Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 31 (2008) 346–357
computer communications
www.elsevier.com/locate/comcom

Hierarchical multidimensional search in peer-to-peer networks

D.A. Tran [a,*], T. Nguyen [b]

[a] Department of Computer Science, University of Massachusetts, Boston, MA 02125, USA
[b] School of EECS, Oregon State University, OR 97331, USA

Available online 15 August 2007

Abstract

We propose a P2P search solution, called EZSearch, that enables efficient multidimensional search for remotely located contents that best match the search criteria. EZSearch is a hierarchical approach; it organizes the network into a hierarchy in a way fundamentally different from existing search techniques. EZSearch is based on Zigzag, a P2P overlay architecture known for its scalability and robustness under network growth and dynamics. The indexing architecture of EZSearch is built on top of the Zigzag hierarchy, that allows both k-nearest-neighbor and range queries to be answered with low search overhead and worst-case search time logarithmic with the network size. The indices are fairly distributed over a small number of nodes at a modest cost for index storage and update. The performance results of EZSearch drawn from our performance study are encouraging.

Keywords: Peer-to-peer networks; Information retrieval; Similarity search

1. Introduction

One of the most important problems in information retrieval is similarity search. Informally, the problem is: given a similarity query, whether it is a k-nearest-neighbor (k-NN) query or a range query, we seek a set of contents (or data objects) that are the most relevant to the search criteria according to some semantic distance function. Our goal is a decentralized solution to this problem for P2P networks.

Search techniques have been proposed for unstructured P2P networks. Most of them, however, are based on query flooding (e.g., [4]), which is communicationally inefficient, or based on random walking (e.g., [15]), which is ineffective for similarity queries. Therefore, a structuralization of the network is more favorable, which should include two constituent components: a communication architecture and an indexing architecture. Their design must consider the following issues:

- *Communication architecture:* Without central servers, we need a distributed overlay to interconnect the nodes for efficient navigation between them. A main problem in designing this overlay is due to the dynamics of P2P networks; nodes can join, leave, publish, and remove data at any time and so connections and data coherence may be lost frequently. The overlay should be self-adjustable in a way to minimize the effect of such changes. The overlay must be robust in that failure recovery is fast and graceful to avoid interruption in the on-going service. To be part of the network, a node needs not know everything about the network. For efficiency, a node should keep track of just a small number of other nodes and process a small share of service load. Since we target large-scale networks, the overlay must be scalable with the network size.
- *Indexing architecture:* The accumulative volume of distributed data in a P2P network is large and potentially keeps growing. Therefore, instead of employing a number of centralized servers (either dedicated or selected among participating nodes) to carry the load for the entire network, we should let every node in the network share the load of indexing and searching. Then arise the questions: how to determine which node to store an

* Corresponding author. Tel.: +1 617 287 6483.
E-mail addresses: duc@cs.umb.edu (D.A. Tran), thinhq@eecs.oregonstate.edu (T. Nguyen).

0140-3664/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2007.08.007

index? publish or remove an index? achieve index-load balancing? deal with the curse of dimensionality? organize and manage these distributed indices so that search can be performed fast and efficiently? etc. The indexing architecture, built atop the communication architecture, must answer these questions.

1.1. Existing solutions

The existing structural search solutions fall into two main approaches: *flat-based* and *hierarchy-based*. In the flat-based approach, the communication architecture organizes the nodes such that each node is mapped to a non-overlapping region of a multi-dimensional space and adjacency links are added between nodes if their corresponding regions are adjacent [1,11], and/or satisfy some relationship [17]. The indexing architecture stores in each node the indices of all the objects that directly, or indirectly via a transformation, belong to the region assigned to this node. Search for an object is reduced to routing to the node whose region contains the object. Nodes with similar data may also be clustered [9] and additional links between distant nodes may be added [9,17] to reduce the routing distance among them. Examples of the flat-based approach include DHT [3,11,12,17,21], small-world model [9], and Voronoi-diagram [1] based search techniques.

The flat-based approach is less efficient in high dimensionality because of the many adjacency links a node has to maintain with neighbor nodes. The number of neighbors is typically $\Omega(d)$ where d is the dimension. This dimension can be in the hundred's as in the case of image and document retrieval, causing the communication architecture a significant maintenance overhead. Several techniques [9,13,16,18] employ multidimensional reduction methods to reduce the data space to some manageable dimensionality. However, as a tradeoff of dimensional reduction, search in the lower dimensionality cannot be of the same quality as in the original dimensionality; it either must search a wider scope or the results may only be partial.

