group nodes. The short-cut approach relies on the presence of interest-based locality. 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. Similar to these systems, our OntoSum system uses semantic clustering to organize the network topology and reduce search space to semantically related clusters. Unlike the super-peer approaches, OntoSum uses a fully-decentralized approach which automatically forms the semantic group, thus avoiding the bottleneck problem. Compared with SSW, OntoSum supports complex semantic web data, not just multi-attribute data. Compared with short-cut approaches, OntoSum dramatically outperforms them in accuracy, scalability and efficiency, as illustrated by our simulation experiments.

7. Summary

In this paper, we presented an effective framework, OntoSum, for resource discovery in a large-scale heterogeneous grid. In OntoSum, grid nodes automatically organize themselves based on their semantic properties to form a semantically-linked overlay network. A semantics-based routing algorithm, RDV, is proposed to enable efficient information retrieval within semantically related small-worlds. We evaluated the performance of our system with extensive simulation experiments, the results of which confirmed the effectiveness of the design in scalability, efficiency, robustness, and overhead.

References

[1] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proceedings 10th IEEE International Symposium on High Performance Distributed Computing, HPDC-10'01, 2001, pp. 181–194.
[2] H. Casanova, Distributed computing research issues in grid, computing, typescript, Univ. of California, San Diego, 33 (3) (2002) 50–70.
[3] Globus Toolkit: http://www.globus.org/toolkit/.
[4] M. Cai, M. Frank, J. Chen, P. Szekely, MAAN: A multi-attribute addressable network for grid information services, in: The 4th International Workshop on Grid Computing, 2003, pp. 184–191.
[5] 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, pp. 51–62.
[6] H. Zhuge, Ruixiang Jia, Jie Liu, Semantic link network builder and intelligent semantic browser, Concurrency - Practice and Experience 16 (14) (2004) 1453–1476.
[7] H. Zhuge, Yunchuan Sun, Ruixiang Jia, Jie Liu, Algebra model and experiment for semantic link network, IJHPCN 3 (4) (2005) 227–238.
[8] 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.
[9] J. Kleinberg, Navigation in a small world, Nature 406 (2000) 845.
[10] W.A. Gruver, J.C. Boudreaux, Intelligent Manufacturing: Programming Environments for CIM, Springer-Verlag, London, 1993.
[11] O. Lassila, Ralph R. Swick, W3C Resource Description framework (RDF) Model and Syntax Specification, World Wide Web Consortium, 1999.
[12] OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004 http://www.w3.org/TR/owl-features/.
[13] H. Zhuge, Jie Liu, Liang Feng, Chao He, Semantic-based query routing and heterogeneous data integration in peer-to-peer semantic link networks, in: ICSNW, 2004, pp. 91–107.
[14] J. Jiang, D. Conrath, Semantic similarity based on corpus statistics and lexical taxonomy, in: Proceeding of the Int'l Conf. Computational Linguistics (ROCLING X), Vol. cmp-lg/9709008, 1997.
[15] J. Lee, M. Kim, Y. Lee, Information retrieval based on conceptual distance in IS-A hierarchies, Journal of Documentation 49 (1993) 188–207.
[16] M.A. Rodriguez, M.J. Egenhofer, Determining semantic similarity among entity classes from different ontologies, IEEE Transactions on Knowledge and Data Engineering 15 (2) (2003) 442–456.
[17] G.A. Miller, R. Beckwith, C. Fellbaum, D. Gross, K.J. Miller, Introduction to WordNet: An on-line lexical database, International Journal of Lexicography 3 (4) (1990) 235–244.
[18] A. Tversky, Features of similarity, Psychological Review 84 (4) (1977) 327–352.
[19] D. Lin, An information-theoretic definition of similarity, in: Proceeding of the 15th International Conf. on Machine Learning, San Francisco, CA, 1998, pp. 296–304.
[20] College of American Pathologists. "SNOMED RT - Systematized Nomenclature of Medicine Reference Terminology," VERSION 1.1, USER GUIDE, 2001.
[21] B. Bloom, Space/time tradeoffs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426.
[22] X. Tempich, S. Staab, A. Wranik, REMINDIN': semantic query routing in peer-to-peer networks based on social metaphors International World Wide Web Conference (WWW), New York, USA, 2004, pp. 640–649.
[23] Gnutella website. http://gnutella.wego.com/.
[24] K. Sripanidkulchai, B. Maggs, H. Zhang, Efficient content location using interest-based locality in peer-to-peer systems, in: Proceedings of the INFOCOM'03, 2003.
[25] S. Castano, A. Ferrara, Montanelli, D. Zucchelli, Helios: A general framework for ontology-based knowledge sharing and evolution in P2P systems, in: IEEE Proc. of DEXA WEBS 2003 Workshop, Prague, Czech Republic, 1(5), 2003, pp. 597–603.
[26] S. Castano, A. 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, pp. 55–68.
[27] D. Watts, S. Strogatz, Collective dynamics of small-world networks, Nature 393 (1998) 440.
[28] C. Tang, Z. Xu, S. Dwarkadas, Peer-to-peer information retrieval using self-organizing semantic overlay networks, in: Proceedings of 2003 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications, 2003, pp. 175–186.

