SSW: A Small World Based Overlay for Peer-to-Peer Search

Mei Li[†], Wang-Chien Lee[†], Anand Sivasubramaniam[†], and Jing Zhao[‡]

[†] Pennsylvania State University, University Park, PA, 16802, USA

[‡] Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China

[†] {*meli, wlee, anand*}*@cse.psu.edu,* [‡] {*zhaojing*}*@cs.ust.hk*

Abstract

Peer-to-peer (P2P) systems have become a popular platform for sharing and exchanging voluminous information among thousands or even millions of users. The massive amount of information shared in such systems mandates efficient semantic based search instead of key-based search. The majority of existing proposals can only support simple key based search rather than semantic based search. This paper presents the design of an overlay network, namely *semantic small world (SSW)*, that facilitates efficient semantic based search in P2P systems. SSW achieves the efficiency based on the following four ideas: 1) semantic clustering: peers with similar semantics organize into peer clusters; 2) dimension reduction: to address the high maintenance overhead associated with capturing high-dimensional data semantics in the overlay, peer clusters are adaptively mapped to a one-dimensional naming space; 3) small world network: peer clusters form into a one-dimensional small world network, which is search efficient with low maintenance overhead; 4) efficient search algorithms: peers perform efficient semantic based search, including approximate point query and range query, in the proposed overlay. Extensive experiments using both synthetic data and real data demonstrate that SSW is superior to the state-of-the-art on various aspects, including scalability, maintenance overhead, adaptivity to distribution of data and locality of interest, resilience to peer failures, load balancing, and efficiency in support of various types of queries on data objects with high dimensions.

Index terms: Distributed systems, distributed indexing, overlay, Internet content sharing, peer-to-peer systems

I. INTRODUCTION

Peer-to-peer (P2P) systems have become a popular platform for sharing and exchanging resources and voluminous information among thousands or even millions of users. In contrast to the traditional client-server computing models, a host node in P2P systems can act as both a server and a client. Despite avoiding performance bottlenecks and single points of failure, these

Digital Object Indentifier 10.1109/TPDS.2007.70757

decentralized systems present fundamental challenges when searching for resources (e.g., data and services) available at one or more of these numerous host nodes.

These challenges have motivated the proposals of distributed hash tables (DHTs), e.g., CAN [19], Chord [22], Pastry [21], and Tapestry [25], which use hashed keys to direct the searches to the specific peer(s) holding the requested data objects. While these techniques address the scalability issue with respect to the number of peers in the P2P system, it is equally important to address the voluminous information content of such systems. Given the vast repositories of information (e.g., documents) shared in peer-to-peer systems, it is undesirable to require users to remember the key or identifier associated with a document in order to search for such a document. Instead, it is more favorable to allow users to issue searches based on the content of documents (just as in the Internet today). This mandates the employment of *content/semantic-based searches*[1]. The primary goal of this study is to design a P2P overlay network that supports efficient semantic based search.

Various digital objects, such as documents and multimedia, can be represented and stored as data objects in P2P systems. The semantics or features of these data objects can be identified by a k-element vector, namely, *Semantic Vector (SV)* (or called *feature vector* in the literature). Semantic vectors can be derived from the content or metadata of the objects. Each element in the vector represents a particular feature or attribute associated with the data object with weight representing the importance of this feature element in representing the semantics of the data object. The SV of a data object can be mapped to a point in a k-dimensional space. Thus, each data object can be seen as a point in a multi-dimensional *semantic space*. As a result, queries on data objects in this semantic space can be specified in terms of these attributes. The number of attributes (elements) capturing the semantics of data objects is normally very large, and thus the dimensionality of the semantic space (k) is high. For instance, the dimensionality of semantic vector for documents (latent semantic vector [6]) is around 50-300. Euclidean distance is used to represent the *semantic distance* between two SVs while SVs are assumed to be unit vectors.

There are several challenges faced by realizing our goal to design a P2P overlay network that supports efficient semantic based search. 1) Based on the accumulated knowledge of *clustered indexes* in database research community, it is safe to assume that clustering data objects with similar semantics close to each other and indexing them in certain attribute order can facilitate efficient search of these data objects based on indexed attributes. Thus, to support efficient

[1]We do not exploit the differences between semantic and content based searches. These two terms are used interchangeably in the paper.

semantic based search, the peer hosts and data objects should be organized in accordance with the semantic space that they are located in. 2) For many real life applications, the number of attributes used to identify data objects and to precisely specify queries is large. Thus, a well designed P2P overlay network needs to be able to facilitate efficient navigation and search in a *high dimensional space* without incurring high maintenance overhead. 3) The P2P overlay network of our goal mandates all the good properties of a robust network such as scalability, load balance, and tolerance to peer failures. 4) Various types of queries that might be issued by users should be efficiently supported in the overlay.

This paper presents the design of a P2P overlay network, called *Semantic Small World (SSW)*, which overcomes the above challenges to facilitate semantic based search. The primary contributions of this work are five-fold. 1) We adopt an effective semantic clustering strategy that places peers based on the semantics of their data. 2) We show a way to build a small world overlay network for semantic based P2P search, which is scalable to large network sizes yet adapts to dynamic membership and content changes. 3) To address the high maintenance overhead associated with the high dimensionality of semantic space, we propose a dimension reduction technique, called *adaptive space linearization* (ASL), for constructing a one-dimensional SSW (called *SSW-1D*). 4) We propose efficient algorithms to support various types of queries, including approximate point query and range query, in the framework of SSW. 5) We conduct extensive experiments using both synthetic data set and real data set to evaluate the performance of SSW on various aspects, including scalability, maintenance, adaptivity to data distribution and query locality, resilience to peer failures, load balancing, and query performance.

The rest of this paper is structured as follows. Background on small world network and related work are provided in Section 2. In Section 3, we provide an overview on the design of SSW. The details on dimension reduction and search algorithms are given in Section IV and V, respectively. Performance evaluation is presented in Section VI-B. Finally, we conclude this paper and outline the directions for future research in Section VII.

II. Preliminaries

A. Background on Small World Network

Small world networks can be characterized by *average path length* between two nodes in the network and *cluster coefficient* defined as the probability that two neighbors of a node are neighbors themselves. A network is said to be small world if it has small average path length and large cluster coefficient. Studies ([12], [13]) on a spectrum of networks with small world characteristics show that searches can be efficiently conducted when the network has the

following properties: 1) each node in the network knows its local neighbors, called *short range contacts*; 2) each node knows a small number of randomly chosen distant nodes, called *long range contacts*, with probability proportional to $\frac{1}{d}$ where d is the distance. A search can be performed in $O(log^2 N)$ steps on such networks, where N is the number of nodes in a network [13]. The constant number of contacts (implying low maintenance cost) and small average path length serve as the motivation for constructing a small world overlay network in our approach.

B. Related Work

DHTs (e.g., CAN and Chord) organize peers and data objects in accordance with randomly hashed keys and can efficiently support simple key based exact match queries. Different from DHTs, skip graph [3] and P-Grid [2] organize peers and data objects in accordance with the values of the indexed attribute instead of randomly hashed keys. Therefore, these techniques can support complex queries, such as range query, in addition to exact match query. Nevertheless, these studies focus on applications involving data objects with a single attribute (i.e., one-dimensional data objects). A few recent works (e.g., [4], [7], [8], [10], [15]) propose techniques to support efficient search for data objects with a small number of attributes (around 10 attributes) in P2P systems. In contrast, SSW addresses searches for data objects with hundreds of attributes. The **high dimensionality** associated with the large number of attributes raises very challenging research issues, which are not addressed in the aforementioned studies. In order to overcome the challenges of high dimensionality, the design of SSW is radically different from what have been proposed so far in the literature.

The idea similar to semantic clustering has appeared in [5], [9], [17], [18]. While in [17], the super-peer in a cluster decides which peer can join its cluster, in [5] a centralized server is responsible for clustering documents and peers. Different from these studies, here we do not assume the existence of any super-peers or centralized server. [9] mentions the idea of clustering peers with similar interest together without discussing how to form clusters. [18] relies on periodic message exchanges among peers to keep track of other peers with similar documents, which incurs excessive communication overhead.

A couple of studies, e.g., [11], [24], following the principle of small world network, improve the search performance of *unstructured P2P systems* (where peers form into a random topology and maintain no index information about data objects stored at other peers). In addition to the neighbors in the network, a peer in [11] also maintains some short range contacts and long range contacts based on the similarity among the topics shared by peers. The short range contacts are the peers sharing similar topics within two hops and the long range contacts are the peers with

majority of their documents in one specific topic. Although the long range contacts conform to the aforementioned principle of small world network, the short range contacts do not conform to the principle of small world network since they are selected from peers within two hops rather than from the whole system. Therefore, to obtain satisfiable search results, queries still need to be propagated to a large portion of the network, incurring high search cost (in the order of N). On the contrary, our study improves the search cost to ($log^3(N)$). Reference [24] improves the performance of Freenet by caching search results at peers following the principle of small world network. This technique requires a warm up phase to populate the caches properly. In addition, the effectiveness of this proposal highly depends on the query access pattern. Different from this study, we design a small world overlay network that has efficient search performance regardless of query access patterns. [16] proposes to build a one-dimensional small world network, which can only support single key based exact match queries. Different from this study, our proposal can support content based search (i.e., high dimensional complex queries).

