Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 4543–4560
Computer Networks
www.elsevier.com/locate/comnet

Information retrieval in schema-based P2P systems using one-dimensional semantic space

Tao Gu [a,b,*], Hung Keng Pung [b], Daqing Zhang [a]

[a] *Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore*
[b] *National University of Singapore, 3 Science Drive 2, Singapore*

Available online 30 June 2007

Abstract

The widespread use of RDF-based information necessitates efficient information retrieval techniques in wide-area networks. In this paper, we present *Dynamic Semantic Space*, a schema-based peer-to-peer overlay network that facilitates efficient lookup for RDF-based information in dynamic environments. Peers in this overlay are grouped based on the semantics of their data which are extracted according to a set of schemas, and self-organized as a semantic overlay network. To reduce overheads incurred by peer joining, leaving and content changes in a high-dimensional overlay network, peers are constructed as a one-dimensional semantic space that facilitates efficient routing for both pull and push requests. A search or a subscription request is only routed to the appropriate cluster that holds related data, thus reducing unnecessary search cost and increasing the efficiency of locating information. Through a comprehensive simulation study, we demonstrate the effectiveness of our proposed techniques.

Keywords: RDF; Ontology; Schema-based peer-to-peer overlay network; Semantic peer-to-peer network

1. Introduction

Resource Description Framework [1] has been widely recognized as the standard for storing and exchanging information on the World Wide Web. RDF statements which describe resources and their semantics are machine-understandable and machine-processable, and can be created by different users and widely distributed on the Web. The distribution of RDF statements provides great flexibility for describing resources and building applications. With the increasing use of RDF statements, there is a need to support efficient retrieval of RDF data in wide-area networks. In recent years, dynamic applications such as e-business and context-aware pervasive systems are becoming more and more popular. Information to be shared in these applications typically exhibits a variety of dynamic characteristics, e.g., the location of a user may be changed frequently. It is more challenging to provide efficient information retrieval for such applications due to the dynamic changes of their data.

One approach is to use centralized search engines to index RDF data. These indices can be obtained by

* Corresponding author. Address: Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore.
E-mail addresses: tgu@i2r.a-star.edu.sg (T. Gu), punghk@comp.nus.edu.sg (H.K. Pung), daqing@i2r.a-star.edu.sg (D. Zhang).

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2007.06.019

crawling Web pages such as in RDF Google. Although this approach can provide fast response to a query, it is difficult to keep these indices up to date due to dynamism of data sources. In addition, this approach has limitations such as scalability, processing bottleneck and single point of failure. Peer-to-Peer (P2P) approaches have been proposed to overcome some of these obstacles, and provide potential solutions for building non-centralized information lookup systems. P2P systems such as Gnutella [2] allow nodes to interconnect freely and have low overlay maintenance overhead, making it easy to handle the dynamic changes of peers and their data. However, a query has to be flooded to all nodes in the network including those nodes that do not have relevant data. The blind flooding mechanism used without any restriction on the scope of flooding can become very inefficient because of excessive redundant messages. Other P2P systems such as Chord [3], CAN [4], Pastry [5] and Tapestry [6] typically implement distributed hash tables (DHTs) and use hashed keys to direct a lookup request to the specific nodes by leveraging on a structured overlay network among peers. However, data placement in these systems is tightly controlled based on distributed hash functions. In dynamic environments peers may join or leave the system frequently and data may be changed rapidly; thus higher overlay maintenance overhead for updating the relevant information in the DHT-based overlay networks is inevitable. Moreover, for certain applications such as those in context-aware systems [7], it is desirable but may not be possible to place data in a particular node (i.e., near the data source) using a hash value.

To facilitate the efficient retrieval of RDF data in dynamic environments, we present *Dynamic Semantic Space* (DSS), a schema-based P2P overlay network in which RDF data are organized and retrieved based on their semantics to support both pull and push services. In this overlay, data can be represented by a collection of RDF statements based on a set of schemas (i.e., ontologies). RDF statements which are semantically similar are ''tied'' together so that they can be retrieved by a query which has the same semantics. As a result, the system is able to forward a query to nodes which are likely to contain the relevant data.

While the basic idea appears simple, there are several issues that have to be considered in order to make the system work efficiently. Firstly, overlay maintenance cost can be high due to the frequent changes of peers and their data. Hence, minimizing overlay maintenance cost is important in designing the search mechanism in DSS. Secondly, the mapping from data and queries to semantic clusters should not incur much overhead. Thirdly, the number of semantic clusters used in real-life applications can be potentially large. To accommodate the heterogeneity of data sources, the search mechanism in DSS should operate efficiently in a high-dimensional space without incurring high overhead. Finally, as data may change rapidly in dynamic environments, it is important to automatically notify consumers when changes occur. Hence, DSS should be able to adapt and scale to data change and growth.

To address these issues, we propose the following techniques:

- Use ontology-based metadata to extract the semantics of data and queries. This technique can map data and queries to the appropriate semantic cluster(s) with minimum computational overhead in the presence of frequent peer joining/leaving or content changes.
- Upon joining the system, peers are grouped and arranged into a one-dimensional semantic space where various semantic clusters are organized and interconnected in a ring space. This structure enables the mapping of the clusters in a k-dimensional semantic space to a one-dimensional semantic space, and hence reduces overlay maintenance overhead.
- A cluster encoding scheme enables the system adapt to the number of peers by splitting or merging clusters. This can result in a system which has good scalability and load balancing characteristics. This scheme also enables the use of parallelism in our system when searching for data within a semantic cluster.
- Deploy both pull and push services in DSS. Consumers can either submit search requests or subscription requests. The latter allows consumers to be notified whenever data changes occur.

The rest of the paper is organized as follows. We discuss related work in Section 2. We present the details of DSS in Section 3, and the evaluation results in Section 4. Finally, we conclude the paper in Section 5.

2. Related work

Centralized RDF repositories and lookup systems such as RDFStore [8] and Jena [9] have been

implemented to support the storing and querying of RDF documents. These systems are simpler to design and reasonably fast for low to moderate number of triples. However, they have the traditional limitations of centralized approaches, such as single processing bottleneck and single point of failure.

Schema-based P2P networks such as Edutella [10] are proposed to combine P2P computing and the Semantic Web. These systems build upon peers that use explicit schemas to describe their contents. They use super-peer based topologies, in which peers are organized in hypercubes for routing queries. However, current schema-based P2P networks still have some shortcomings; queries have to be flooded to every node in the network, making the system difficult to scale. Chirita et al. [11] built a publish/subscribe system on the Edutella P2P infrastructure. This system uses content advertising, subscribing and notifying. However, content advertising may create additional overhead. In our system, a subscription request is first directed to a set of potential producer peers in a semantic cluster. Following that, each producer peer will map the request against its local RDF data. Crespo et al. [12] proposed the concept of Semantic Overlay Networks (SONs) in which peers are grouped by semantic relationships of documents they store. Each peer stores additional information about content classification and route queries to the appropriate SONs, increasing the chances that matching objects will be found quickly and reducing the search load. However, the maintenance cost in SONs becomes more expensive when the number of SONs increases. We adopt the basic idea of semantic clustering, and we impose certain link structures on these semantic clusters to facilitate both intra-cluster and inter-cluster routing and to reduce the overlay maintenance cost. Cai et al. [13] proposed a distributed RDF repository that duplicates and stores each triple at three places in a multi-attribute addressable network which extends Chord by using a globally known hash function. Queries can then be efficiently routed to those nodes in the network where the triples in question are known to be stored if they exist. However, the overlay maintenance cost is high in this system. In addition, storing each RDF triple multiple times in the network increases the data storage and maintenance costs.

Tang et al. [14] applied classical Information Retrieval techniques to P2P systems and built a decentralized P2P information retrieval system called pSearch. The system makes use of a variant of Content-Addressable Networks (CAN) to build the semantic overlay and uses Latent Semantic Indexing (LSI) [15] to map documents into term vectors in the space. Li et al. [16] built a semantic small world network in which peers are clustered based on term vectors computed using LSI. They proposed an adaptive space linearization technique, and constructed the link structures based on small world network theory. The small world network model was originally introduced by Kleinberg [17]. He proposed a two-dimensional grid where every node maintains four links to each of its closest neighbors and one long distance link to a node chosen from a probability function. He has shown that a query can be routed to any node in $O(\log^2 n)$ hops where n is the total number of nodes in the network. Our work is inspired by the small world model. DSS not only maps a k-dimensional semantic space to a one-dimensional semantic space, but also allow peers to be grouped into sub-clusters in a semantic cluster (through the cluster encoding scheme) to better accomplish scalability as well as facilitate parallel search. To route queries across clusters in DSS, we select two long distance links which are located at certain positions of the ring space instead of choosing one randomly as in the small world network model. Through our simulation, we show how these two *shortcuts* improve the search efficiency with a variant setting of number of semantic clusters. Furthermore, we propose the use of schema-based metadata to extract data semantics which has low overhead as compared to LSI. We show how these ideas can be applied into a semantic-based P2P lookup system.

3. Dynamic semantic space

In this section, we first present an overview of DSS, followed by a description of technical details. For ease of discussion, we use the terms node and peer interchangeably for the rest of the paper.