[29] D. Stutzbach, R. Rejaie, Understanding churn in peer-to-peer networks, in: Proceeding of the Internet Measurement Conference, IMC, 2006, pp. 189–202.
[30] H. Zhuge, Communities and emerging semantics in semantic link network: Discovery and learning, IEEE Transactions on Knowledge and Data Engineering 21 (6) (2009) 785–799.
[31] H. Zhuge, X. Li, Peer-to-peer in metric space and semantic space, IEEE Transactions on Knowledge and Data Engineering 6 (19) (2007) 759–771.
[32] H. Zhuge, Semantics, resource and grid, Future Generation Computer Systems 20 (1) (2004) 1–5.
[33] H. Zhuge, The Knowledge Grid, World Scientific Publishing Co., Singapore, 2004, 280 p.
[34] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. SIntek, A. Naeve, M. Nilsson, M. Palmer, T. Risch, Edutella: A P2P networking infrastructure based on RDF, in: Proceedings of the WWW2002, May 7–11, Honolulu, Hawaii, USA, 2002, pp. 604–15.
[35] M. Arumugam, A. Sheth, I.B. Arpinar, Towards peer-to-peer semantic web: A distributed environment for sharing semantic knowledge on the web, in: International World Wide Web Conference 2002, WWW2002, Honolulu, Hawaii, USA, 2002.
[36] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, The Piazza Peer Data Management System, 16(7) (2004) 787–798.
[37] M. Cai, M. Frank, RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network, in: Proc. of WWW Conference, 2004, pp. 650–657.
[38] OntoGrid project: http://www.ontogrid.net/.
[39] M. Bawa, G.S. Manku, P. Raghavan, SETS: Search enhanced by topic segmentation, in: Proceedings of ACM SIGIR, 2003, pp. 306–313.
[40] A. Iamnitchi, M. Ripeanu, I.T. Foster, Locating data in (small-world?) peer-to-peer scientific collaborations, in: Proceedings of International Workshop on Peer-to-Peer Systems, IPTPS, 2002, pp. 232–241.
[41] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M.T. Schlosser, I. Brunkhorst, A. Lser, Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, in: Proceedings of International World Wide Web Conference, WWW, 2003, pp. 536–543.
[42] C.H. Ng, K.C. Sia, C.H. Chang, Advanced peer clustering and firework query model in the peer-to-peer network, in: Proceedings of International World Wide Web Conference, WWW, Poster ID S130, 2003.
[43] A. Crespo, H. Garcia-Molina, Semantic overlay networks for p2p systems. Technical report, Stanford University, 2002.

Juan Li is an assistant professor in computer science at North Dakota State University. She received her Ph.D. degree in computer science from the University of British. She also holds an M.S. degree in computer science from the Chinese Academy of Sciences. Her main research interests include P2P overlay network, grid computing, and semantic web technologies.