The hierarchy-based approach [8,10] organizes the nodes into a hierarchy and provides a node-to-index mapping to this hierarchy so that it can utilize a multidimensional indexing tree structure in traditional databases [14]. The success of traditional multidimensional indexing has been well-documented, making the hierarchy-based approach potentially more suitable than the flat-based counterpart for high-dimensional retrieval. A hierarchy is also more scalable and tends to cope better with network dynamics.

1.2. Contributions

Research in the hierarchical approach has been sparse, the main reason due to the lack of an efficient hierarchical communication architecture that can best work with a successful multidimensional indexing scheme such as the R-tree [6] and kd-tree [2]. In the present paper, we propose a fundamental change in the communication architecture used in the search system. Our proposed search solution, called EZSearch, uses the Zigzag hierarchy to significantly alleviate the above problems. Zigzag is a hierarchy of dual-head clusters and provides the following *dimension-invariant* features:

- Nodal degree is bounded by a constant, limiting the amount of search traffic coming in or exiting every node
- The number of nodes that a node needs to know is bounded by a constant on the average
- The hierarchy diameter is logarithmic with the network size, therefore a search can be completed in logarithmic time
- Cluster splitting/merging is infrequent and requires at most a constant number of link changes
- A node's failure/departure is handled regionally and requires at most a constant number of link changes

Because the hierarchies used by existing techniques cannot provide all these features, EZSearch has a more efficient and robust communication foundation than the others. We propose a scheme for the indexing architecture atop the Zigzag overlay. The challenge of this task is to decide how clusters are formed, split, and merged under network dynamics so that the indexing costs can be minimized and balanced. Our indexing scheme addresses effectively the issues of communication cost, storage cost, and the update cost due to network changes.

The Zigzag hierarchy is a result of our earlier work [20]. Our idea of using it as the overlay for P2P search is initiated in [19]. As a significant extension from [19], the present paper studies extensively and comprehensively all the cases involved in the design of EZSearch. We provide search algorithm for kNN queries. We propose algorithms of more detail for node addition and departure, cluster merging and split, and index publication. We have also conducted a new and extensive performance study and present the results in the present paper. None of these extensions are included in the preliminary work [19].

The remainder of this paper is organized as follows. We present the Zigzag hierarchy in Section 2. Next in Section 3, we propose indexing and search mechanisms on top of Zigzag and how to construct and maintain this hierarchy under network dynamics. We provide the results of our simulation study in Section 4. We conclude the paper in Section 5 with pointers to our future research.

2. Zigzag hierarchy

A Zigzag hierarchy of N nodes is a multi-layer hierarchy of clusters. The size of a cluster is bounded by $[z, 3z]$ where $z \geqslant 4$. Parameter z, called the z-factor of the hierarchy, controls the size of the cluster and so the height of the hierarchy. The size range $[z, 3z]$ provides flexibility for the hierarchy to adapt with nodes joining and leaving. In each

cluster, two nodes are designated as the "head" and the "associate-head", respectively. The clusters at a layer of the hierarchy are formed by the heads of the layer right below. The layer 0 contains all the nodes. The top layer contains only one cluster and since it does not have any layer above it, this cluster may have any size in $[2, 3z]$. It is easy to prove that the number of layers of this hierarchy is in $[\log_{3z} N, \log_z N + 1]$.

Fig. 1 shows a possible Zigzag-4 hierarchy (the z-factor is 4) of 52 nodes. All 52 nodes appear at layer 0. They are partitioned into 13 clusters, each containing four nodes. From left to right, the head and associate-head of these clusters, respectively, are node 2 and node 1, node 5 and node 6, node 9 and node 10, node 13 and node 14, etc. All the layer-0 heads (i.e., nodes 2, 5, 9, 13, 17, 22, 26, 29, 33, 37, 41, 45, and 49) appear at layer 1. Since there are 13 such nodes at layer 1, which is more than $3z = 12$ nodes, we partition them again into 2 smaller clusters to satisfy the cluster size condition. The head and associate head of the first layer-1 cluster are nodes 22 and 17, respectively. The head and associate-head of the other layer-1 cluster are chosen to be nodes 26 and 29, respectively. The layer-1 cluster heads (nodes 22 and 26) automatically appear at layer 2 and form a single cluster. We do not need to partition in this layer anymore because the cluster size is 2, which is already less than $3z$.