3.1. Overview

In DSS, a large number of nodes are self-organized into a semantic overlay network, in accordance with their semantics. A user or an application can act as a producer, a consumer or both. Producers provide various RDF data for sharing; whereas consumers obtain data by submitting their queries and receiving query results. Each peer maintains a local data repository which supports

RDF-based semantic query using RDQL [18]. Upon its creation, each producer peer will join a semantic cluster based on the semantics of its major data and publish its data indices to peers in other semantic clusters. Peers within a cluster are interconnected using an overlay structure. There is no restriction on the type of overlay used within a cluster. Upon receiving a query, a peer first pre-processes the query to obtain the information about the semantic cluster associated with the query, and then routes it to an appropriate semantic cluster. When the query reaches the designated semantic cluster, it is forwarded in parallel. Peers that receive the query will do a local search, and return results if available. There are two types of queries in DSS: search request and subscription request. Search requests enable consumers to pull data from the network at a one-time basis, while subscription requests enable consumers to subscribe RDF data and be notified when data changes occur over a period of time.

3.2. Ontology-based semantic clustering

In this section, we describe how to use ontology-based metadata to extract the semantics of both RDF data and queries. There are several advantages as compared to other semantic extraction techniques such as Vector Space Model (VSM) [19] and LSI. The formal design of ontologies minimizes the problems of synonyms and polysemy incurred by VSM. Based on ontologies, data and queries can be mapped to appropriate semantic clusters directly without costly computation as in LSI, yet the same precision is retained.

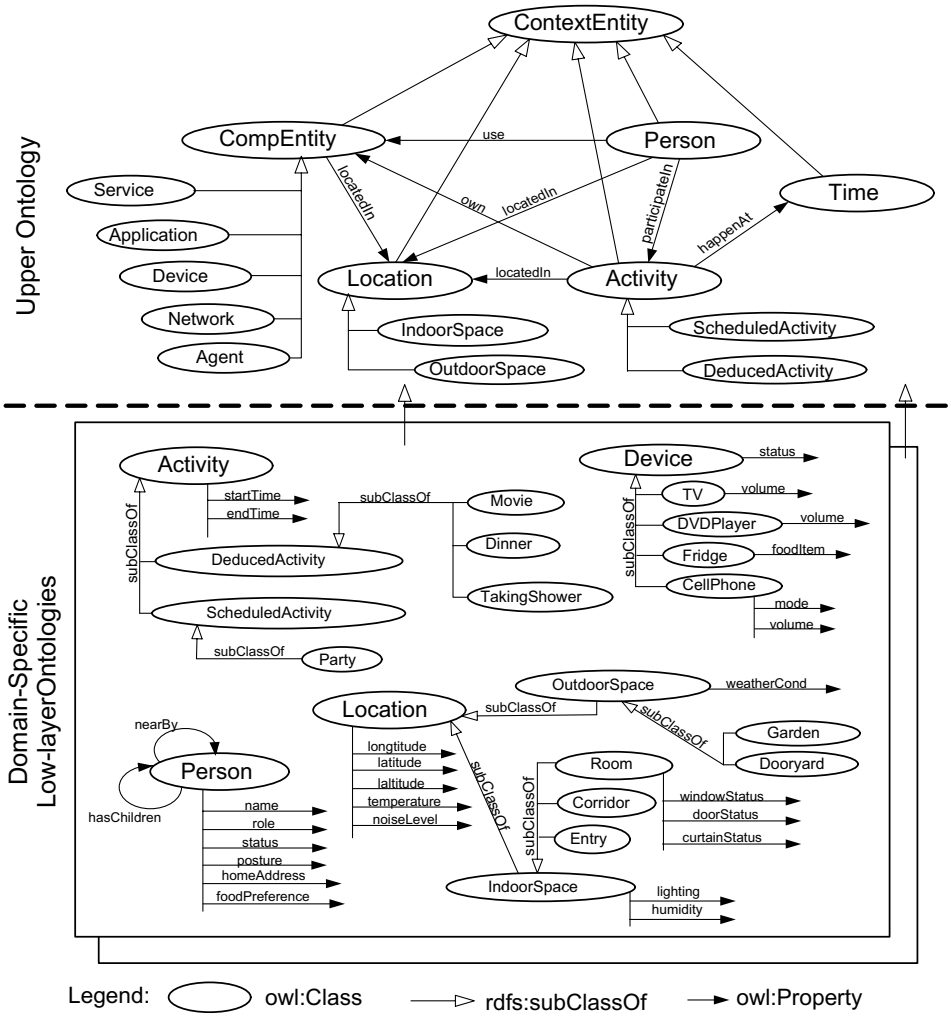


Fig. 1. An example of ontological structure in the context-aware computing domain.

We adopt a two-tier hierarchy in ontology design. The upper ontology defines common concepts and is shared by all peers. Each peer can define its own concepts in its domain-specific low-layer ontology. Different peers may store different sets of low-layer ontologies based on their application needs. We illustrate the mapping process using an example of ontology in the context-aware computing domain as shown in Fig. 1. The leaf concepts in the upper ontology are used as semantic clusters, and are denoted as a set $E = \{Service, Application, Device, \ldots\}$. Each of these pre-defined semantic clusters will be assigned with a unique *Semantic ID* (described in Section 3.3) upon their presence in DSS.

The mapping computation is done locally at each peer. For the mapping of RDF data, a peer needs to define a set of low-layer ontologies and store them locally. Upon joining DSS, a peer first obtains the upper ontology and merges it with its local low-layer ontologies. Then it creates instances (i.e., RDF data) and adds them into the merged ontology to form its local knowledge base. A peer's local data may be mapped into one or more semantic clusters by extracting the subject, predicate and object of an RDF data triple. Let SCn_{sub}, SCn_{pred}, SCn_{obj} where $n = 1, 2, \ldots$ denote the semantic clusters extracted from the subject, predicate and object of a data triple respectively. Unknown subjects/objects (which are not defined in the merged ontology) or variables are mapped to E. If the predicate of a data triple is of type *ObjectProperty*, we obtain the semantic clusters using $(SC1_{pred} \cup SC2_{pred} \cup \cdots SCn_{pred}) \cap (SC1_{obj} \cup SC2_{obj} \cup \ldots SCn_{obj})$. If the predicate of a data triple is of type *DatatypeProperty*, we obtain the semantic clusters using $(SC1_{sub} \cup SC2_{sub} \cup \ldots SCn_{sub}) \cap (SC1_{pred} \cup SC2_{pred} \cup \ldots SCn_{pred})$. Examples 1 and 2 in Fig. 2a show RDF data triples about location and light level in a bedroom provided by a producer peer. In Example 2, we first obtain the semantic clusters from both subject and predicate, and then intersect their results to get the final semantic cluster – *IndoorSpace*.

A query follows the same procedure to obtain its semantic cluster(s), but it needs all the sets of low-layer ontologies. In real applications, users may create duplicate properties in their low-layer ontologies which conflict with the ones in the upper ontology.

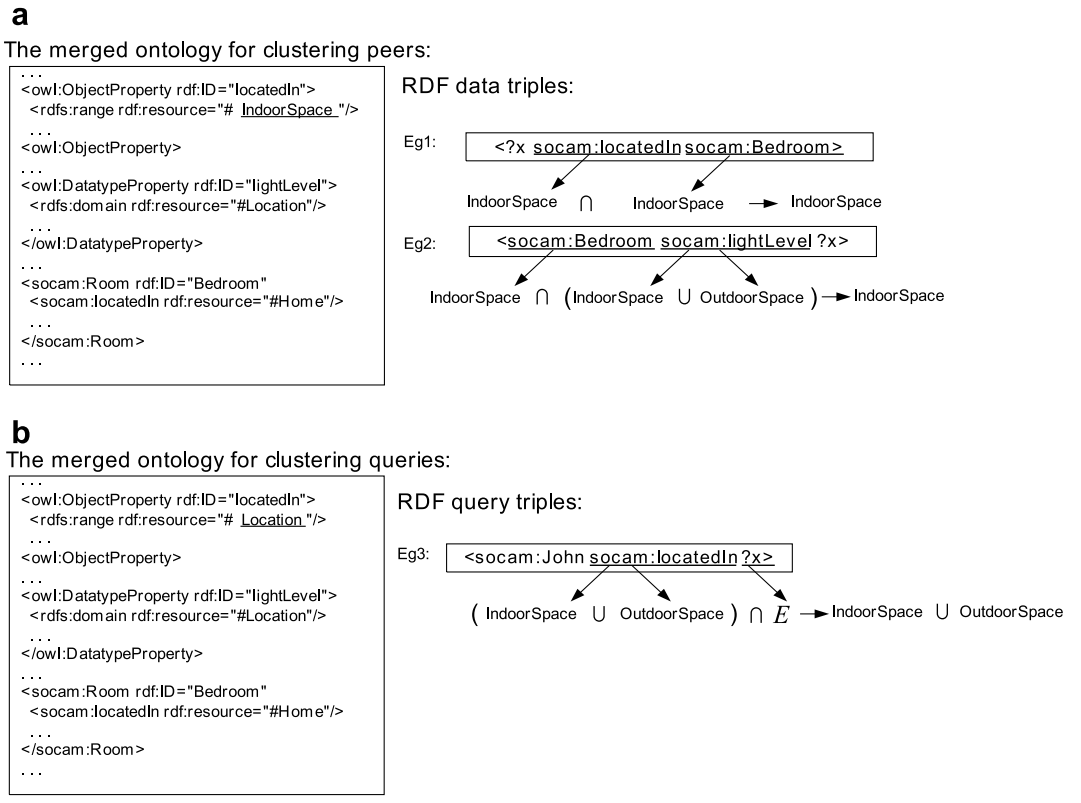


Fig. 2. An example of semantic cluster mapping.