In addition to our study, some other works propose techniques to support content-based search in P2P systems by leveraging DHTs, e.g., [20], [23]. The basic idea proposed in [20] is to publish each keyword of a document on top of DHTs. Since the number of keywords associated with each document is large (in the order of hundreds or even thousands), publishing each of these individual keywords of a document on top of DHTs incurs excessive overhead. In addition, answering the searches involving multiple keywords requires join among the peers that index these keywords, which is an very expensive operation in P2P systems. Although [20] proposes various techniques to address the aforementioned issue associated with search, it does not address the excessive overhead incurred by publishing the keywords to the system.

pSearch [23] is the work most relevant to our study based on the authors' best knowledge, and thus we compare with it when necessary in this paper. While pSearch is intended for searching text documents only and can not be simply extended to support other types of digital data objects, our proposal is rather generic and can support searches for a variety of digital data objects in P2P systems. pSearch maps the latent semantic index (semantic vector for text document) to CAN overlay network by applying a dimension reduction technique, *rolling index*. Rolling index partitions a small portion (the lower dimensions) of the semantic vectors into p subvectors where each subvector consists of m dimensions with m as the dimensionality of CAN overlay. The partial semantic space corresponding to each subvector is then mapped into the key space of CAN. To process a semantic based search, p separate searches are performed on the CAN key space based on some heuristics. The most similar data object(s) in the result of these p searches is returned as the answer. Figure 1 shows an example of pSearch mapping a semantic vector

to a 2-dimensional CAN. The example shows that the first six elements of the semantic vector (which has totally 300 elements) are grouped into three 2-dimensional subvectors and mapped to three partial semantic spaces realized in one CAN.

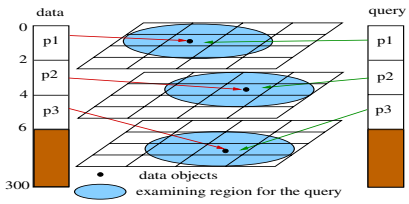


Fig. 1. **An illustrative example of pSearch.**

Although simple, rolling index incurs high index publishing overheads and search costs since each index publishing/search involves p operations with one corresponding to each subvector. In contrast, the dimension reduction technique we propose, ASL, requires only a single operation for index publishing and search. In addition, ASL considers all semantic information during a search, instead of only m dimensions as in rolling index, thereby no complex heuristics are required to direct the search (which are required by pSearch).

III. Semantic Small World

In this section, we present the overview on the design of semantic small world (SSW) and provide the technical details on constructing a k-dimensional SSW.

A. Overview

SSW plays two important roles: 1) an overlay network that provides connectivity among peers; and 2) a distributed index that facilitates efficient search for data objects. In SSW, the peers and data objects are organized in accordance with the k-dimensional semantic space populated by the k-dimensional SVs capturing the semantics of data objects. As such, in addition to navigation, a peer in the overlay network is responsible for management of data objects (or the location information of data objects stored at other peers - referred to as *foreign indexes*) corresponding to a *semantic subspace*. Foreign indexes, similar to the leaf node pointers of typical tree-based index structures, provide location information regarding to where data objects are physically stored[2]. To enhance the robustness of SSW, instead of assigning each individual peer to a semantic subspace,

[2]Due to the potential high cost of redistributing a large number of data objects within the overlay network, we choose to have a newly joined peer to publish the location information of its locally stored data objects to the other peers managing the subspaces corresponding to semantics of those data objects.

several peers form into a *peer cluster* to share the responsibility of managing a semantic subspace. These peer clusters then self-organize into a small world network.

Corresponding to a k-dimensional semantic space, a k-dimensional SSW can be formed as follows. Each peer in this k-dimensional SSW maintains s *short range contacts* and l *long range contacts*. The short range contacts are selected to ensure the connectivity among peers so that a search message issued from any peer can reach any other peer in SSW. For a k-dimensional semantic space, the short range contacts of a peer P_1 can be intuitively set to the peers in the neighboring clusters next to P_1 in both directions of the k dimensions ($s = 2k$). Note that it is possible to use a s smaller than $2k$ as long as the short range contacts can ensure the connectivity among peers. On the other hand, the long range contacts aim at providing short cuts to reach other peers quickly. Via short range contacts and long range contacts, navigation in the network can be performed efficiently.

There are several critical issues that need to be addressed in the design of SSW: 1) *semantic clustering* - how to organize peers with similar semantics into peer clusters; 2) *dimension reduction* - how to handle the maintenance issue of the overlay network given the high dimensionality of the corresponding semantic space; 3) *search* - how to conduct searches efficiently in SSW. In the following, we briefly discuss our strategy for semantic clustering while introducing the tasks of constructing a k-dimensional SSW. For details, please refer to [14]. We leave the discussions on dimension reduction and search to the following two sections.

B. Constructing a k-Dimensional Semantic Small World

Constructing a k-dimensional SSW depicted above involves two major tasks: 1) organizing peers with similar semantics into peer clusters; 2) constructing an overlay network across the peer clusters to form a semantic small world network.

Semantic Clustering. In order for peers with similar semantics to form into peer clusters, we need to address the following two sub-issues: 1) *peer placement* - where in the semantic space a peer should be located; 2) *cluster formation* - what is the strategy for forming clusters.

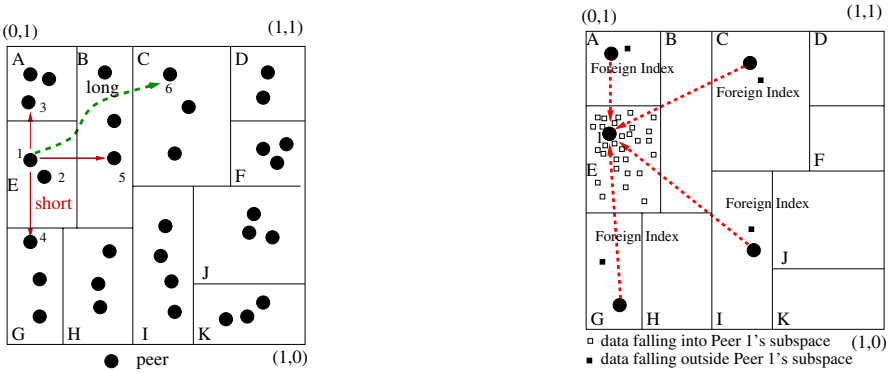
Peer Placement. A new peer executes a clustering algorithm (e.g., k-means) on its local data objects and chooses the centroid of the largest data group obtained by clustering as its position in the data space. This position is called *semantic position* of this peer. In contrast to random peer positioning as adopted in most overlay structures, this scheme places peers in the data space adaptively according to data distribution, and takes advantage of the homogeneity existed among data objects hosted at a peer by reducing the index publishing cost.

Cluster Formation. A cluster of peers share the responsibility of managing a data subspace to make the system adaptive to dynamic membership changes and achieve fair load distribution. A new peer joins a peer cluster based on its semantic position obtained as above. If the number of peers in a peer cluster exceeds a predefined threshold value M, the peer cluster and the corresponding semantic subspace is partitioned into two in a way adaptive to data distribution (the details on space partitioning are explained in [14]).

Overlay Network Construction. To construct the overlay, each peer maintains a set of short range contacts, each of which points to a peer in a neighboring peer cluster, and a certain number of long range contacts. A long range contact is obtained as the peer responsible for a point randomly drawn from the semantic space following a distribution, $\frac{1}{d^k}$ where k is the dimensionality of the semantic space and d is the semantic distance. These extra long range contacts reduce the network diameter and transform the network into a small world with poly-logarithmic search cost (Theorem 1, which is an extension of the theorem in [13] to k-dimensional space). In addition, there are no rigid rules on which specific distant clusters should be pointed to by long range contacts. This flexibility of long range contacts selection can be utilized easily to make SSW adapt to locality of interest (to be detailed in Section V-B).

Theorem 1 Given a k-dimensional semantic small world network of N peers, with maximum cluster size M and number of long range contacts l, the average search path length for navigation across clusters is $O(\frac{log^2(\sqrt{k}(\frac{2N}{M})^{1/k})}{l})$.

Proof: Omitted due to space constraints.



(a) overlay structure (b) foreign indexes

Fig. 2. **An illustrative example for SSW.**

Figure 2 shows an example of SSW ($k = 2$). As shown in the figure, the search space is partitioned into 11 clusters after a series of peer joins and leaves. Figure 2(a) shows the overlay structure. Peer 1 in cluster E maintains three short range contacts to neighboring peer clusters A,

B and G and one long range contact to a distant peer cluster C. The contacts of other peers are not shown here for clarity of presentation. Figure 2(b) illustrates the concept of foreign indexes. The dark circles denote the semantic positions of peers in the semantic space. The small rectangles represent the data objects stored in Peer 1. Most of them (the white rectangles) are located in the subspace of Peer 1, but some of them (the dark rectangles) are mapped to other subspaces. Thus, the location information of those data objects are stored as foreign indexes at the peers in charge of those subspaces.