We denote by *head*(.) and *ahead*(.) the head and associate-head, respectively, of a cluster or a node. Below are the *terms* we use for the rest of the paper:

- *Foreign head:* If X and Y are clustermates at layer $j > 0$ and Z is a clustermate of X at layer $j - 1$, then Y is called a "foreign head" of Z. E.g., in Fig. 1, node 2 is a foreign head of nodes 6, 7, 8, 10, 11, 12, 14, 15, 16, 18, 19, 20, 21, 23, and 24.
- *Super cluster:* A layer-j cluster is the "super cluster" of any layer-$(j-1)$ cluster whose head appears in the layer-j cluster.
- *Sibling cluster:* Two clusters are "sibling" if they have the same super cluster.

We define links between nodes in the Zigzag hierarchy as follows:

- *Intra-cluster links:* In a cluster, the associate-head links to every other non-head node. *E.g., in Fig. 1, associate-head 17 of its layer-1 cluster has a link to all of its layer-1 non-head clustermates (nodes 2, 5, 9, 13).* An exception applies to the highest-layer cluster, where all nodes link to the head.
- *Inter-cluster links:* The associate-head of a cluster must be linked from one of its foreign heads. *E.g., in Fig. 1, associate-head 18 at layer 0 has a link from node 13, which is one of node 18's foreign heads.*

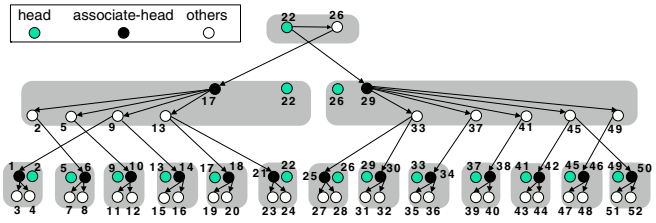

Fig. 1. A zigzag-4 hierarchy of 52 nodes.

The above rules guarantee a near-balanced tree structure including all the nodes; we call this tree the Zigzag tree. This tree has *height bounded by* $2\log_z N + 1$ and *node degree bounded by* $3z - 1$. To maintain the hierarchy, each node must track the existence of its clustermates, its parent node, and its child nodes. However, traffic is only forwarded along the hierarchy links and thus limited by its node degree. Under network dynamics, the Zigzag hierarchy is highly robust. Due to additions of new nodes and failures/departures of exiting nodes, a cluster may overflow or underflow, in which case it has to be split, or merged with another cluster, respectively. Zigzag provides the following robustness properties (see [20] for complete proofs and algorithms): (1) *Recovery of a node failure requires at most* $O(z)$ *link changes*; and (2) *A cluster split or merger requires at most* $O(z)$ *link changes*. All these overheads are independent of the network size, making the Zigzag hierarchy a highly scalable communication architecture.

3. Indexing and search mechanisms

EZSearch organizes nodes into a Zigzag hierarchy and assigns to each cluster an index zone. All the zones at layer 0 form a complete disjoint partitioning of the entire index space. At higher layers, the zone of a cluster is the union the zones of all the clusters that call it their supercluster. Therefore, the index zone of the top-most cluster is the entire index space. Without loss of generality, an object x is described as a point $(x_1, x_2, \ldots, x_d)$ in the unit d-dimension hypercube.

The description of a cluster C's zone (denoted by $zone(C)$) is stored at its associate-head $ahead(C)$. In addition, each node P stores a list of pairs $(ch_i, zone(ch_i))$ for each child ch_i of P in the Zigzag tree, where $zone(ch_i)$ is the index zone covered by this child. The index zone covered by a node P, denoted by $zone(P)$, is the union of its child zones. If P has no child, $zone(P) = \emptyset$.

For example, we consider the hierarchy in Fig. 1. Suppose that the index zones owned by the 13 layer-0 clusters are $I_1, I_2, \ldots, I_{13}$ (respectively, from left to right); e.g., $zone(1) = I_1$, $zone(5) = I_2$, $zone(9) = I_3$, etc. Because node 9 has two children (peer 1 and peer 14), node 9 stores the information $\{(1, I_1), (14, I_4)\}$ and the description of its zone $zone(9) = I_1 \cup I_4$. The index zone assignments are similar for other nodes and shown in Fig. 2. Since nodes other than the heads and associate-heads at layer 0 do not manage any index zone, they are not present in this figure.