For example, the upper ontology defines the *rdfs:range* of predicate *locatedIn* as *Location* whereas the low-layer ontology defines its *rdfs:range* as *IndoorSpace*. To resolve this issue, we create two merged ontologies, one for clustering peers and the other for clustering queries. If such a conflict occurs, we select the affected properties defined in the low-layer ontology to generate the merged ontology for clustering peers' data and select the affected properties defined in the upper ontology to generate the merged ontology for clustering queries. With this scheme, a peer can extract the semantics of its data triples more precisely based on its low-layer ontology without losing generality for queries. For example, predicate *locatedIn* may have the *rdfs:range* of *IndoorSpace* (underlined in Fig. 2a) in the merged ontology for clustering peers' data and have the *rdfs:range* of *Location* (underlined in Fig. 2b) in the merged ontology for clustering queries. Data triple '*Someone is located in the bedroom*' will be mapped to *IndoorSpace*; and query '*where is John*' will be mapped to *IndoorSpace* and *OutdoorSpace* rather than only *IndoorSpace*. This is most likely the case in real-life applications.

3.3. One-dimensional semantic space

In DSS, peers are organized in such a way that those with semantically similar data are grouped together. To enable search across semantic clusters, an intuitive solution is to construct k-dimensional semantic clusters by connecting each peer to all dimensions of the corresponding clusters such as in [12,7]. However, overlay maintenance cost becomes expensive when the number of semantic clusters increases. To reduce overlay maintenance cost, we present a new approach to facilitate efficient search in a high-dimensional semantic space. We build an overlay network using the one-dimensional ring structure which enables the mapping from a k-dimensional semantic space into a one-dimensional semantic space.

3.3.1. Peer placement

To join a semantic cluster in the network, a peer first obtains the semantics of its local data. This is done by mapping each RDF triple in its repository to one or more semantic clusters using the technique described in Section 3.2. We then count the RDF triples for each semantic cluster obtained. The semantic cluster corresponding to the most triple count (i.e., its major data) is called its major semantic cluster; and the remaining semantic clusters are called its minor semantic clusters.

A peer will then join its major semantic cluster. In order for a query to reach all nodes that provide the same semantics, we adopt index publishing. A peer publishes the indices of its data to its minor semantic clusters as follows: it selects a node in each of its minor semantic clusters and publishes the index (i.e., reference pointer) to these nodes. Each index points to a node where the data is physically stored. For example, as shown in Fig. 3, *Peer 1* publishes its index to semantic cluster *SC1* by putting its index to *Peer 3* which is selected in random within *SC1* (for balancing the load of indices). As a result, a semantic cluster can be viewed as a set of interconnected nodes separated by clusters and a collection of indices stored in these nodes.

The above scheme has several positive effects. For example, if a peer has homogeneous data in its local repository, most of its data will be categorized into one corresponding semantic cluster, therefore reducing the cost to publish data indices. This is likely to be the case in real applications. Furthermore, many applications are designed in such a way that a peer is likely to query for data available in its nearby peers. By placing a peer into one particular semantic cluster based on the majority of its local data, a query can be resolved very efficiently. It should be noted that while we have only elaborated the joining of a single semantic cluster, the same principle can be applied by any peer for joining multiple semantic clusters.

A peer may periodically calculate the triple count for each semantic cluster. The time interval to per-

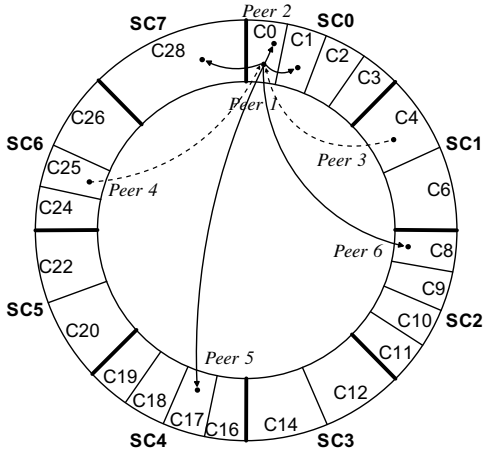


Fig. 3. One-dimensional semantic space.

form such operation is application-specific and depends on the dynamism of its data (i.e., how often the data is changed or updated); hence it will not be further studied in this paper. If its major semantic cluster is changed, a peer will need to initiate a new joining process. If its minor semantic clusters are changed, a peer will need to remove the outdated indices or publish new indices.

3.3.2. Cluster naming scheme

In the design of DSS, one crucial issue is how to design a naming space. We propose a cluster naming scheme which allows sub-clustering within a semantic cluster. We distinguish the concepts of *cluster* and *semantic cluster*. A cluster refers to a partition which consists of a set of nodes bundled together such as $C0$ in Fig. 3. A semantic cluster refers to a set of clusters corresponding to the same semantics. For example, cluster $C0$, $C1$, $C2$, and $C3$ belongs to semantic cluster *SC0*. We propose our cluster encoding scheme as follows. A *Cluster ID* which is represented by a k-bit binary string (where $k = m + n$) is a unique ID that identifies a cluster in DSS. The first m-bit binary string (called *Semantic Cluster ID*) is used to identify a semantic cluster. Hence, a DSS can have a maximum of 2^k clusters and 2^m semantic clusters. An example of a DSS which assumes $k = 5$ and $m = 3$ is illustrated in Fig. 3. The rationale behind this encoding scheme is that, for a given query, we need to obtain the appropriate *Semantic Cluster ID* to match the same semantics of the query. Semantic clusters can be viewed as an additional semantic layer on top of actual clusters. Partitioning peers into a set of clusters in a same semantic cluster also provides better load balancing and enables parallel search within the same semantic cluster.

3.3.3. Ring construction

In DSS, clusters are placed in the ring space based on their cluster IDs. Each node maintains a set of node entries in its routing table for the purpose of both intra-cluster routing and inter-cluster routing. A node, say x, first decides in which semantic cluster to participate. It then picks a cluster randomly within this semantic cluster to join by connecting to a number of nodes in this cluster. These node entries (called x's neighbors in its own cluster) will be maintained in x's routing table as intra-cluster routing information. Node x also creates and maintains two node entries in each of its adjacent clusters. We call these two nodes x's neighbors in its adjacent clusters. Each node joins the network by performing this operation, resulting in all the clusters being linked linearly in a ring fashion. With this ring structure, a k-dimensional semantic space can be linearized.

Maintaining two neighbors in the adjacent clusters for every node in DSS also ensures that a query generated at any node will be able to reach any other cluster by navigating the ring space. However, queries have to be passed around the ring space linearly either clockwise or anticlockwise until the destination cluster is reached. To accelerate search across clusters in DSS, node x maintains a set of links to nodes in other semantic clusters except the two adjacent clusters. These nodes provide *shortcuts* (similar to long contacts in Kleinberg's small world model) for x to route a query to other semantic clusters quickly. For example, in Fig. 3, x creates and keeps track of two *shortcuts*: one points to the opposite semantic cluster (i.e., *shortcut* to *Peer 5*) and the other points to the semantic cluster located in a quarter of the ring space (i.e., *shortcut* to *Peer 6*). These *shortcuts* and neighboring nodes in adjacent clusters are used by node x to perform inter-cluster routing. In the process of cluster splitting and merging or when a new semantic cluster is inserted into the ring space, a node needs to update its neighboring nodes in both its own cluster and its adjacent clusters. However, a node only needs to update its *shortcuts* upon the insertion or deletion of a semantic cluster as a *shortcut* points to an appropriate semantic cluster rather than a cluster.