IV. Dimension Reduction

The k-dimensional SSW obtained by simply assigning short range contacts in all dimensions of the corresponding semantic space (as described in above section) is feasible when k (the dimensionality) is small. When k is large, the maintenance of such an overlay becomes costly and non-trivial due to the decentralized and highly dynamic nature of P2P systems. Unfortunately, as mentioned in the beginning of this paper, the number of attributes capturing the semantics of data objects is normally very large, and thus the dimensionality of the semantic space (k) is high. For instance, the dimensionality of latent semantic index is around 50-300.

One idea to address this issue is to construct an overlay network of low dimensionality to support the function of semantic based search in the high dimensional semantic space. This idea can be realized by linearizing the peer clusters from the high dimensional semantic space to a one-dimensional naming space, i.e., assigning a unique *ClusterID* to each peer cluster, and then constructing a one-dimensional SSW (SSW-1D) over the linearized naming space. SSW-1D is constructed as a double linked list consisting of peer clusters connected via two short range contacts of each peer. In addition to this linear network structure that provides basic connectivity, long range contacts provide short cuts to other peer clusters, which facilitate efficient navigation. While the original semantic space has been partitioned and then linearized, the peer clusters in SSW-1D are still corresponding to their original semantic subspace of high dimensionality.

In order to facilitate efficient navigation in SSW-1D based on the high dimensional semantic information (i.e., SVs) corresponding to the original semantic space, the following two issues should be addressed carefully:

- *Naming Encoding.* The mapping from the high dimensional semantic space to a one dimensional naming space should preserve data locality, i.e., clusters located nearby in the semantic space should be assigned ClusterIDs with similar values.
- *Naming Embedding.* The aforementioned mapping should be recorded or embedded in the overlay network so that an arbitrary peer can determine the ClusterID of the peer cluster

responsible for a given data object (this is necessary to process a search, which will become clear in the following section).

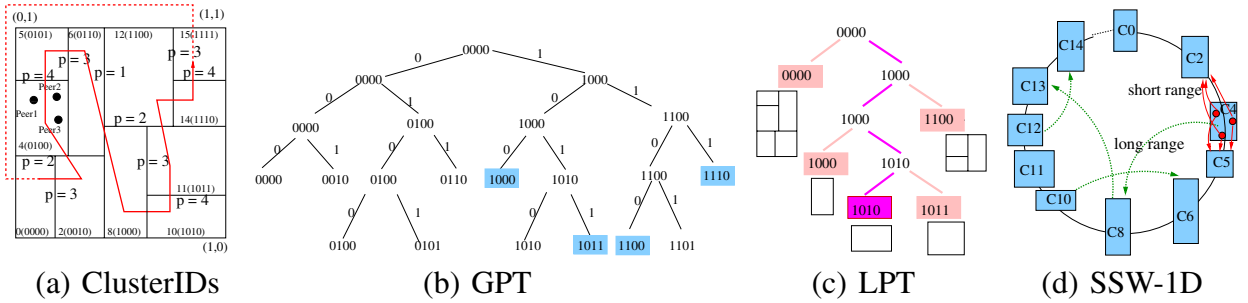
The well known space filling curves such as Hilbert curve, Z-curve, etc., can only be employed to map a regularly coordinated high-dimensional space to a low-dimensional space. In our case, the high-dimensional semantic space is adaptively (irregularly) partitioned according to data distribution, and thus these techniques are not applicable here.

In this study, we propose a technique, called *adaptive space linearization* (ASL), which linearizes the peer clusters in the high dimensional semantic space into a one-dimensional naming space during the process of cluster split in SSW construction. In the following, we explain how ASL addresses the aforementioned two issues, i.e., naming encoding and naming embedding.

A. Naming Encoding

We observe that the partition of the data space can be represented by a 'virtual' binary tree and encoding this binary tree properly can lead to a location-preserving naming encoding scheme. Based on this observation, we propose the following naming encoding scheme. We use a bit string with fixed length to represent ClusterIDs. We call the virtual binary tree depicting the partition of the data space as *global partition tree (GPT)*. Note that GPT is not maintained anywhere in the network. We simply use it for the explanation of the partitioned space and the naming encoding scheme. The root node of the GPT represents the initial data space. Each partition event generates two subtrees where the left (right) subtree corresponds to the subspace with smaller (larger) coordinate along the partition dimension. The subspace corresponding to a leaf node in the GPT is called *leaf subspace*, which peers are residing in (and responsible for managing). We associate each bit in the ClusterID (starting from the most significant one) with a level in the GPT (starting from the root level). Therefore, level i at the GPT corresponds to the i^{th} most significant bit in ClusterID. These bits are set to 0 by the left subtrees and "1" by the right subtrees. Any tree node obtains a *tree label* by concatenating the bits along the path from the root to itself and padding the remaining bits in its ClusterID with 0 (note that the combination of the depth and the tree label uniquely identifies a node in the GPT). The tree label of a leaf node is the ClusterID of the peer cluster in charge of the corresponding leaf subspace.

Figure 3(a) shows the same snapshot of the system as Figure 2 where the whole semantic space is partitioned into 11 clusters with the ClusterIDs indicated in the figure. Figure 3(b) depicts the corresponding GPT (with tree labels). We illustrate the process in a 2-dimensional space with a naming space of 4-bit long. In this example, the semantic space is first partitioned along the vertical line as indicated by "p = 1" in the figure. Peers at the left side and right side



(a) ClusterIDs (b) GPT (c) LPT (d) SSW-1D

Fig. 3. **An illustrative example of ASL.**

of this line obtain ID ”0000” and ”1000”, respectively. The left side is then partitioned along the horizontal line as indicated by ”p = 2”. At this point, peers at the lower left side and top left side obtain ID ”0000” and ”0100”, respectively. The data space is eventually partitioned as shown in the figure. We use the solid line to illustrate the order of the assigned ClusterIDs. The dashed portion of the line, naturally created since SSW-1D is a double linked list, indicates that a search can be performed bi-directionally.

B. Naming Embedding

We now proceed to the second issue - how the mapping from the high dimensional semantic space to the one dimensional naming space is embedded in the overlay network so that a peer can determine the ClusterID of the peer cluster responsible for a given data object. A simple scheme is to replicate the complete partition history, i.e., the entire GPT tree structure, at each peer. However, this scheme is not scalable since any subspace partition needs to be propagated to the whole network.

Here we propose to only store the history of the partitions (in terms of the partition dimension and partition point) that a peer has been involved with at this peer. Note that this partial partition history corresponds to the path in the GPT starting from the root to the leaf node corresponding to this peer’s subspace. Thus, we call this partial partition history as *local partition tree (LPT)*. In the LPT, each node (except the leaf node) has two children representing the two subspaces generated in a space partitioning event. Between the two subspaces, the one that the peer does not reside in may undergo further partitioning that is not known to the peer, and thus it is called *old neighbor subspace*. The readers should not confuse the old neighbor subspaces with short range contacts or long range contacts. While a peer has maintained the history of the data space partition events generating these subspaces in its LPT, it doesn’t necessarily have contacts in each of these old neighbor subspaces.

Figure 3(c) illustrates the LPT for a peer residing in Cluster 10 (marked by dark rectangle). Cluster 10 has been involved in four partitioning events, generating four old neighbor subspaces (marked by light rectangles). In this figure, we also show how the subspace corresponding to a leaf node in the LPT is partitioned further. From this figure we can see the two old neighbor subspaces (i.e., the tree node with label 0000 and 1100) are partitioned further and consist of a collection of smaller subspaces, respectively.

Figure 3(d) illustrates SSW-1D built upon the ClusterIDs. A peer in Cluster 4 maintains short range contacts to the neighboring clusters 2 and 5. It also maintains a long range contact to a distant cluster 8. A peer in Cluster 8, 10 and 12 maintains a long range contact to Cluster 13, 6 and 14, respectively. The contacts of other peers are not shown here for readability.

V. SEARCH

In this section, we explain the search process in SSW-1D. For the details of overlay maintenance, please refer to [14]. For simplicity, we refer SSW-1D as SSW in the rest of this paper where the context is clear. We present the algorithms for two common types of semantic bases searches, i.e., *approximate point query* (denoted as *point* query*) and *range query*. Point* query, specified by a query semantic vector or query point q, returns the data objects matching q if there is such a data object in the system (similar to a regular point query); otherwise, it returns a data object similar to q (different from a regular point query). Range query, specified by a query point q and a query radius r, returns the data objects with dissimilarity to the query point q smaller than the specified range r. Note that a point query can be treated as a special case of range query with $r = 0$.