A valid index zone in EZSearch must be a set of disjoint hyperrectangles. An example for 2-dimensionality is given

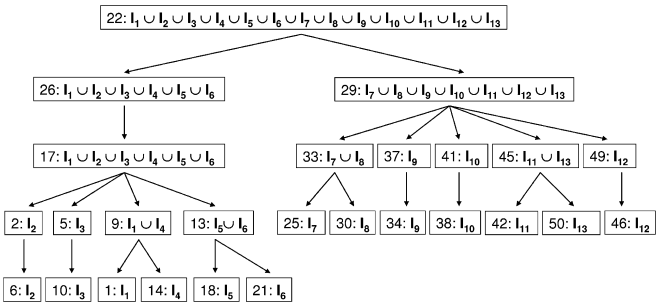

Fig. 2. Index zone assignments of the 52-node Zigzag-4 hierarchy shown in Fig. 1.

in Fig. 3 which shows the index zones of several nodes of the hierarchy illustrated by Fig. 1 and 2. Each layer-0 cluster's zone I_i ($i = 1, 2, \ldots, 13$) is a rectangle. All these rectangles form a complete partitioning of the unit square. The set of gray rectangles therefore represents *zone*(17) while the set of striped rectangles represents *zone*(45).

The actual indices are *physically* stored at the head nodes of layer 0. Once an index zone is determined for each cluster, the index of an object x is stored in the cluster (to be exact, the head of this cluster), whose zone contains x. For cost-effectiveness and search efficiency, the index zone assignment should achieve the following criteria:

- *Indexing cost:* The distance between the node that owns an object and the node that stores this object's index should be short, so that the cost of index publication and maintenance is kept small. This is already achieved by EZSearch because the Zigzag hierarchy guarantees that the path length between any two nodes is bounded by $O(log(N))$. We should, however, need to minimize the number of indices that may be migrated due to cluster split/merge events.
- *Index balancing:* The indices are stored only at the head nodes of layer 0. It is desirable that the index storage is fairly distributed over these nodes. This balancing also helps the search traffic be fairly distributed on the hierarchy

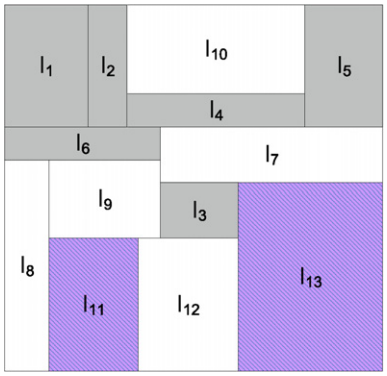

Fig. 3. Examples of index zones in 2-dimension for the hierarchy in Fig. 1 and Fig. 2: the set of gray rectangles and set of striped rectangles represent the index zones of node 17 and 45, respectively. Each single rectangle represents an index zone at layer 0.

- *Index locality:* Each index zone may contain more than a hyperrectangle and so it is not always guaranteed that all indices covered by the same cluster are near each other. Therefore, index locality should be preserved in every cluster so that a range search can be done quickly and efficiently. In Fig. 3, both *zone*(17) (gray rectangles) and *zone*(45) (striped rectangles) are not optimal because each contains indices far away from each other.

In the following subsections, we first present how search can be conducted given an existing zone assignment, and then discuss how to obtain a good zone assignment, to construct the hierarchy, and to main these two components under network dynamics.

3.1. Search algorithms

Given the index zones already assigned to the nodes, we provide search algorithms for range and kNN queries below.

3.1.1. Range queries

A range query can be specified as a hyperrectangle. The query follows the links in the Zigzag tree to branches that lead to the index zones *overlapping* with the query. Each time a query visits a node, the query is forwarded to the node's parent if the visited zone does not strictly contain the query. The query is also forwarded to those child nodes whose zone overlaps with the query. As a result, there may be multiple instances of the same query, called subqueries, travelling different branches of the hierarchy. All the subqueries will eventually reach the layer-0 clusters whose zone overlaps with the query region, where relevant objects will be collected.

Our search algorithm guarantees completeness. In other words, it retrieves all the results that satisfy the query. Until layer 0 is visited, the search requires only overlap checking instead of computing the intersections between the visited index zones with the query, and so avoids the complexity and time of doing the latter in high dimensionality.

The search path length is at most the diameter of the Zigzag tree, and therefore $O(\log_z N)$. The search time also depends on how long it takes a visited node to check whether its zone overlaps with the query. Since the query is a hyperrectangle and the node's zone is a set of hyperrectangles, checking for overlap should be very quick. The search overhead is proportional to the total number of nodes contacted by all the subqueries. This overhead depends on the range of the original query; in our performance study, we found this overhead indeed very small.