3.3.4. Cluster splitting and merging

The operations of cluster splitting and merging enable DSS to scale to a large number of peers. Let M represent the maximum cluster size. If the size of a cluster exceeds M, the splitting process is invoked to split the cluster into two. A simple way of cluster splitting is to partition a cluster into two clusters of equal size without considering load distribution in the two clusters such as in CHORD. To balance the load during splitting and merging, each node maintains a *CurrentLoad* which measures its current load in terms of the number of RDF triples and data indices it stores. When node x joins the network, it sends a join request message to an existing node, say y. If y falls into the same semantic cluster that x wishes to join, x joins the cluster by connecting to y if its cluster size is below M; otherwise y will direct the request to a node, say z, in the semantic cluster that x wishes to join, and x will

connect to z if its cluster size does not exceed M. If the cluster size exceeds M, node y or z (called an initial node) will initiate the splitting process. The initial node first obtains a list of all the nodes in this cluster which is sorted according to their *CurrentLoads*. Then it assigns these nodes in the list to the two sub-clusters alternatively. After splitting, we obtain two clusters with relatively equal load. The initial node is also responsible for generating a new cluster *ID* for each of the two sub-clusters. To obtain a new cluster ID, each node maintains a *bit split pointer* which indicates the next bit to be split in the n-bit binary string (where $n = k - m$). For example, in Fig. 3, we assume $m = 3$, $n = 2$ and there exists a cluster $C4$ in the network. Initially, the *bit split pointer* points to the most significant bit of the n-bit string. When cluster splitting occurs, the bit pointed to by the *bit split pointer* is split into 0 and 1, and therefore we obtain cluster *ID* $C4$ and $C6$ corresponding to the same semantic cluster *SC1*. The *pointer* is then moved forward to the next bit in the n-bit string. Cluster $C4$ or $C6$ can be further split into $C4$ and $C5$ or $C6$ and $C7$, and finally the bit split pointer is set to null indicating no cluster splitting is allowed. The same mechanism follows for the insertion of a new semantic cluster in DSS. A semantic cluster can be split into a maximum number of 2^n clusters. After splitting, a node updates its *cluster ID*, the *bit split pointer* as well as the neighbors list in both its own cluster and its adjacent clusters.

When node x leaves the network, it first checks whether its cluster size has fallen below a threshold $M_{\min}$. If the current size is above $M_{\min}$, x simply leaves the network by transferring its indices to a randomly selected node in its cluster. Otherwise, this cluster needs to be merged into one of its neighboring clusters within the same semantic cluster. The leaving node triggers cluster merging which is the inverse process of cluster splitting. To obtain the newly merged cluster *ID*, the *bit split pointer* moves backwards by 1 bit in the n-bit string, and the bit pointed to by the *bit split pointer* is set to 0. The nodes in the merged cluster need to perform the same updating as in the splitting process. For the selection of $M_{\min}$, a simple method is to let $M_{\min} = 1$ so that cluster merging is invoked when the last node in a cluster leaves. However, if there exists only one node in a cluster, this node may become a hot spot as all the nodes in its two adjacent clusters have links to it. The actual value of $M_{\min}$ should be determined by the statistics of node joining and leaving within this cluster. If the last node in a semantic cluster leaves, it initiates two messages to all the nodes in its two adjacent clusters informing them to update their neighbor lists. Subsequently, the semantic cluster will be removed from DSS.

3.4. The routing algorithm

In this section, we describe the routing operation in DSS. As described above, each node in DSS maintains a routing table with a set of node entries (in the form of a pair $\langle NodeID, ClusterID\rangle$) in its own cluster, two adjacent clusters and another two semantic clusters. It also keeps state information about its own cluster, consisting of a k-bit *ClusterID* (where $k = m + n$) which indicates the cluster it resides in and *ClusterSize* which specifies the current size of its cluster. The query routing process involves two steps: inter-cluster routing and intra-cluster routing. Upon receiving a query, node x first obtains the destination *Semantic Cluster ID* (denoted as D). Then node x will check whether D falls into its own semantic cluster by comparing D against the most significant m-bits of its *ClusterID*. If that is the case, x will flood the query to all the nodes in its own cluster and also forward the query to the nodes in its adjacent clusters corresponding to D. The first node in a cluster receiving the query is always responsible for flooding the query within its cluster and forwarding the query to its adjacent cluster. The forwarding processes are recursively carried out until all the clusters corresponding to D have been covered and all nodes in each of the clusters are reached. Every node, upon receiving a query, will check its local data repository and return the matched data and indices.

For example, as illustrated in Fig. 4a, if a query is initiated at *Peer 1* with $D = SC0$, *Peer 1* first forwards the query to its neighboring node in $C1$, and then floods the query to all the nodes in $C0$. The same process is repeated for cluster $C1$, $C2$ and $C3$. If D is not the semantic cluster that node x belongs to, say its adjacent semantic cluster, the query will be forwarded to D and flooded to all the clusters corresponding to D. For example, in Fig. 4a, a query generated at *Peer 2* with $D = SC3$ will hop through $C16$ and will be flooded in $C14$ and $C12$. If D neither falls into node x's own cluster nor its adjacent semantic cluster, x will rely on its *shortcuts* to route the query across clusters. A query can be routed to a semantic cluster which is closer to

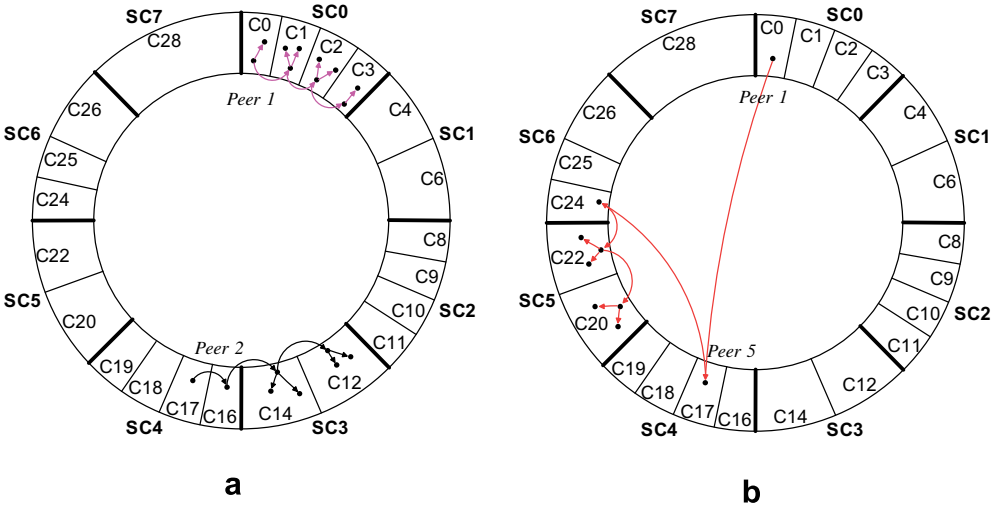


Fig. 4. Query routing.

the destination semantic cluster quickly with the help of these *shortcuts*.

In the design of these *shortcuts*, we have several design options. We need to decide which semantic cluster a *shortcut* should point to and how many *shortcuts* each node should maintain. An intuitive strategy is for a semantic cluster to select a set of other semantic clusters randomly and assign a *shortcut* to a node in each of these semantic clusters. Each node can have s *shortcuts* ($s \geqslant 1$) with the tradeoff that the cost of creating and maintaining these *shortcuts* is proportional to s. Upon receiving a query, if the distance between D and the semantic clusters that its *shortcuts* point to falls below a threshold – a preset minimum distance in terms of number of hops, the query will be forwarded to the closest semantic cluster and hop towards the destination semantic cluster. If not, x selects a *shortcut* randomly, and forwards the query to this *shortcut*. The same process is invoked until the distance to D is below the threshold. This approach is similar to Kleinberg's Small World network model in which a query can be routed to any node in $O(\log^2 n)$ hops.

Our approach is based on the observation that the ring space can be equally divided into several partitions. Each node maintains two *shortcuts* ($s = 2$) that are used to partition the ring space. For example, we can partition a 2^m semantic space where $m = 3$ into four by creating two *shortcuts*: one pointing to the opposite semantic cluster and another pointing to the semantic cluster located in a quarter of the ring space. Given the maximum cluster size M, the system can have a total of $M \cdot 2^{m+n-1}$ nodes when $M_{\min} = 1$. Let Cx denote the cluster where x resides in and SCx denote the semantic cluster that Cx corresponds to. SCx can be obtained by truncating Cx to m bits from the most significant bit. The two semantic clusters SC_{half} and SC_{quarter} that x's *shortcuts* point to are denoted as $(SCx + 2^i) \bmod 2^m$, where $i = m - 1$, $m - 2$. To initiate a search, x obtains D based on a query and checks which cluster range (partitioned by x's *shortcuts*) D falls into. Then node x forwards the query to the closer semantic cluster through its *shortcut*. If D is closer to SCx, node x will forward the query across its adjacent cluster towards D. A query takes a maximum of $2 + 2^{m-3}$ hops to reach the destination semantic cluster.

The above search algorithm is shown in Fig. 5. To illustrate, consider Fig. 4b, *Peer 1* generates a query and computes the destination semantic cluster as *SC5*. *Peer 1* first realizes that *SC5* falls into the interval [*SC4*, *SC0*] and *SC4* is close to *SC5*. Then *Peer 1* forwards the query to *Peer 5* at *C*17. As *SC5* falls into [*SC4*, *SC6*] and *C*24 is closer to *SC5* as compared to *C*17, *Peer 5* forwards the query to *SC6* through its quarter *shortcuts*. Finally, the query reaches *SC3* and is then flooded in both *C*22 and *C*20.