To process these queries, we need to address the following two issues:

- *Search space resolution (SSR).* To process a query, a peer first needs to determine which portions of the overlay (peers, data subspaces) host the requested data (or the index information for the data). Since the overlay (index) structure of SSW is constructed over the linearized naming space, and each peer only has partial knowledge of the mapping from the high dimensional data space to the one dimensional overlay, SSR is a non-trivial issue.
- *multi-destination query propagation (MDQP).* By the nature of range queries, it is easy to observe that the searched data may be spread out in multiple semantic subspaces. Therefore, for range query, we need to address how to efficiently propagate a query to multiple peer clusters that may have the qualified data. The lack of complete neighborhood information in all data dimensions makes this issue very difficult to solve.

The solution to SSR is presented in Section V-A, followed by the algorithms for point* and range query in Section V-B and V-C, respectively. The solution to MDQP is also discussed in Section V-C.

A. Search Space Resolution

Algorithm 1 Algorithm for SSR.

Peer *i* issue ssr(x, q, r, a, h): $i.ssr(x, q, r, a, h)$ **(x, set to the root node initially, is the current tree node to be examined in i's LPT. q and r are the center and range of the search space. a and h, set to 0 initially, indicate the value and the number of bits resolved in the PCN.)**

```
1: if x is the leaf node in i.LPT then
2:    Return a_h.
3: else
4:    h = h + 1;
5:    if mindist(q, x.left) ≤ r then
6:       Set the i^th most significant bit of a to 0.
7:       i.ssr(x.left, q, r, a, h).
8:    end if
9:    if mindist(q, x.right) ≤ r then
10:      Set the i^th most significant bit of a to 1.
11:      i.ssr(x.right, q, r, a, h).
12:   end if
13: end if
```

Since SSW is constructed over the one dimensional naming space captured by ClusterIDs, SSR basically is to figure out the ClusterIDs of the peer clusters intersecting with the search space (or query region) associated with a given query. We propose a localized algorithm for SSR based on the concept of LPT (local partition tree) as follows. A peer examines its LPT starting from the root node and decides whether a subtree in the LPT should be pruned or not by examining whether the corresponding subspace intersects with the search space or not. To determine whether a subspace intersects with the search space, the peer first computes the minimum distance, $mindist$, from the query point to the subspace. If the computed $mindist$ is greater than r (the range of the search space), the subspace does not intersect with the search space, and thus the corresponding subtree is pruned. Otherwise, the corresponding subtree is examined further. Both subtrees may be examined if both corresponding subspaces intersect with the query region. This process continues till the leaf nodes in the LPT, called as *terminal nodes*, are reached. We assign a *pseudo-cluster-name (PCN)* to the corresponding subspace of a terminal node. PCN is a tuple of $\langle a, h \rangle$, denoted as a_h, where a and h are set to the tree label and depth of the corresponding terminal node in the LPT. Here a and h correspond to the value of the PCN and the number of bits resolved in the PCN, respectively. If a PCN's value is the same as the ClusterID of current peer, this PCN is completely resolved. Otherwise, the

PCN may be refined further by invoking SSR at the peers managing the old neighbor subspace corresponding to the terminal node. Algorithm 1 shows the pseudo code for SSR at a peer.

Figure 4(a) illustrates an example for SSR. The large circle ($q2$) intersecting with Clusters 8, 11, 12 and 14 depicts the search space (also marked on the GPT in Figure 4(b)). A peer in Cluster 10 resolves the search space and obtains the corresponding PCNs as 8_3, 11_4 and 12_2. The search space corresponding to PCN 12_2 is later resolved as 12_4 and 14_3 by a peer residing in the old neighbor subspace corresponding to the tree node with label 1100 in Cluster 10's LPT.

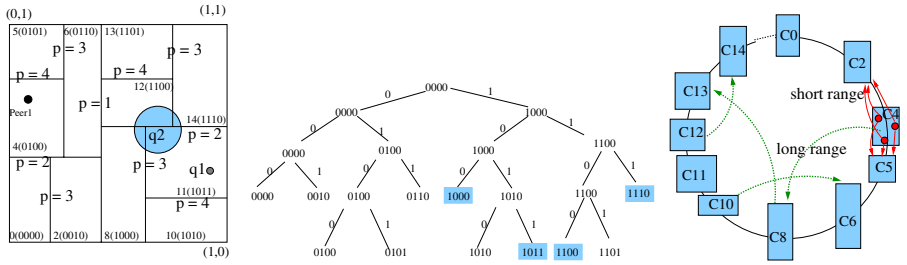


Fig. 4. **An illustrative example of SSR.**

B. Point Query*

The algorithm to process a point* query consists of two stages: *navigation-across-clusters*, which navigates to the *destination peer cluster* covering the query point, and *flooding-search-within-cluster*, which floods the query within the destination peer cluster for query results. When a message is received, a peer first checks whether q falls within the range of its peer cluster. If that is not the case, SSR is invoked to obtain the PCN (pseudo-cluster-name) for q based on the partition history (LPT) stored at this peer. The query message is then forwarded to the contact with the shortest distance to the PCN in the one-dimensional naming space. The above process is repeated until the cluster whose semantic subspace covering q is reached. At this point, flooding-search-within-cluster is invoked by flooding the message to other peers in the same peer cluster. Then the data object with the highest similarity to the query is returned as the result.

Here, we show an example to illustrate point* query processing in SSW-1D. Let's go back to Figure 4(a). Assume Peer 1 in Cluster 4 issues a point* query $q1$ [0.9,0.3] indicated by the small grey circle in Cluster 11. Peer 1 first checks its own cluster range. Since [0.9,0.3] is not within the subspace of its peer cluster, Peer 1 invokes SSR and estimates the PCN for the query as 8_1. Peer 1 then forwards the message to Cluster 8, which invokes SSR and refines the PCN as 10_3.

The message is then forwarded to Cluster 10, which refines the PCN further as 11_4. The query is finally forwarded to a peer in Cluster 11, which finds $q1$ is within its own cluster range and thus floods the message within its peer cluster to obtain the query results.

Despite the need for SSR, navigation in SSW-1D is efficient. We group the routing hop(s) to resolve one bit of PCN as a *PCN resolving phase*. A query may need to go through multiple PCN resolving phases, where each phase brings the query message half-way closer to the target. The following theorem obtains the search path length for SSW-1D (different from Theorem 1 where a peer has short range contacts along all the dimensions, in SSW-1D, a peer has only 2 short range contacts in a k-dimensional space ($k >> 1$) and thus it only has a partial knowledge of its neighborhood.).

Theorem 2 For a k dimensional space, given a SSW-1D of N peers, with maximum cluster size M and number of long range contacts l, the average search path length for navigation across clusters is $O(\frac{log^3(2N/M)}{l})$.

Proof: The average number of peer clusters in the system is $\frac{2N}{M}$, which implies total $log\frac{2N}{M}$ bits in PCN need to be resolved, requiring $log\frac{2N}{M}$ PCN resolving phases. Starting from the most significant bit, each phase resolves one bit in PCN, thereby reducing the distance to the target cluster by half. S denotes the initial distance to the target cluster, i.e., $S = \frac{2N}{M}$. According to Theorem 1 (dimensionality k is 1 here), the path lengths for these $log\frac{2N}{M}$ phases are $\frac{log^2 S}{l}$, $\frac{log^2(S/2)}{l}$, $\frac{log^2(S/4)}{l}$, ..., $\frac{log^2(S/2^{logS-1})}{l}$, respectively. The search path length in SSW-1D is the summation of all these path lengths incurred by different phases, which is $O(\frac{log^3 S}{l})$. Hence, the above theorem is proved. □

During the query process, SSW adapts to locality of users interests as follows. Each peer maintains a *query-hit* list, which consists of the peers who have query hits (provide query results) in the past X queries issued by this peer. For every X queries, a node replaces the long range contact having the lowest hit rate with the entry having the highest hit rate in the query-hit list with probability of $\frac{d_i}{d_i+d_j}$, where d_i and d_j represent the naming distance of the old long range contact and the candidate long range contact to current peer, respectively. We prove this update strategy maintains the properties of small world networks, i.e., the distribution of long range contacts is still proportional to $\frac{1}{d}$ where d is the distance of a long range contact.

Theorem 3 By replacing a long range contact at distance d_i by another peer at distance d_j with probability $\frac{d_i}{d_i+d_j}$, the distribution of long range contacts is proportional to $\frac{1}{d}$.

Proof: Basically if we can prove at steady state the probability that a peer at distance d becomes a long range contact, denoted by p_{in}, equals to the probability that a long range contact at distance d is replaced by other peers, denoted by p_{out}, we prove the above theorem.

p_i denotes the probability that Peer i is the long range contact of current peer, which equals to $\frac{C}{d_i}$ where C is the normalization constant that brings the total probability to 1. $p_{i\rightarrow j}$ donates the probability that a long range contact i is replaced by another peer j, which equals to $\frac{d_i}{d_i+d_j}$. We can then obtain $p_{out} = \sum_{j=1}^{N} p_i \cdot p_{i\rightarrow j} = \sum_{j=1}^{N} \frac{C}{d_i} \cdot \frac{d_i}{d_i+d_j} = \sum_{j=1}^{N} \frac{C}{d_i+d_j}$, and $p_{in} = \sum_{j=1}^{N} p_j \cdot p_{j\rightarrow i} = \sum_{j=1}^{N} \frac{C}{d_j} \cdot \frac{d_j}{d_i+d_j} = \sum_{j=1}^{N} \frac{C}{d_i+d_j}$. Since $p_{in} = p_{out}$, this proves the above theorem. □

C. Range query

The query region (search space) of a range query may intersect with multiple subspaces, called *candidate subspaces*, managed by different peer clusters (called as *candidate clusters*, accordingly). An idea to process a range query is to greedily route the query message, based on the shortest distance to the specified query region, towards a peer located in a candidate subspace. Once the query message reaches such a subspace, in addition to being propagated to other peers located in the same subspace, the query message is propagated further (as needed) to other candidate subspaces.