3.1.2. k-NN query

Consider a k-NN query that finds the k objects most similar to the query point $q = (q_1, q_2, \ldots, q_d)$. First, we apply the range-query search algorithm on q to find the layer-0 cluster whose zone contains q. This search is quick

with little overhead because q is a point, not a range. Suppose that the query reaches the cluster C such that $q \in zone(C)$. The associate-head of C takes the steps below:

1. If $zone(C)$ contains fewer than k objects, compute ε_0 as the maximum distance between q and the objects in $zone(C)$. Otherwise, ε_0 is the distance between q and the object k-closest to Q in $zone(C)$.
2. Submit a range query $q_{\varepsilon_0} = [q_1 - \varepsilon_0, q_1 + \varepsilon_0] \times [q_2 - \varepsilon_0, q_2 + \varepsilon_0] \times \ldots \times [q_d - \varepsilon_0, q_d + \varepsilon_0]$, but only accept objects x whose distance with q is less than ε_x. Denote the result set by RESULT.
3. There are two cases:
   (a) If $|\text{RESULT}| \geqslant k$, select the best k objects from RESULT and send the location links to the node initiating q.
   (b) If $|\text{RESULT}| < k$, set $\varepsilon = \epsilon_0$ and follow the algorithm below to find the remaining top $k' = (k - |\text{RESULT}|)$ objects:
      i. Compute range query $q' = q_{2\varepsilon} \backslash q_{\varepsilon}$. Submit query q' but only accept objects x whose distance with q is less than 2ε. Denote the result set by RESULT′.
      ii. If $|\text{RESULT}'| \geqslant k'$, return the best k' objects from RESULT′ to the node initiating q. Otherwise, set $\varepsilon = 2\varepsilon$, $k' = k' - |\text{RESULT}'|$, and repeat Step (i).

The results are always correctly found. The worst-case search path length to get all k nearest objects is $D = O(\log_z N) + p \times D'$, where D' is the length due to Step 3b and p the probability that Step 3b occurs. Since ε is doubled each time Step 3b(i) is re-encountered, there are no more than $\log_2(1/\varepsilon_0)$ times of running Step 3b(i). Therefore, $D' = \log_2(1/\varepsilon_0)O(\log_z N)$, and so, $D = (1 + p \times \log_2(1/\varepsilon_0))O(\log_z N)$. In the next section, we discuss mechanisms for construction and maintenance of the network and index zones. One of our goals is to construct fairly-sized index zones at layer 0. If this goal is achieved, assuming that an average node contributes at least one object, the size of each zone at layer 0 is at least $N/(3z)$. This size is greater than z when N is large enough, and, therefore, we need not run algorithm 3b and the worse-case search path length is $O(\log_z N)$. The search overhead should be small because the search overheads for finding cluster C and for processing range query Q_{ε_0} are small.

3.2. Hierarchy construction: initial case

Initially, there is only one node in the network. It serves as the head and associate-head of its self-formed cluster C. This cluster grows larger as subsequent nodes join. The index zone of this cluster is $zone(C) = I = [0,1)^d$. When the cluster overflows, C is partitioned into two smaller clusters, C_0 and C_1, whose sizes are in the interval $[z, 3z]$; some nodes of C forms cluster C_0, the remainder forming cluster C_0.

We propose to partition I along a dimension l into two halves $I_{0l} = [0,1)^{l-1} \times [0,1/2) \times [0,1)^{d-l}$ and $I_{1l} = [0,1)^{l-1} \times [1/2,1) \times [0,1)^{d-l}$, each to be owned by C_0 and C_1. It is possible that a node in cluster C_0 has an object in I_{1l}; in this case, we "migrate" the index of this object to C_1. Similarly, if a node in cluster C_1 has an object in I_{0l}, this object's index is stored in cluster C_0. We want to *minimize the number of such indices while balancing the index load* between cluster C_0 and C_1. For this purpose, the following simple algorithm is run at node $head(C)$:

1. To balance index load, select dimension l such that $(\sum_{P \in C} n^P_{1l} - \sum_{P \in C} n^P_{0l})^2$ is minimum where $n^P_{0l} = \text{cardinality}(\{\text{object}x\text{in}P | x \in I_{0l}\})$ and $n^P_{1l} = \text{cardinality}(\{\text{object}x\text{in}P | x \in I_{1l}\})$.
2. To keep index migration overhead low, partition C into C_1 and C_2 such that $|C_1|, |C_2| \in [z, 3z]$ and $\sum_{P \in C_0} n^P_{1l} + \sum_{P \in C_1} n^P_{0l}$ is minimum. This summation can be expressed as

$$\sum_{P \in C} n^P_{0l} + \sum_{P \in C_0} (n^P_{1l} - n^P_{0l}) \quad (1)$$

   So, to minimize (1), we greedily add z nodes P with least value of $(n^P_{1l} - n^P_{0l})$ to C_0 (initially empty), and then keep adding nodes with least value of $(n^P_{1l} - n^P_{0l})$ as long as $\sum_{P \in C_0} (n^P_{1l} - n^P_{0l})$ decreases and $|C_0| < 3z/2$.