The more *shortcuts* we create to partition the ring space, the finer the granularity we gain to locate the destination semantic cluster. As a result, we achieve better search performance in terms of lower routing hops. However, more *shortcuts* imply higher cost of creating, updating and maintaining these *shortcuts*. In DSS, we set the number of *shortcuts* to two for the reason of minimizing maintenance cost. To

```
Assuming the longest shortcut point to 1/p (p=2^i, i=0,1,…) of the ring;
Obtain the destination semantic cluster D based on the query q;
if dist(SC_x, D) ≤ 2^m/(4p) then
  forward q to x's adjacent cluster towards D;
else if D falls into [SC_x, SC_quarter] or [SC_quarter, SC_half] or [SC_half, SC_x]
then
  forward q to the semantic cluster that is
  closer to D
end if
```

Fig. 5. Pseudocode of the search algorithm.

partition the ring space in a finer granularity when the number of semantic clusters m increases, we can place the longest *shortcut* into different points in DSS. The other *shortcut* always points to the middle semantic cluster between *SCx* and the semantic cluster that the longest *shortcut* points to. For example, if we place the longest *shortcut* to one-quarter of the ring, the ring space is divided by eight, and so on. More generally, the following theorem obtains the search path length for DSS.

Theorem 1. *Given a m-dimensional DSS of N nodes, with maximum cluster size M, number of bits to identify sub-cluster n and number of shortcuts s, the average path length for routing across semantic clusters is* $O\left(\frac{1}{s}\log^2(N/M\cdot 2^{n-2})^{1/m}\right)$.

Proof. We follow a process similar to that in [17] to prove the theorem. In [17], Kleinberg proved that the optimal setting for shortcuts is $f_x = 1/x^m$, where m is the dimensionality. Thus, in DSS, a peer chooses another peer at distance x as one of its *shortcuts* using the pdf: $f_x = 1/x^m$ for $x\cdot\in[r,1]$ where r, the minimum distance of a *shortcut*, is the average diameter of a semantic cluster. The average size of a semantic cluster is $\frac{M}{2}2^{n-1}$, there are altogether $N/M\cdot 2^{n-2}$ semantic clusters in the system, and each semantic cluster takes charge of a $M\cdot 2^{n-2}/N$ portion of the whole semantic space on average. Therefore, the diameter of each partition r is approximately $(M\cdot 2^{n-2}/N)^{1/m}$.

We extend the small world network model from two-dimensional space to m-dimensional space. We use unit data space in DSS. Since each subspace has side length r on average, there are $1/r$ subspaces along each side. The distance between two clusters along a dimension is the range of $[1,2,\ldots,1/r]$. Thus, we separate the search process into phases $1,2,\ldots,\log(1/r)$. Let d be the distance from a query message's current node to the destination, and $d_i = 1/2_i$. Search is at phase i if $d_{i+1}\leqslant d < d_i$. Phase i ends when the message is forwarded to a peer less than d_{i+1} distance away from the destination. The set of peers less than d_{i+1} distance away from the destination is denoted as D_{i+1}, whose volume is d^m_{i+1}. The largest distance from a peer at phase i to a peer in set D_{i+1} is $d_i + d_{i+1}$. Since a peer has s *shortcuts*, the probability that a peer at phase i has contacts to set D_{i+1} is at least $s\cdot d^m_{i+1}\cdot f_{d_i+d_{i+1}} = s/c\cdot\log(1/r)$ where c is a constant that depends on m. Therefore, a query message requires $c\cdot\log(1/r)/s$ steps to reach the next phase on average. Since there are in total $\log(1/r)$ phases, the total search path length is $O(\frac{1}{s}\log^2(N/M\cdot 2^{n-2})^{1/m})$. □

3.5. Subscription

In addition to search requests which pull data from the network, DSS enables consumers to issue subscription requests to the network and be notified when data changes occur over a period of time. When a subscription request is generated, it will be first mapped to a semantic cluster (say D) and then forwarded to all nodes in D. The mapping and routing processes of a subscription request are identical to those of a search request. When a node in D receives a subscription request, it will check its local RDF data and decide whether it should accept the request. For example, an application may subscribe the event "*John is in the bedroom*" in the RDF triple form of ⟨*socam*[1] *:John socam:locatedIn socam:Bedroom*⟩ to the network. As this RDF triple may not exist in the network (i.e., John may be in

[1] *socam* is a namespace. Please refer to the SOCAM project Web site at http://www1.i2r.a-star.edu.sg/~tgu for more details.

some other place) at the time of receiving a request, the subscription request may end up with no producers. To avoid losing potential producers or ending up with many irrelevant producers, we employ the following subscription acceptance policy, as shown in Fig. 6, and illustrate how it works in context-aware computing and sensor network domains.

Based on this policy, a producer peer attempts to match a subscription request against its local RDF data. This policy works for a subscription request in the form of any RDF triple pattern whose subject, predicate or object may take variables. Although predicates can be specified as variables, this situation seldom occurs since users or applications are always in favor of more specific events in real-life applications. We now consider the case that a predicate is specified in a subscription request. If a subscription request's predicate is of type *DatatypeProperty*, a producer peer determines if its local RDF data contains triple(s) with the same subject–predicate pair as the request. For example, for a given subscription request ⟨*socam:Bedroom socam:lightLevel 'LOW'*⟩, a producer peer will accept the request if there exists an RDF triple with subject *socam:Bedroom* and predicate *socam:lightLevel* in its local data. If a subscription request's predicate is of type *ObjectProperty*, a producer peer determines if its local RDF data contains triple(s) with the same predicate–object pair as the request. For example, for a given subscription request ⟨*socam:John socam:locatedIn socam:Bedroom*⟩, a producer peer will accept the request if there exists an RDF triple with subject *socam:locatedIn* and predicate *socam:Bedroom* in its local data.

To understand the rationale behind this technique, consider a subscription request in the form of the RDF triple $\langle Sub_s, Pred_s, Obj_s \rangle$. Such a triple may be obtained from raw data generated by a sensor which could be physical or virtual. In the domain of sensor networks, a predicate always corresponds to a sensor type. For example, *socam:locatedIn* corresponds to a physical location sensor and *socam:participateIn* corresponds to a virtual activity sensor. If $Pred_s$ is of *DatatypeProperty*, Sub_s should correspond to the target this sensor is monitoring, while Obj_s corresponds to the sensor output. For example, the RDF triple of ⟨*socam:Bedroom socam:lightLevel 'LOW'*⟩ can be interpreted as the output of a light level sensor monitoring the bedroom's light level. If a producer peer's local RDF data contains at least one triple with this Sub_s-$Pred_s$ pair, it can be inferred that this producer peer has the type of sensor specified by this pair. Hence, we can conclude that this producer peer can provide triples of this same subject–predicate pair. On the other hand, if $Pred_s$ is of *ObjectProperty*, Obj_s should correspond to the target this sensor is monitoring, while Sub_s corresponds to the

```
Given a subscription request in the form of an RDF triple pattern <Subs,
Preds, Objs>, a variable in the RDF triple represents any arbitrary constant;
Let <Sub1, Pred1, Obj1> represents any RDF triple in a peer's local data set
called L.

accept = false; //initialization
for each RDF triple in L
  if Preds is of DatatypeProperty && ((Subs == Sub1)∩ (Preds == Pred1))== true
then
    accept = true; break;
  else if Preds is of ObjectProperty && ((Preds == Pred1) ∩ (Objs == Obj1)) ==
true then
    accept = true; break;
  end if
end for
if accept == true then accept the subscription request;
else reject the subscription request;
end if
```

Fig. 6. Subscription acceptance policy.

sensor output. In this case, the producer can provide triples with the same Sub_s-$Pred_s$ pair as in the subscription request.

Once a producer peer accepts a subscription request, it keeps monitoring the request. Whenever a change (i.e., an RDF triple is added or removed) occurs, the producer peer will notify the subscribers if the RDF triple matches the subscription request. An RDF triple $\langle Sub_c Pred_c Obj_c \rangle$ is said to match the subscription request if $(Sub_c = Sub_s) \cap (Pred_c = Pred_s) \cap (Obj_c = Obj_s) = 1$. The routing of notification traces the exact path of the subscription request in the reverse direction. A subscriber can unsubscribe an event by sending an unsubscription request directly to its producers.

3.6. Peer dynamics and failure

In dynamic environments, a node may join and leave the system freely. To keep track of its neighboring nodes in DSS, a node maintains a number of additional backup links for every link a node has. The approach is used in many other P2P systems such as Pastry and CAN. However, in a highly dynamic environment, detecting link failure during the query routing process can introduce additional overhead. Moreover, in the event of failure of all its backup links, a node has to re-establish its neighboring links during the search operation, and hence it may affect search performance. With this approach, a node will need to inform its neighboring nodes about its leaving and transfer its indices to a randomly selected node in its cluster before leaving. Another approach is that each node periodically sends a keep-alive message to each neighboring node such as the ping message in Gnutella-like overlay networks. If no response is received, the neighboring node is assumed to be dead and a new link needs to be established. The failure detection is done in an off-line manner to avoid affecting search performance, but it may increase the overall traffic. In this approach, a node is not required to inform its neighboring nodes before its leaving. A node leaves the system by simply transferring its indices. In the above two approaches, when a node is involved in subscription, it has to transfer its back route information to a node in its cluster or inform the subscriber about its leaving. Both the above two approaches have their pros and cons, and require a good study on the justification when they are applied to a real-life application. In the following evaluations, we rely only on the backup states to study how well DSS performs in the presence of failure.