In the following, we first describe a general algorithm for processing range query and then discuss our strategy to propagate the query message to multiple candidate subspaces, i.e., the issue of MDQP (multi-destination query propagation).

1) Range Query Algorithm: Based on the above idea, we develop an algorithm for processing range query. Given that a peer issues or receives a range query, if its subspace intersects with the specified query region, the peer propagates the query message to all other members in its peer cluster. Meanwhile, regardless of the existence of overlapping between a peer's subspace and the query region, this peer invokes SSR and determines whether there exist (other) subspaces intersecting with (the remaining part of) the query region and obtains their PCNs. Next, the peer forwards the query message greedily towards the candidate subspace(s) using our MDQP algorithm (to be discussed shortly).

2) Multi-Destination Query Propagation (MDQP): If a peer maintains the neighborhood information of all data dimensions as in CAN, MDQP can be solved easily by propagating the message to the neighbors recursively. However, as discussed in Section IV, to address the issue of high dimensionality, the peer clusters form into SSW-1D and a peer does not maintain the complete neighborhood information of all data dimensions.

One possible strategy for MDQP is to propagate the query message to the candidate subspaces sequentially in certain order. Therefore, the single-destination routing protocol (as described for Point* query earlier in Section V-B) can be directly applied here to solve the problem of multi-destination query propagation. However, this sequential forwarding strategy incurs

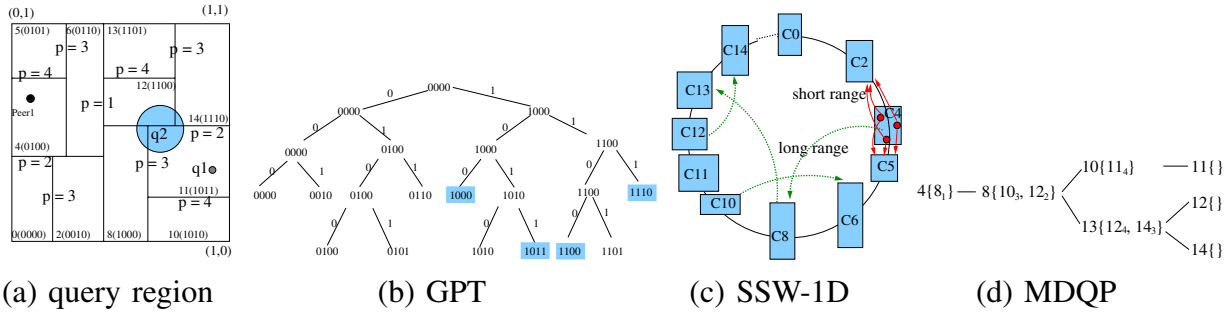
long query latency. In the following, we present a MDQP algorithm to propagate a query message to multiple candidate subspaces in parallel. Since our MDQP algorithm invokes multiple propagation paths simultaneously, an important issue is to avoid redundant query message propagation and processing. Thus, a peer, upon receiving a query message, needs to know which portions of the query region have been processed or are being processed by other peers so that it will not process these regions again. We observe that a PCN (e.g., a_h) corresponds to a GPT subtree rooted at the tree node with the tree label as a and depth as h. This information can be utilized to propagate the query message systematically from the virtual GPT tree downwards along different paths. In other words, if a peer obtains multiple PCNs through SSR, this peer forks multiple query messages, each of which is attached with a PCN and forwarded towards the subspace(s) corresponding to the PCN. A receiver of the query message only processes the portion of the query region that intersects with the subspace indicated by the attached PCN. This process is repeated recursively until there is no more subspace to be examined. Algorithm 2 shows the pseudo codes for range query processing.

Algorithm 2 Algorithm for range query.

Peer *i* issue range(q, r): $i.range(q, r, a_h)$ **(a_h is the currently estimated PCN for q where a and h are initialized to 0, respectively.)**

1: **if** $range(q, r)$ intersects with i's subspace **then**
2: Invoke flooding-search-within-cluster.
3: **end if**
4: Invoke SSR and record in U the PCNs (indicated by $a'_{h'}$) of the subspaces intersecting with the query region corresponding to PCN a_h.
5: **for all** $a'_{h'} \in U$ **do**
6: Fork a query message $range(q, r, a'_{h'})$ and forwards it towards a'.
7: **end for**

Figure 5 illustrates an example for range query processing. Assume Peer 1 in Cluster 4 issues a range query depicted by the large circle ($q2$) in Figure 5(a). For presentation clarity, here we redisplay the figures for the corresponding GPT and SSW-1D from Figure 3(b) and (c). Peer 1 first invokes SSR and obtains the PCN of the only candidate cluster/subspace as 8_1. As a result, the query is forwarded to Cluster 8. The peer receiving the message in Cluster 8 further obtains the PCNs for the query region via SSR as 10_3 and 12_2. Therefore, this peer forks two query messages and forwards them in accordance with the two PCNs, respectively. The message corresponding to 10_3 reaches Cluster 10 and then is further forwarded to Cluster 11. The message corresponding to 12_2 reaches Cluster 13 first and then is forwarded to Cluster 12 and 14 simultaneously.



(a) query region (b) GPT (c) SSW-1D (d) MDQP

Fig. 5. **An illustrative example of range query.**

VI. Performance Evaluation

We move on to evaluate SSW's benefits through extensive simulations. We compare SSW with pSearch, the state-of-the-art on semantic based search in P2P systems. Based on [23], pSearch takes 4 groups of subvectors, each with $2.3lnN$ dimensions (i.e., $p = 4$, m = $2.3lnN$). In addition, to compare the effectiveness of our proposed dimension reduction method ASL with rolling index used in pSearch, we also implement the rolling index on top of m-dimensional small world network, called as *small world rolling index (SWRI)*. We conduct point* query as well as range query in the simulation. We extend pSearch/SWRI to support point* query and range query as follows. To process a point* query, the query vector is partitioned into p subvectors, and the query is routed to p subspaces covering each of these p subvectors. The most similar data object is returned from each of the p subspaces and the most similar one among these p data objects is returned as the result. To process a range query, the query message is forwarded to a peer whose subspace intersects with the query region in the first of the p subvectors, and then propagated to the peers whose subspaces intersect with other portion of the query region in this subvector. These peers then return the data objects whose semantic vectors are within the query region in the complete semantic space as the results. In the following, we explain the simulation setup before we present the simulation results.

A. Simulation Setup

A random mixture of operations including peer join, peer leave and search (based on certain ratios) are injected into the network. During each run of the simulation, we issue $10{\cdot}N$ random queries into the system. The proportion of peer join to peer leave operations is kept the same to maintain the network at approximately the same size. The simulation parameters, their values and the defaults (unless otherwise stated) are given in Table I. Most of these parameters are self-explanatory. More details for some of the system parameters, synthetic data set, real data set, and query workload are given below.

TABLE I
PARAMETERS USED IN THE SIMULATIONS

	Descriptions	**Values, default**
N	Number of peers in the network	256 - 16K, $\underline{1K}$
l	Number of long range contacts	$\underline{4}$
M	Size of peer clusters	1 - 1024, $\underline{8}$
n	Number of data objects per peer	1 - 100, $\underline{100}$
α_{d1}	Skewness of Dataseed-Zipf	0 - 1.0, $\underline{0}$
α_{d2}	Skewness of Data-Zipf	0 - 1.0, $\underline{0}$
γ	Percentage of peer join and peer leave operations	0% - 50%, $\underline{20\%}$
α_{q1}	Skewness of Queryseed-Zipf	0 - 1.0, $\underline{0}$
α_{q2}	Skewness of Query-Zipf	0 - 1.0, $\underline{0}$
W	SV dimension weight assignment	uniform, $\underline{linear}$, zipf
k	Dimensionality of semantic space	10 - 100, $\underline{100}$
R	Maximum query radius	0 - 1, $\underline{0.5}$

Synthetic Data Set. The data set is defined by the dimensionality of SV (and the semantic space), the weight of each dimension in SV, the number of data objects per peer (n), and the data distribution in the semantic space. The default setting for the dimensionality of SV (k) is 100. We use three different weight assignments for SV, namely, uniform, linear, and Zipf dimension weight assignments. In the uniform dimension weight assignment, each dimension has the same weight. In the linear dimension weight assignment, Dimension i has weight $\frac{k+1-i}{k}$ ($1 \leq i \leq 100$). Zipf-distribution is captured by the distribution function $\frac{1}{i^{\alpha}}$ where α is the skewness parameter. In our Zipf dimension weight assignment, α is set to 1, i.e., Dimension i has weight $\frac{1}{i}$. If unspecified, linear dimension weight assignment is the default setting.