As a result of this algorithm, we have two newly created clusters C_0 and C_1 with the index zones I_{0l} and I_{1l}, respectively. For each cluster C_i we randomly selects two nodes as its head $head(C_i)$ and associate-head $ahead(C_i)$ (the old head of cluster C, however, is preferred to remain head of the newly created cluster it belongs to). For those objects that belong to a node in cluster C_i but map to points in zone $I_{(1-i)l}$, their index will be stored at $head(C_{1-i})$ of cluster C_{1-i}. The description of the index zone of cluster C_i is stored at its associate-head $ahead(C_i)$, as required by the zone assignment policy.

Once the newly created clusters have designated their head and associate-head, the heads will automatically belong to layer 1 and form a new cluster. Since layer 1 now is the highest layer, only the head needs to be designated; this head is randomly chosen between the two member nodes. The index zone owned by this cluster is the union of the zones owned by its child clusters; in this case, it is $I_{0l} \cup I_{1l} = I$.

3.3. Hierarchy construction: incremental update

We assume that a Zigzag hierarchy of nodes currently exists with the corresponding zone assignment. This section details algorithms for the following sub-problems: (1) a node publishes or removes an object, (2) a new node joins the network, and (3) an existing node departs from the network.

3.3.1. Object publication and removal

Suppose that a node P wants to publish an object x. The index of this object is to be stored at the layer-0 cluster whose index zone contains x. For this purpose, P generates a publication request in the form of a point query $q = x$ and then the range-query search algorithm is used to find the destination cluster. Object removal is similar to object publication, except that after finding the cluster whose zone covers the deleted object, the corresponding associate head removes the index of this object from its index database.

3.3.2. Node join

A node P_{new} wants to join the network. It will be added to the cluster C with highest index load at layer 0. This policy helps *balance the index load* among clusters. Indeed, if a cluster has a heavy index load, it will be split quickly because many new nodes join this cluster and make it overflow. The question is how P_{new} knows this special cluster C? The answer is simple. In the current version of data structures at each node, a non-leaf node on the Zigzag tree stores (ch_i,$zone(ch_i)$) for each child ch_i. We extend this structure by allowing a non-leaf node to also store the ID of the child that, among all the children, leads to a layer-0 cluster storing the most indices. As a result, the root of the Zigzag tree always knows the ID of the cluster at layer-0 with maximal index load. The join request travels the following path to reach the cluster C for P_{new} to join: (1) P_{new} sends the join request to an existing node P_{con}; (2) P_{con} forwards the request upstream until reaching the root node P_{root}; and (3) P_{root} informs P_{new} of the layer-0 cluster C with maximum index load. P_{new} will contact the associate-head node of C to join it.

The join results in only one new link added to the Zigzag tree. The join delay consists of two delays: delay due to searching for cluster C and delay due to publication of new objects. In the worst case, the length of the longest travel path is $O(\log_z N)$, and so we expect the join delay to be short. The join overhead consists of the total number of nodes contacted during the search for cluster C and new object publications; therefore, it is at most $O(m\log_z N)$ where m is the number of objects whose index needs migrating.

3.3.3. Node removal

A node may leave the network intentionally or fail to exist in the network. Either way, the layer-0 clustermates of the quitting node can detect this departure because members of the same cluster periodically keep track of each other's existence. Let us name the departing node P_{quit} and its layer-0 cluster C. There are two cases:

- *The highest layer of P_{quit} is layer 0* (i.e., it does not appear at any higher layer): We do not need to process further unless P_{quit} is the associate-head of C. If P_{quit} is the associate-head, a random non-head node $P_{promoted}$ in C will be selected by $head(C)$ to assume the associate-head role. Furthermore $zone(P_{promoted})$ is set to the index zone associated with cluster C. $P_{promoted}$ knows this information from $head(C)$.
- *The highest layer of P_{quit} is layer $i_P \geqslant 1$*: Since P_{quit} must be the head of every layer-i cluster it belongs to ($i < i_P$), a different node has to be selected as the new head for each of such clusters. The solution is straightforward. A randomly selected layer-0 non-associate-head clustermate of P_{quit}, say node $P_{promoted}$, will assume the position of P_{quit} in every cluster P_{quit} used to be part of. In addition, since P_{quit} is the head of its layer-0 cluster, it may store some indices; these indices will be transferred to $P_{promoted}$.