4. Evaluation

We use simulation to evaluate the effectiveness of DSS and compare DSS with SONs [12]. We show the performance results by setting various variables such as m, n, M and *shortcut* positions, and justify our choices. We first describe our simulation model and the performance metrics. Then we report the results obtained from a range of experiments.

4.1. Simulation model and metrics

To simulate the performance of DSS in a more realistic environment, we create two types of network topologies in our model: physical topology and P2P overlay topology. All peer nodes are a subset of nodes in the physical topology. We use the AS model to generate these topologies as previous studies have shown that P2P overlay topologies [20] follow the small world and power law properties.

The simulation is started by having a pre-existing node in the network and then performing a series of join operations invoked by new arriving nodes. A node joins a semantic cluster based on its local data and publishes its data indices. Various RDF data are mapped into different semantic clusters and each semantic cluster is associated with a unique ID ranging from 0 to 2^m. RDF data stored in each peer may be heterogeneous or homogeneous. To evaluate the capability of handling heterogeneous data in DSS, we introduce a parameter β, which is the ratio of the number of semantic clusters corresponding to all the local data stored in a peer to the maximum number (2^m) of semantic clusters. β falls into the range of $1/2^m$ to 1. When $\beta = 1/2^m$, it implies that a peer has homogeneous RDF data in its local repository which maps to one particular semantic cluster in DSS. When $\beta = 1$, it implies that a peer has heterogeneous RDF data which maps to all the semantic clusters; however, this case is unlikely to occur in real-life applications. In our experiments, we set β to $1/2^m$, 0.25 and 0.5 respectively. The semantic cluster(s) are selected in random by each peer according to β.

A peer also selects a random node in each of the semantic clusters to publish its indices, if necessary. When the size of a semantic cluster exceeds the maximum size M (in nodes), it will be split into two. This operation may be performed recursively until the number of sub-clusters reaches 2^n. After the net-

work reaches a certain size, a mixture of node joining and leaving are invoked to simulate the dynamic characteristic of the overlay network. Each node is assigned a query generation rate, which is the number of queries that it generates per unit time. In our experiments, each node generates queries at a constant rate. If a node receives queries at a rate that exceeds its capacity to process them, the excess queries are queued in its buffer until the node is ready to read the queries from the buffer. Data are randomly replicated on nodes at a fraction α. Thus, querying for data with fraction α implies that a query hit can be found at a fraction α of all the nodes in the system. A query is selected randomly among different semantic dimensions.

When a node initiates a query, it is first mapped to a particular semantic cluster, and then routed to the destination semantic cluster and flooded to all the sub-clusters in parallel. In our simulation study, we use a Gnutella overlay network to organize nodes within a cluster. The average outgoing degree of a node in its cluster is set to 4 by default, and *shortcuts* are set to the half and quarter of the ring space unless specified. For the simplicity of generating RDF data in our simulation model, we use a set of keywords to represent RDF data triples; different sets of keywords correspond to different semantic clusters. In our simulation, we use the following performance metrics to measure the effectiveness of DSS:

Fraction of nodes contacted per query is the average fraction of nodes contacted for a query. It captures the efficiency of a lookup system. A smaller fraction of nodes implies less overhead in the network.

Search path length is the average number of hops traversed by a query to the destination.

Search cost is the average number of query messages incurred during a search operation in the network.

Maintenance cost is the average number of messages incurred when a node joins or leaves the network. It consists of the costs of node joining and leaving, cluster splitting and merging and index publishing. We measured these costs in terms of number of messages.

Routing load is the average number of query messages a node processes.

4.2. Semantic cluster mapping

For the evaluation of the semantic clustering mapping, we have implemented a working prototype based on the Jena 2 toolkit [9]. We created a set of RDF-based data which corresponds to each of the domain-specified ontologies and various query patterns. The system performs the mapping process by iterating each data triple or query. We ran the prototype on a 2.0 GHz Pentium machine with 1 GB of memory. We used 1000 different data triples and 1000 different queries in this experiment. On average, the mapping process takes about 3.38 s for the mapping initialization and 0.234 ms for the mapping of each data triple and each query. The mapping initialization reads the ontology files stored locally and generates internal data structures for mapping. It is done only once when a peer starts and is only repeated if there is a change in the ontologies. The computation cost of our mapping process is much lower as compared to the computation cost of LSI (results can be found in [13]).

4.3. Search efficiency

The efficiency of executing a search request is captured in the fraction of nodes contacted and search path length during the search. For a given query, DSS only needs to contact $N/2^m$ nodes in the system plus those nodes pointed to by a set of indices. Fig. 7 plots the fraction of nodes contacted per query by setting n to 0 (i.e., parallel search in a semantic cluster is disabled) and varying the number of semantic clusters from 2^0 to 2^8. The values are obtained by taking the average over various network sizes N from 2^8 to 2^{13}. As expected, the fraction of nodes contacted per query decreases in proportion to $1/2^m$. When $\beta = 0.25$ or 0.5, DSS has to contact about a quarter or a half of nodes. This is because that, besides contacting all the nodes

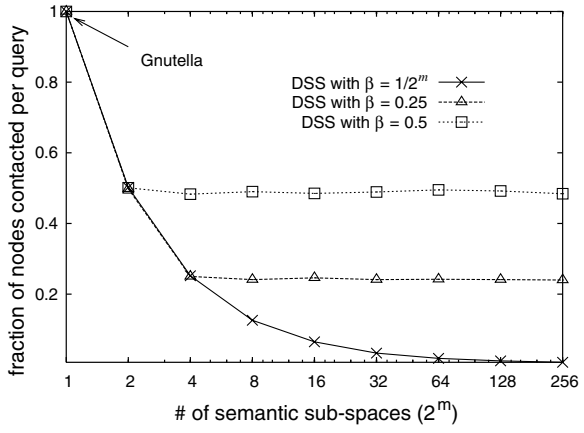


Fig. 7. Fraction of nodes contacted per query.

in the destination semantic cluster, DSS has to contact the nodes in other semantic clusters pointed to by their indices. Due to randomness of selecting semantic clusters and nodes to publish its indices by a peer, the fraction of nodes contacted is almost identical to β. Note that for a search request, Gnutella has to contact every node in the network. In the case of SONs, this fraction is equal to $\bar{C}/C_{\max}$, where $\bar{C}$ is the average number of SONs each node participates in and $C_{\max}$ is the maximum number of SONs in the system. With less nodes contacted by DSS and SONs, the network traffic load incurred by a query will also be reduced. The result confirms that DSS has the same efficiency as SONs in terms of number of nodes contacted per query.

Fig. 8 shows the search path length comparing DSS, SONs and Gnutella when the network size N is varied from 2^8 to 2^{13}. We disabled the clustering effect by setting M to 1 for DSS since Gnutella does not have a clustering feature. We also disabled parallel search in DSS by setting n to 0. Hence, the network size N is 2^{m-1}. Since $M = 1$ and $n = 0$, there will be no flooding within a semantic cluster. As shown in Fig. 8, the search path lengths for both DSS and SONs increase slowly with the network size as compared to Gnutella, confirming that the search path is bound (note that the x-axis uses a log scale). The search path length for DSS is almost identical to the one for SONs, showing that DSS has the same search effectiveness as SONs. In the case that a peer has heterogeneous local data (i.e., $\beta = 0.25$ or 0.5), the search path length is almost identical to the case that a peer has homogeneous local data (i.e., $\beta = 1/2^m$). It shows that it does not have any negative effect on DSS in terms of search path length when a peer has heterogeneous local data. This is because a peer can directly contact the node(s) in other semantic clusters using a set of indices that point to them.

In DSS, we explore the parallel search mechanism within a semantic cluster. We evaluated the parallel search effect by comparing DSS and SONs. We set up a network with $m = 4$ and varied the network size from 2^{10} to 2^{13}. We set n to 2 and 3 respectively for DSS; as a result, a semantic cluster will be split into two when the size exceeds $N/2^5$ and $N/2^6$. Hence, a search can be performed in parallel among these sub-clusters. Fig. 9 shows that the parallelism in DSS has effectively reduced search path length as compared to SONs. The result also shows that the parallel search effect increases (i.e., search path length decreases) with respect to n. The results in both Figs. 8 and 9 show that the search path length in DSS is sensitive to 2^m and n, but not sensitive to β.

4.4. Overhead

In this experiment, we evaluated search overhead by comparing search costs among DSS, SONs and Gnutella. We set m to 5 (i.e., the number of semantic clusters is 32), n to 0 (parallel search is disabled), and varied the network size from 2^8 to 2^{13}. As shown in Fig. 10, the search cost of Gnutella increases rapidly when the network size grows. In contrast, DSS and SONs significantly reduce the search cost with the setting of 32 semantic clusters. We repeated the experiment by turning on the parallel search mechanism (i.e., $n = 2$ and 3) while keeping other settings. We obtained similar results as in the case where $n = 0$. This confirms that the parallel search mechanism in DSS does not incur

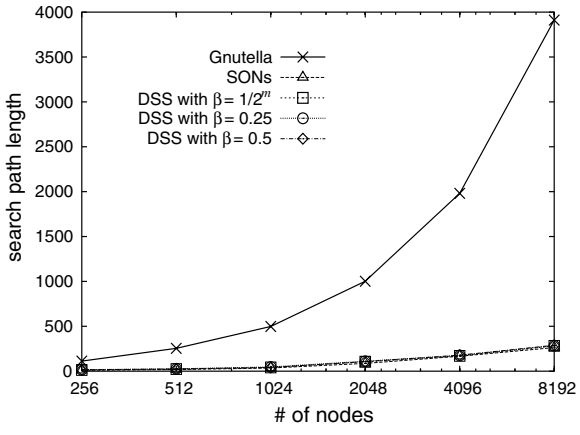


Fig. 8. Search path length.