Data distribution in the semantic space is determined by two factors: 1) semantic distribution of data objects among different peers; and 2) semantic distribution of data objects at a single peer. The former controls the data hot spots in the system and the latter controls the semantic similarity between data objects at a single peer, namely *semantic closeness*. To model both factors, we associate a Zipf-distribution each, *Dataseed-Zipf* for the former and *Data-Zipf* for the latter, with their skewness parameters as α_{d1} and α_{d2}, respectively. We first draw a seed for each peer following the Zipf distribution controlled by α_{d1}. This serves as the centroid around which the actual data objects for that peer are composed following α_{d2}.

Real Data Set. We derive the latent semantic indexes for 527000 documents in TREC version 4 and version 5 [1]. We distribute these documents to peers in the following two ways: 1) random data distribution, where documents are randomly distributed among peers, and 2) clustered data distribution, where documents are first assigned into N clusters using k mean algorithm and each cluster is then randomly assigned to a peer.

Workload Parameters. A point* query is specified by the query point, and a range query is specified by the query radius and the query point. The query radii for range query are randomly drawn from the range $[0, R]$ where R indicates the maximum query radius. Similar to data parameters, we consider two factors in generating query points: distribution of query points emanating across the peers in the system and the skewness of the query points emanating from a single peer. The former controls query hot spots (i.e. more users are interested in a few data items) in the system and the latter controls the locality of interest for a single peer, namely *query locality* (i.e. a user is more interested in one part of the semantic space). We use two Zipf distributions with parameters α_{q1} (for *Queryseed-Zipf*) and α_{q2} (for *Query-Zipf*) to control the skewness, i.e., α_{q2} captures the skewness of the queries around the centroid generated by α_{q1}.

While the main focus of this paper is to improve search performance with minimum overhead, we also try to explore the strengths and weaknesses of SSW in other aspects, such as fault tolerance and load balance. In this paper, we use the following metrics for our evaluations.

- **Search path length** is the average number of logical hops traversed by search messages to reach the destination.
- **Search cost** is the average number of messages incurred per search.
- **Maintenance cost** is the number of messages incurred per membership change, consisting of **overlay maintenance cost** and **foreign index publishing cost**. Since the size of different messages (join, query, index publishing, cluster split, cluster merge) is more or less the same (dominated by the size of one SV (400 bytes)), we focus on the number of messages in the paper for clarity.
- **Search failure ratio** is the percentage of unsuccessful searches that fail to locate existing data objects in the system.
- **Index load** is the number of foreign index entries maintained at a peer.
- **Routing load** is the number of search messages that a peer processed.
- **Result quality** is to measure the quality of the returned data object for a point* query. To calculate the result quality, we first calculate the normalized dissimilarity (Euclidean distance)[3] between the query and the result returned by SSW (or pSearch/SWRI), denoted as d_{real}, and the normalized dissimilarity between the query and the data object most similar to the query in the system, denoted as d_{ideal}. Then we use $1-(d_{real}-d_{ideal})$ to represent the result quality. When the difference between d_{ideal} and d_{real} is very small, the result quality

[3]To perform the normalization, we divide the Euclidean distance between two vectors by $\sqrt{k}$, which is the maximum Euclidean distance between two vectors in the semantic space.

is high.

B. Simulation Results

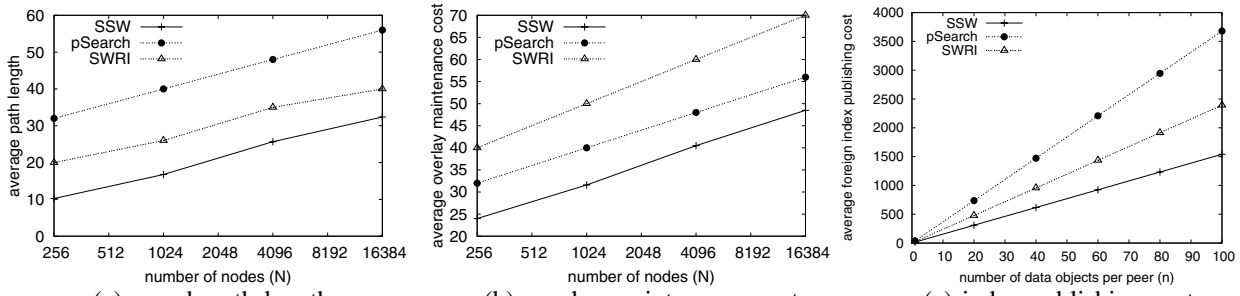
We have conducted extensive experiments under both synthetic data set and real data set to demonstrate SSW's strength on various aspects. In the following, we first present the results that demonstrate SSW's nice properties as a generic overlay structure, followed by the results demonstrating SSW's efficiency in support of different types of queries. We present the results under different data sets when they display different trends. Otherwise, we present the results under one specific data set for presentation brevity and the interest of readers.

1) Overlay Properties: In this following, we demonstrate SSW's scalability, adaptivity to data distribution and query locality, resilience to peer failures, and load balancing property.

Scalability. We vary the number of peers from 2^8 to 2^{14} to evaluate the search efficiency and maintenance cost of SSW. According to our preliminary simulation results ([14]), we find SSW with 4 long range contacts has reasonable trade-off between search efficiency and maintenance overhead for most of the γ settings, and we use this value in the experiments. Since pSearch does not use any clustering, we disable the clustering feature of SSW (i.e., cluster size M is set to 1) in this set of experiments. We have demonstrated that SSW can perform even better with appropriate cluster sizes (as detailed in [14]).

Figure 6(a) shows the average path length under uniformly distributed synthetic data set ($\alpha_{d1} = \alpha_{d2} = 0$). The results under other data sets are almost the same and are omitted. Since the size of peer clusters is set to 1 in this experiment, there is no flooding within a cluster and the average search path length for SSW represents the search cost as well. The search path length for SSW increases slowly with the size of network, confirming search path length bound in Theorem 2. In addition, the constant hidden in the big-O notation is smaller than 1 as shown in the figure. The plot of pSearch's path length has a slope close to SSW (with 4 long range contacts) but has a much higher offset. In fact, the search path length of SSW is about 40% shorter than that of pSearch at network size 16K. The search path length of SWRI is between pSearch and SSW. This confirms the effectiveness of ASL vs. rolling index in terms of search path length.

Overlay maintenance cost is proportional to the number of states maintained at each peer, which are 20, 24 (20 short range contacts and 4 long range contacts), 6 (2 short range contacts and 4 long range contacts) for pSearch, SWRI and SSW respectively. Figure 6(b) shows the overlay maintenance cost for the same experiments as Figure 6(a). These two figures confirm our expectation that compared to pSearch, SSW can achieve much better search performance



(a) search path length (b) overlay maintenance cost (c) index publishing cost

Fig. 6. **Comparing the scalability of the schemes.**

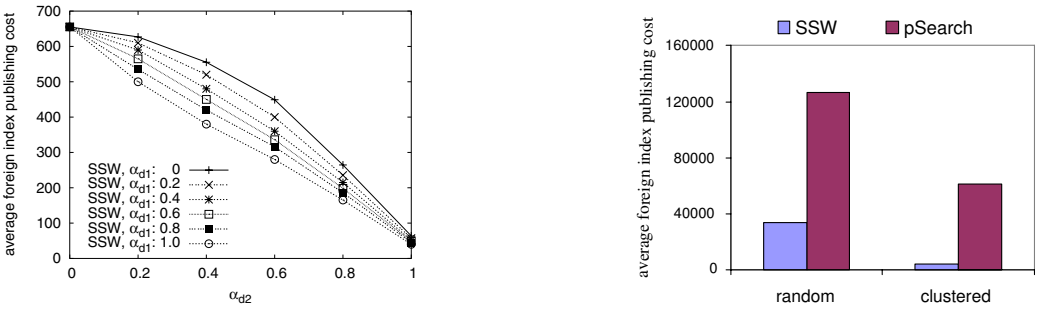
with smaller overlay maintenance overhead.

The other maintenance cost to consider is the overhead of publishing foreign index upon peer joins (apart from the cost shown in Figure 6(b)). This cost is proportional to the number of data objects that need to be published, and the corresponding relationship is shown in Figure 6(c). Due to the fact that pSearch/SWRI have to publish a data object multiple times, the index publishing costs for pSearch/SWRI are much higher than that for SSW. In addition, the index publishing cost for SWRI is lower than that for pSearch. Note that these results for SSW are conservative since $\alpha_{d2} = 0$ (uniform data distribution) and the overhead is likely to be lower with any skewness (as will be shown later).

This set of experiments confirms the scalability of SSW. It also confirms our expectations that ASL is a better dimension reduction method than rolling index in terms of various aspects, including search cost and index publishing cost (later, we will show ASL is better than rolling index in terms of the result quality for point* query as well). In the remaining experiments, we only compare with pSearch for presentation clarity.