In the case P_{quit} departs due to a failure, all the indices it stores (if any) are lost and, also, the indices of the objects of P_{quit} stored remotely become invalid. To address this, it is recommended that we employ the following mechanism. If a node P_1 has an object x whose index is stored at another node P_2, these two nodes "ping" each other periodically to check their existence. The ping period should be long enough to avoid heavy communication overhead. If P_1 does not hear from P_2, P_1 re-publishes x. Similarly, if P_2 does not hear from P_1, the former deletes all the indices associated with P_1. This mechanism also helps the removal of an object become very quick because we would not have to search for the node that stores the index for the deleted object; this node is already known by the owner of the object.

Since the zone covered by a node is the union of the children's zones, index zones are updated accordingly for those nodes that change links as a result of a node's removal. The maximum number of nodes that change links is at most the number of children of the departing node; hence, at most $3z - 1$. The overhead to deal with the departure of a node is therefore small and independent of the system size.

3.4. Hierarchy construction: cluster split and merging

As nodes join and depart a cluster may overflow or underflow, in which case it must be split into smaller clusters or merged with another cluster, respectively. This task involves two main steps: (1) In a cluster's split we must decide which nodes belong to each new cluster, while in a cluster's merger we must decide which other cluster to merge with; (2) Once such a decision is made, the Zigzag hierarchy will be updated accordingly with no more than $O(z)$ link changes. Since our earlier work [20] provides the algorithms for Step (2), in the present paper we present only the criteria and algorithms for Step (1).

3.4.1. Cluster split

We consider splitting an overflowed cluster C at layer h into two clusters C_a and C_b. Let $\mathcal{P} = \{P_1, P_2, \ldots P_n\}$ ($n = 3z + 1$) define the set of nodes in C. We present the algorithm for $h \geqslant 1$ in this section. This algorithm can be

modified to work with the case $h = 0$ and so the this case is omitted here due to lack of space.

Let C_1, $C_2, \ldots, C_n$ be the clusters headed by P_1, $P_2, \ldots, P_n$, respectively, at level $h-1$. The set of nodes $\mathcal{P}$ is halved to form $C_a = \{P_1, P_2, \ldots, P_{\lfloor n/2 \rfloor}\}$ and $C_b = \{P_{\lfloor n/2 \rfloor+1}, .., P_n\}$. Hence,

$$zone(C_a) = \bigcup_{P_i \in C_a} zone(C_i) \tag{2}$$

$$zone(C_b) = \bigcup_{P_i \in C_b} zone(C_i) \tag{3}$$

We find C_a and C_b such that $volume(box(C_a) \cap box(C_b))$ is minimum, where $box(C_i)$ is the minimal hyperrectangle bounding $zone(C_i)$. This criterion guarantees that each cluster's index zone contains indices close to each other. Since an index zone may contain many hyperrectangles, computing the intersection between two zones may be time consuming. Therefore, we propose the following heuristic algorithm:

- For each dimension t, sort $C_1, C_2, \ldots, C_n$ into the ordered list $C_{t_1}, C_{t_2}, \ldots, C_{t_n}$ such that the t-dimension maximum coordinate of $box(C_{t_i})$ is non-decreasing. Then, let $C_a = \{P_{t_1}, P_{t_2}, .., P_{t_{\lfloor n/2 \rfloor}}\}$ and $C_b = \{P_{t_{\lfloor n/2 \rfloor+1}}, .., P_{t_n}\}$, and compute

$$box_a = box\left(\bigcup_{P_{t_i} \in C_a} box(C_{t_i})\right) \tag{4}$$

$$box_b = box\left(\bigcup_{P_{t_i} \in C_b} box(C_{t_i})\right) \tag{5}$$

$$V_t = volume(box_a \cap box_b) \tag{6}$$

- After all dimensions are considered, select the dimension t and the corresponding C_a and C_b that minimizes the value V_t.

Since $box(C_i)$ can be computed off-line by the associate-head of cluster C_i (i.e., during its the idle time), the above algorithm runs in $O(d(z\log_2 z + dz)$ time. The algorithm is run at the associate-head of cluster C. A cluster split does not result in index migration unless the split occurs at layer 0. Therefore, the main overhead of a cluster split is due to $O(z)$ link changes and the reassignment of index zone to the nodes that change links.

3.4.2. Cluster mergence

We consider merging an underflowed cluster C at layer h with another cluster C'. We find C' among the sibling clusters of C such that (1) $|C + C'| \in [z, 3z]$, and (2) $volume(box(C \cup C'))$ is minimum.