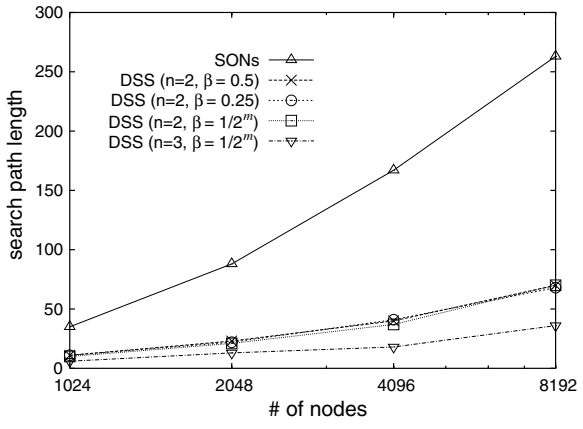


Fig. 9. The effect of parallel search in DSS.

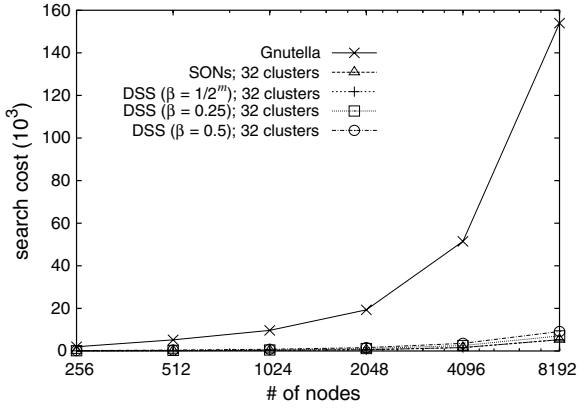


Fig. 10. Search cost.

any extra search overhead. When $\beta = 0.25$ or 0.5, search cost increases because search requests have to reach many nodes in other semantic clusters (other than D) as well, which are pointed to by a set of indices.

In this experiment, we evaluated the average maintenance cost by comparing DSS and SONs. The maintenance cost of SONs only contains the cost of node joining and leaving. As shown in Fig. 11, the maintenance cost of SONs increases rapidly when the number of semantic clusters (dimensions) grows. This is because the required number of outgoing degrees of a node in SONs increases in proportional to the dimension. In the case of DSS with $M = 32$ and $n = 2$, the average maintenance cost of a node consists of the costs of node joining and leaving, cluster splitting and merging and index publishing. The maintenance cost in DSS also increases with respect to the dimension, but in a much lower gradient. In the case of heterogeneous data stored in peers (i.e., $\beta = 0.25$ or 0.5), the maintenance cost increases due to the increased cost of index publishing; however, it is still much lower than SONs as shown in Fig. 11. This confirms our design goal of reducing maintenance overheads incurred by high-dimensional semantic overlay networks such as in SONs.

4.5. Clustering effects

In this section, we evaluate the effect of clustering in DSS by varying the cluster size M from 2^0 to 2^{10}. We first evaluate the effect of cluster size on search path length by constructing a network of size $N = 2^{10}$. We turn off parallel search within a semantic cluster by setting n to 0, and ensure no data duplication in DSS. Hence all clusters are semantic clusters. We also set β to $1/2^m$ as we focus on cluster operations in this section. Fig. 12 plots the search path length in DSS when M increases from 2^0 to 2^{10}. The search path length across clusters decreases while the search path length within clusters increases with larger cluster sizes (note that there are 2^{10} clusters in the network when $M = 1$ and only one cluster when $M = 2^{10}$). This is because with a fixed network size, the total number of clusters in DSS decreases with larger cluster sizes. Fig. 12 suggests that the search path length achieves its minimum when the number of semantic cluster equals to 32, 16 and 8 corresponding to $M = 32$, 64 and 128 respectively.

With the same settings as in the previous experiment, we evaluated the search cost and its breakdown within clusters and across clusters with various cluster sizes. From Fig. 13, we observe that the search cost in DSS increases rapidly from a

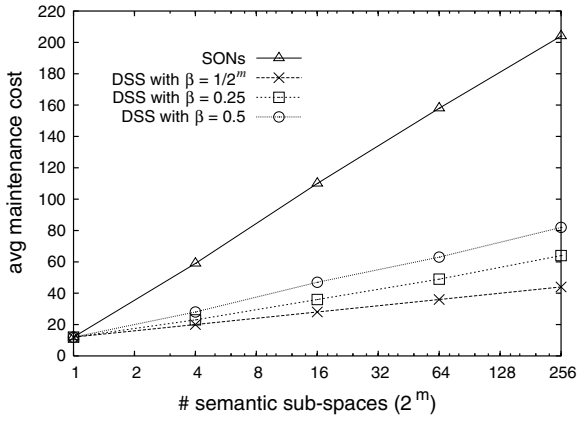


Fig. 11. Average maintenance cost.

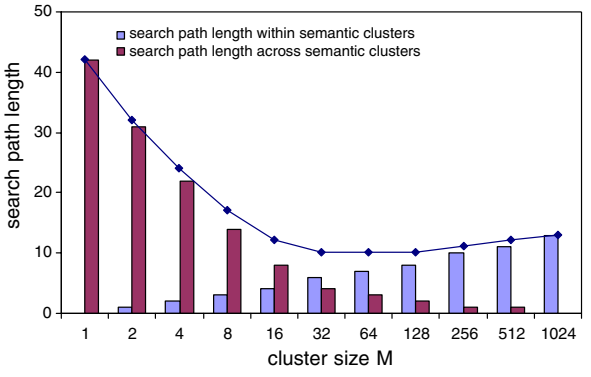


Fig. 12. Search path length vs. cluster size M.

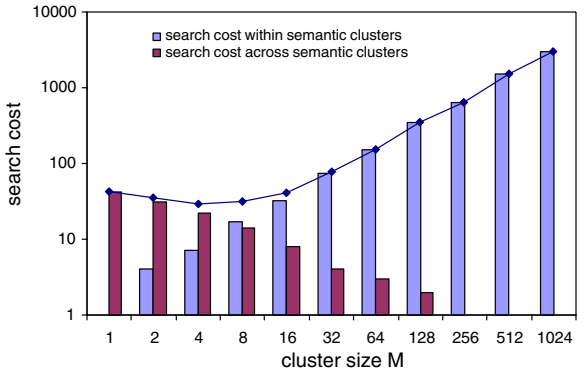


Fig. 13. Search cost vs. cluster size.

point where $M = 16$. This is due to the effect of blind flooding within a cluster.

We plot the cost of node joining/leaving and cluster splitting/merging over different cluster sizes in Fig. 14. As there are fewer clusters in DSS with larger cluster sizes, a new node requires a smaller number of hops to join the network. Therefore the cost of joining/leaving decreases with respect to M. With a larger cluster size, cluster splitting and merging occur less frequently, resulting in a lower cluster splitting/merging cost.

From the results in this section, we observe that the setting of 16 and 32 semantic clusters provides a good tradeoff between search efficiency and overhead. With larger cluster sizes, the search path length and the cost of node joining/leaving and cluster splitting/merging are not so sensitive to M as compared to the search cost. One should notice that we set n to 0 in the experiments in this section. If the parallel search mechanism is turned on ($n > 0$), the search path length can be further reduced as a query can be flooded in parallel in a semantic cluster. To further reduce the search cost incurred by blind flooding within a cluster, the Cost-Aware Selective Flooding technique proposed in [7] can be deployed.

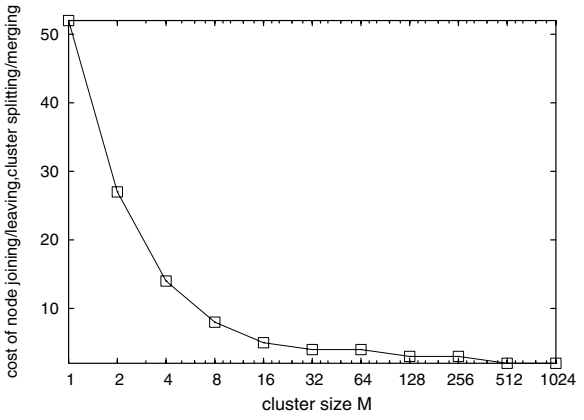


Fig. 14. The cost of node joining/leaving and cluster splitting/merging vs. cluster size.