In above experiments, the size of the peer clusters (M) has been set at 1. When M is larger, cluster splits or merges occur less frequently, resulting in lower overlay maintenance costs. Further, the total number of peer clusters in the system decreases with larger cluster sizes, thereby reducing the cost of navigation-across-clusters. The down-side of large sized clusters is the higher search cost within a cluster (due to flooding). We have performed a series of experiments by varying the cluster size from 1 to N, which show within a spectrum of cluster sizes between 2 and 16, SSW does better than the cluster size of 1 in terms of all maintenance cost, path length, and search cost (please see [14] for the details). Thus, we set the cluster size to 8 for the rest of the experiments if unspecified otherwise.

Adaptivity to Data Distribution and Query Locality. In SSW, a peer selects the semantic



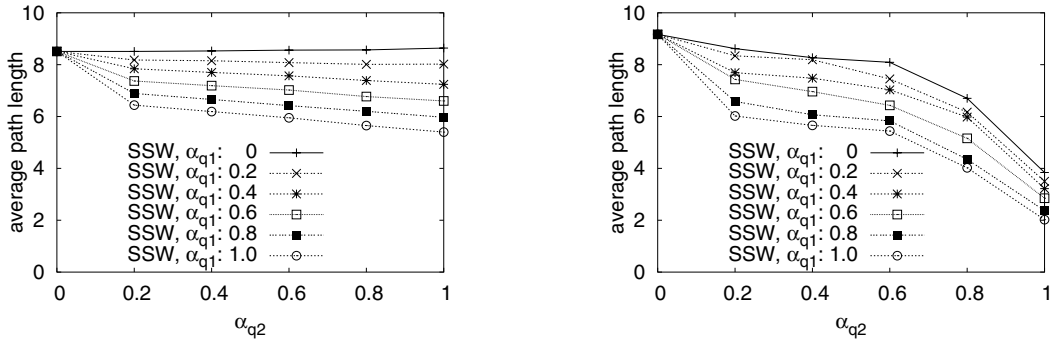
(a) synthetic data set (b) real data set

Fig. 7. **Effect of data distributions.**

centroid of its largest local data cluster as the join point (semantic position) when it joins the network. The rationale is that when data is more skewed around the centroid, fewer foreign indexes for data objects need to be published outside the cluster, thereby reducing the foreign index publishing cost. To better understand the effect of semantic closeness on the foreign index publishing cost, we vary α_{d2}, the skewness for Data-Zipf, from 0 to 1. In addition, we also vary α_{d1}, the skewness for Dataseed-Zipf, from 0 to 1 to observe the effect of data hot spots. The results are shown in Figure 7(a). As pointed out, a higher skewness lowers the foreign index publishing cost of SSW significantly. pSearch's foreign index publishing costs are in the range of 3500 and only decrease slightly under skewed data distribution. We omit the plot for pSearch from this figure to avoid distorting the graph.

Figure 7(b) compares the foreign index publishing cost under real data set with random data distribution and clustered data distribution. Similar to the trends observed under synthetic data sets, if data objects are rather clustered instead of randomly distributed, SSW reduces the foreign index publishing cost significantly. pSearch's foreign index publishing cost is also reduced under clustered data distribution. However, the reduction is not as significant as in SSW.

In SSW, long range contacts can be updated based on query history to exploit query locality. To study this improvement, we vary α_{q2}, the skewness for Query-Zipf, from 0 to 1. We also vary α_{q1}, the skewness for Queryseed-Zipf, to observe the effect of query hot spots. Figure 8 compares the search path length of SSW (a) without updates and (b) with updates of long range contacts (based on what described in Section V-B). Without any updates, query locality has little impacts on the results. With long range contact updates, however, query locality significantly enhances the search performance. For instance, we see nearly a 78% reduction in path length when α_{q2} increases from 0 to 1.0 with α_{q1} set to 1.0. pSearch's result (plot is not shown here)



(a) without updates (b) with updates

Fig. 8. **Effect of query distributions. Cost for pSearch is much higher (not shown for clarity).**

is similar to the one without update except that pSearch's path length is much higher (in the range of 40).

Tolerance to Peer Failures. Peer failure is a common event in P2P systems. Thus, a robust system needs to be resilient to these failures. To evaluate the tolerance of SSW to peer failures, a specified percentage of peers are made to fail after the network is built up. We then measure the ratio of searches that fail to find data objects existing in the network (we do not consider failures due to the data residing on the failed peers).

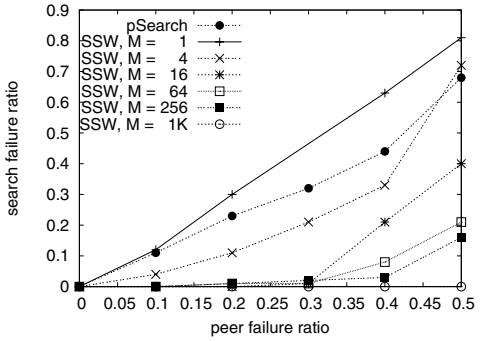
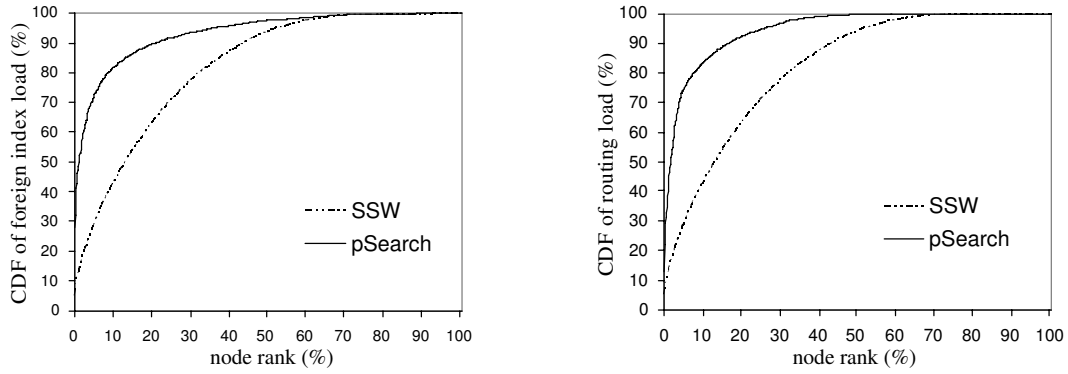


Fig. 9. **Effect of peer failure ratio on the failure of search operations ($\gamma = 0\%$).**

Figure 9 shows the fraction of searches that fail as a function of the ratio of induced peer failures (from 0% to 50%). Since the fault tolerance of SSW is largely dependent on the cluster size, we also consider different values for M (the cluster size) in these experiments. Even though each peer in pSearch maintains a large number of states (20), the search failure ratio grows rapidly with the ratio of peer failures. On the other hand, at cluster size of 1, SSW with much smaller number of states maintained per peer (2 short range contacts and 4 long range contacts) has similar search failure rate as pSearch. Increasing the cluster size to 4 substantially

improves SSW's fault tolerance. Beyond sizes of 4, the search failure ratio, even with as high as 30% peer failure, is very close to 0. These results reiterate the benefits of forming peer clusters.

Load Balancing. We evaluate the load balance of SSW from two aspects: index load and routing load. For the index load, we evaluate the distribution of the foreign index maintained at each peer under different data distribution patterns. Since the load is evenly balanced under the uniform distribution for both pSearch and SSW, we present only the results under the skewed data distribution in Figure 10(a). For presentation clarity, we show the CDF of foreign index load. As we expected, pSearch has a much more uneven index load distribution compared to SSW. For instance, the CDF of foreign index load for pSearch reaches 90% with the first 16% of nodes. In contrast, SSW displays a relatively even distribution of index load even under this skewed data set. This confirms our intuition that placing peers in the semantic space in accordance with their local data objects can effectively partition the search space according to data distribution.



(a) index load (b) routing load

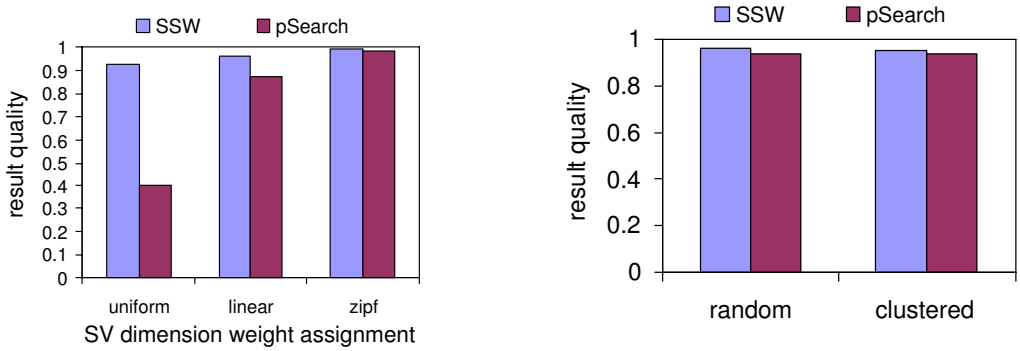
Fig. 10. **Distribution of foreign index load and routing load among the peers.**

Similarly, we have varied the query distribution in order to study the routing load distribution across the nodes, again with uniform and skewed distributions (this time with queries). We present the results for the skewed query distribution (α_{q1} and α_{q2} set to 1) in Figure 10(b). Similarly, we show the CDF of routing load. We find that the routing load is more evenly distributed in SSW compared to pSearch. For instance, the CDF of routing load for pSearch reaches 90% with the first 20% nodes. At this point, the CDF of routing load for SSW is only 57%. This is due to the introduction of randomness through the long range contacts and the good balance within a peer cluster itself.

2) Query Performance: In the following, we present the results specific to point* query and range query.

Point* Query. While the results on the search path length and search cost of point* query have

been presented earlier, here we present the result quality of point* query under different data sets. Since the result quality highly depends on the SV dimension weight assignments, we vary SV dimension weight assignments and the results are presented in Figure 11(a). SSW partitions the data space in a way adaptive to data distribution in the semantic space, while pSearch partitions the data space uniformly (into two equally-sized subspaces). We expect that our partition scheme adapts to different SV dimension weight assignments automatically while pSearch does not. This is confirmed by Figure 11(a). Different SV dimension weight assignments have little effect on the result quality of SSW (and search path length, search cost, and maintenance cost as well) . On the other hand, pSearch is very sensitive to different dimension weight assignments. In all three settings, the result quality of SSW is higher than pSearch.



(a) synthetic data set (b) real data set

Fig. 11. **Result quality of point* query under different data sets.**

Figure 11(b) shows the result quality of point* query under real data set with random data distribution and clustered data distribution. From this figure, we can see that the result quality obtained under real data set is similar to that under synthetic data set with Zipf dimension weight assignment. This is not surprising since the vector elements in the document SVs have weights following Zipf-distribution.

In addition, we vary the dimension of semantic space (SVs) from 10 to 100 and evaluate its effect. The search path length is more or less the same under different dimensions (the results are omitted). This is not out of expectation since the underlying routing structure of SSW-1D is based on the linearized naming space, which is not tied to the data dimensionality. On the other hand, there are some changes on the result quality. Figure 12 shows the result quality of point* query under uniformly distributed synthetic data set (the trends observed under other data sets are similar and omitted). From this figure, we can see when the dimension of data objects increases, the result quality using SSW decreases by a very small amount. On the other hand,

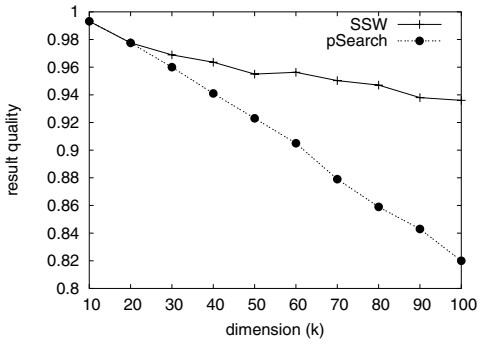


Fig. 12. **Effect of dimensionality on point* query.**

the result quality using pSearch is more sensitive to the dimensionality of data objects. This is expected since only partial subvectors are considered in pSearch, which degrades its result quality when the dimension is high.

Range Query. We conduct simulations on range query where the query radii are randomly drawn from the range $[0, R]$ (R indicates the maximum query radius, varying from 0 to 1). Figure 13 shows the results under uniformly distributed synthetic data set and real data set (the general trends observed under skewed data sets are similar). From this figure, we can see when the query radius is small, the search cost incurred by SSW is similar to that incurred by pSearch. When the query radius increases, the search cost incurred by SSW is substantially lower than that incurred by pSearch. This demonstrates that SSW supports range queries efficiently.

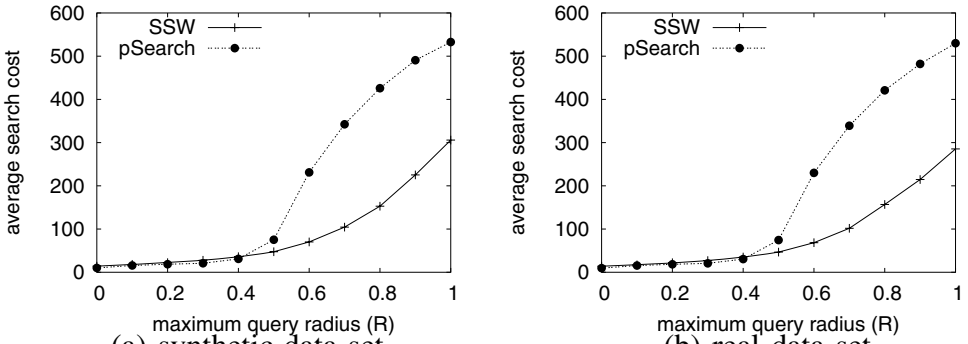


Fig. 13. **Effect of query radius on range query.**

In addition, we vary the dimensionality of data objects and evaluate its effect on the performance of range query. The results are shown in Figure 14. Similarly to what has been observed for point* query, the search cost of range query is insensitive to the dimension of data objects.

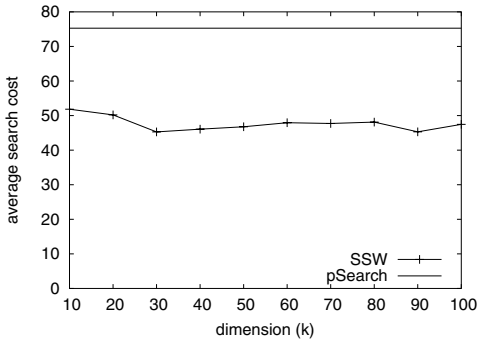


Fig. 14. **Effect of dimensionality on range query.**

VII. Conclusion

The voluminous information shared in peer-to-peer (P2P) systems mandates efficient semantic based search for data objects, which has not been adequately addressed in previous works. In this paper, we present the design of a new P2P overlay network, semantic small world (SSW), that supports efficient semantic based search. SSW integrates four ideas, i.e., semantic clustering, dimension reduction, small world network, and efficient search algorithm, into an overlay structure with following desirable features. It facilitates efficient search without incurring high maintenance overhead. By placing and clustering peers in the semantic space based on the semantics of their data objects, SSW adapts to distribution of data automatically, gains high resilience to peer failures and balances index and routing load nicely. Lastly, SSW can efficiently support various types of queries on data objects with high dimensions. All of the above advantages of SSW are verified through extensive experiments using both synthetic data and real data.

We are exploiting resource heterogeneity among peers by dynamically adjusting the number of join points, long range contacts, and cluster size. In addition, we are investigating how to extend SSW for data management in ad hoc and sensor networks.

VIII. Acknowledgement

This research was supported in part by the National Science Foundation under Grant no. IIS-0328881 and IIS-0534343.

References

[1] Text retrieval conference web site. http://trec.nist.gov/.

[2] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 685–696, 2005.

[3] J. Aspnes and G. Shah. Skip graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384 – 393, January 2003.

[4] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of International Conference on World Wide Web (WWW)*, pages 651–660, May 2005.

[5] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of ACM SIGIR conference on Research and Development in Information Retrieval*, pages 306–313, July 2003.

[6] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *Society for Industrial and Applied Mathematics (SIAM) Review*, 41(2):335–362, 1999.

[7] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of ACM SIGCOMM*, pages 353–366, August 2004.

[8] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. In *Proceedings of ACM SIGCOMM*, pages 97–108, August 2005.

[9] A. Iamnitchi, M. Ripeanu, and I. T. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 232–241, March 2002.

[10] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 661–672, September 2005.

[11] H. Jin, X. Ning, and H. Chen. Efficient search for peer-to-peer information retrieval using semantic small world. In *Poster Proceedings of International Conference on World Wide Web (WWW)*, pages 1003–1004, May 2006.

[12] J. Kleinberg. Navigation in a small world. *Nature*, 406(845), August 2000.

[13] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (SOTC)*, pages 163–170, May 2000.

[14] M. Li, W.-C. Lee, and A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *Proceedings of International Conference on Network Protocols (ICNP)*, pages 228–238, October 2004.

[15] M. Li, W.-C. Lee, and A. Sivasubramaniam. DPTree: A balanced tree based index framework for peer-to-peer systems. In *Proceedings of International Conference on Network Protocols (ICNP)*, pages 12–21, November

2006.

[16] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, March 2003.

[17] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. T. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, pages 536–543, May 2003.

[18] C. H. Ng, K. C. Sia, and C. H. Chang. Advanced peer clustering and firework query model in the peer-to-peer network. In *Proceedings of the 12th World Wide Web Conference (WWW) (Poster)*, May 2003.

[19] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, August 2001.

[20] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.

[21] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings Of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, August 2001.

[23] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM*, pages 175–186, August 2003.

[24] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve Freenet performance. In *Proceedings of INFOCOM*, June 2002.

[25] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.