4.6. Selection of shortcuts

In this experiment, we evaluated the effect of different *shortcuts* in DSS and compared them to the random *shortcut* which is originally used in a small world network model. We started a network with the size of 2^{10} nodes. Each semantic cluster has only one node by setting the cluster size M to 1 and n to 0. Hence, the search path length for intra-cluster routing equals to 0. We selected two *shortcuts* either fix-points or random-points in the network and varied the location of the longest *shortcut*. The other *shortcut* always points to the middle semantic cluster between the semantic cluster where a node resides and the semantic cluster that the longest *shortcut* points to. We plot the search path length for inter-cluster routing with various numbers of semantic clusters in Fig. 15. As compared to fix-point *shortcuts*, random *shortcuts* work well in lower dimensional semantic spaces, but perform worse in higher dimensional semantic spaces. The location of fix-point *shortcuts* depends on the number of semantic spaces. Among these *shortcuts*, the 1/8 *shortcut* seems to provide a balance for the size of semantic spaces below 512.

We evaluated the effect of clustering in DSS by varying the cluster size M. The results suggest that the setting of 16 and 32 semantic clusters provides a good tradeoff between search efficiency and overhead. With larger cluster sizes, the search path length and the cost of node joining/leaving and clus-

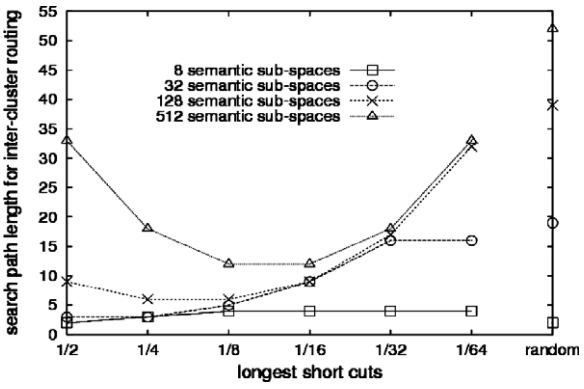


Fig. 15. Selection of shortcuts.

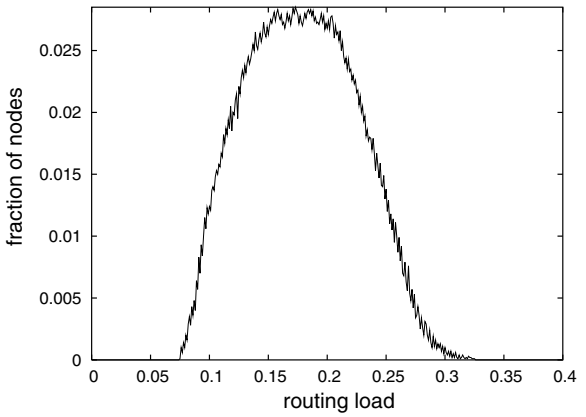


Fig. 16. Routing load.

ter splitting/merging are not as sensitive to M as compared to the search cost.

4.7. Load balancing

We study the load balance in DSS from the aspects of data load, index load and routing load. Since the data load in terms of number of data triples and the index load in terms of number of indices are balanced under the uniform distribution of data, we present only the result of routing load in this section. We evaluate the routing load processed per node in a network setting $m = 3$, $n = 2$ and $M = 64$. The average outgoing degree per node is set to 4 within a semantic cluster. A query is drawn randomly from all the semantic clusters. Each node initializes a lookup uniformly at random. Fig. 16 shows that the routing load distribution across various nodes is relatively well balanced.

5. Conclusion and future work

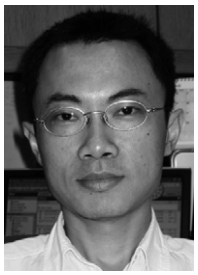
In this paper, we present a schema-based P2P system for information retrieval in dynamic environments. We propose several techniques such as ontology-based semantic clustering, a cluster naming scheme and routing techniques and show how to retrieve RDF data in both pull and push modes. These techniques can be well applied to any P2P searching systems where schemas are explicitly defined. The promising results from a range of experiments show that DSS works effectively and has a good tradeoff between search efficiency and search cost. The overlay maintenance cost is low, and the system adapts to peer dynamics quickly and can scale to a large number of peers.

This paper uses a standardized definition of the upper ontology and the low-layer ontologies for overlay construction. This assumption may restrict the use of DSS in real-life applications since users can only create RDF data according to pre-defined ontologies. However, since there are many on-going efforts to create standard ontologies for various application domains, e.g., in e-commerce applications [21], and ubiquitous and pervasive applications [22], we believe DSS will be widely applied in many real-life dynamic applications. If ontology interoperability mechanisms are put in place, that will offer a greater flexibility for our scheme. While this paper assumes the use of Gnutella-like overlay networks to organize peers within a sub-cluster, a DHT-based overlay network can be used to provide more efficient routing with the tradeoff of higher maintenance overhead. We are studying how to apply DHT-based routing techniques such as Chord or CAN to a sub-cluster in DSS while keeping maintenance overhead low. Finally, we believe that DSS can have a significant practical impact on building large-scale, schema-based P2P lookup systems in dynamic environments.

References

[1] World Wide Web Consortium: Resource Description Framework. <http://www.w3.org/RDF>.
[2] Gnutella. <http://gnutella.wego.com>.
[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of ACM SIGCOMM, San Diego, US, August 2001.
[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in: Proceedings of ACM SIGCOMM, San Diego, US, August 2001.
[5] A. Rowstron, P. Druschel, Pastry: scalable. Distributed object location and routing for large-scale peer-to-peer systems, Lecture Notes in Computer Science 2218 (November) (2001) 161–172.
[6] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004) 41–53.
[7] T. Gu, E. Tan, H.K. Pung, D. Zhang, ContextPeers: scalable peer-to-peer search for context information, in: Proceedings of International Workshop on Innovations in Web Infrastructure, in conjunction with the 14th World Wide Web Conference (WWW 2005), Japan, May 2005.
[8] RDFStore. <http://rdfstore.sourceforge.net>.
[9] Jena 2– A Semantic Web Framework. <http://www.hpl.hp.com/semweb/jena2.htm>.
[10] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Lser, Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks, in: Proceedings of the12th World Wide Web Conference, May 2003.

[11] P.A. Chirita, S. Idreos, M. Koubarakis, W. Nejdl, Publish/subscribe for RDF-based P2P networks, in: Proceedings of the 1st European Semantic Web Symposium, Greece, May 2004.
[12] A. Crespo, H. Garcia-Molina, Semantic overlay networks for P2P systems. Technical report, Stanford University, January 2003.
[13] M. Cai, M. Frank, RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network, in: Proceedings of the 13th International World Wide Web Conference, New York, US, May 2004.
[14] C.Q. Tang, Z.C. Xu, S. Dwarkadas, Peer-to-peer information retrieval using self-organizing semantic overlay networks, in: Proceedings of ACM SIGCOMM, Karlsruhe, Germany, August 2003.
[15] S.C. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, R.A. Harshman, Indexing by latent semantic analysis, Journal of the American Society of Information Science 41 (6) (1990) 391–407.
[16] M. Li, W.C. Lee, Anand Sivasubramaniam, D.L. Lee, A small world overlay network for semantic based search in P2P, in: Proceedings of the 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing, in conjunction with the World Wide Web Conference, May, 2004.
[17] J. Kleinberg, The Small-World Phenomenon: an Algorithm Perspective, in: Proceedings of the 32nd ACM Symposium on Theory of Computing, 2000.
[18] RDQL. <http://www.w3.org/Submission/2004/SUBM-RDQL- 20040109/>.
[19] M. Berry, Z. Drmac, E. Jessup, Matrices, vector spaces, and information retrieval, SIAM Review 41 (2) (1999) 335–362.
[20] S. Saroiu, P. Gummadi, S. Gribble, A measurement study of peer-to-peer file sharing systems, in: Proceedings of Multimedia Computing and Networking, San Jose, CA, US, January 2002.
[21] OntoWeb Special Interest Groups (SIGs) on Content Standardization for B2B E-Commerce. <http://zeus.ics.forth.gr/forth/ics/isl/projects/ontoweb/content_standards.html>.
[22] SOUPA – Standard Ontology for Ubiquitous and Pervasive Applications. <http://pervasive.semanticweb.org>.



Tao Gu (http://www1.i2r.a-star.edu.sg/~tgu) is a Scientist at the Institute for Infocomm Research, Singapore. He received his Ph.D. in computer science from National University of Singapore in 2005. His current research interests involve various aspects of ubiquitous and pervasive computing including architectures, tools, distributed lookup algorithms, and data management. Other areas of interest are Peer-to-Peer computing and Wireless Sensor Networks.

Hung Keng Pung (http://www.comp.nus.edu.sg/~punghk) is an associate professor in the Department of Computer Science at the National University of Singapore. He received his Ph.D. from the University of Kent at Canterbury, UK. He heads the Networks Systems and Services Laboratory as well as holding a joint appointment as a Principal Scientist at Institute of Infocomm Research, Singapore. His research focuses on context-aware systems, service-oriented computing, quality of service management, protocol design and networking.

Daqing Zhang is a Principal Investigator at the Institute for Infocomm Research (I2R), Singapore. He received his Ph.D. from University of Rome "La Sapienza" and University of L'Aquila, Italy in 1996. He has been initiating and leading the efforts in Smart Home, Context-aware Middleware, Context-aware Mobile and Healthcare Applications in I2R, Singapore. His research interests include smart home, pervasive computing, service-oriented computing, context-aware systems and ambient intelligence.