These criteria guarantee that the combined zone contains indices close to each other and looks similar to the form of a hyperrectangle. As a result, the intersection of a query with this zone will result in few subqueries. Since there are no more than $O(z)$ sibling clusters, we can devise an algorithm that finds the best sibling C' in $O(zdh)$ time. In the worst case, $h = O(\log_z N)$, and so this complexity is $O(zd\log_z N)$. To speed up this algorithm, we can approximate $volume(box(C \cup C'))$ by

$$volume(box(box(C) \cup box(C'))) \tag{7}$$

and hence reducing the running time to $O(zd)$. The algorithm to find C' is run at the associate-head of cluster C.

Since the true indices are stored at layer 0 and higher layers only store the description of index zones, a cluster merging does not result in index migration unless the merger occurs at layer 0. Therefore, the main overhead of a cluster merging is due to $O(z)$ link changes and the reassignment of index zone to the nodes that change links.

4. Performance evaluation

We verified the correctness of EZSearch and assessed its performance via simulation. Since we wanted to model a highly dynamic network, we set the z-factor to a small number $z = 5$ so that cluster split and merging occur often. With this z-factor, no cluster contains more than 15 nodes or fewer than 5 nodes. We also let the nodes join the network according to a Poisson process at a rate $\lambda = 6$ arrivals per second. Each node had an active session, after which it quitted the network. The session's period was generated according to a Pareto distribution with $pdf(x; k, x_0) = kx_0^k/x^{k+1}$ for $x > x_0$. Pareto has been widely used to model the node lifetime in a distributed network [5]. We set $k = 1.5$ and $x_0 = 10$ min as also used in [5]. With this configuration, the expected lifetime of a node was $EX = kx_0/(k-1) = 30$ min and the minimum lifetime was $x_0 = 10$ min.

A data object was generated as a d-D uniformly random point in $[0, 1)^d$. We considered three cases: 3-dimension, 6-dimension, and 9-dimension. A node has randomly between 0 and 10 objects. We allowed for a total number of 12,035 nodes to join and quit the network. When the network stabilized (i.e., no more node joined or departed), it contained 8900 nodes and approximately 45,000 indices. We then started querying the network with 800 kNN and range queries posted by random nodes:

- 400 range queries: The range of a query can be 5%, 10%, 15%, or 20% of the index space, each case generating 100 queries
- 400 k-NN queries: k can be 5, 10, 15, and 20, each case generating 100 queries

During the network construction phase, we collected the statistics about the control overhead per node, index storage overhead, its distribution in the network, and index migration overhead due to each cluster split and merging. During the query phase, for each query, the search always attempted to return *all* the results that satisfy the query, and we collected the information about the search time and search overhead. In the following section, for the sake

of convenience, we use the "present tense" instead of "past tense" in discussing the results.

4.1. Node overhead

To assess the efficiency of the Zigzag hierarchy, we compute the number of links (i.e., degree) of each node and the contact list (i.e., number of neighbors that each node needs to keep track of in order to maintain the hierarchy structure of the network). A node should have a small degree to limit the search traffic passing through it. A node should also have a short contact list so it does not have to check the existence of too many nodes; hence, less communication involved.

Fig. 4 shows that no node has to forward or receive search traffic on more than 13 links. This is close to the bound $3z - 1 = 14$ found in our theoretical analysis (see Section 2). Fig. 5 provides the number of contact nodes that each node has to keep track of in order to maintain the network hierarchy. It is understandable that since we use a hierarchy, the nodes appearing at high layers of the hierarchy should have more contact nodes. However, the worst-case node needs to know only about 30 other nodes (out of 8900 nodes in the network), and more than 80% of the nodes each need to know fewer than 10 other nodes. This overhead is considered small; it is still a lot smaller the overhead in many existing DHT-based techniques, in which the number of contacts per node is at least the number of dimensions. In many applications, the dimensionality could be more than 100. The Zigzag hierarchy is therefore highly efficient.

4.2. Index migration overhead

Due to the dynamics of the network, clusters may be split or merged to satisfy the cluster-size bounds, and so indices may have to be moved between nodes. One goal of EZSearch is to keep the overhead of index migration low. During the hierarchy construction phase, there are almost 2000 cluster gers. Figs. 6 and 7 plot the number of indices moved after each cluster split/merger for two cases: 3-D and 9-D, respectively. In both cases, each cluster split or merger causes 30 moved indices on average. When the network is growing larger, some splits and mergers result in more indices moved but never more than 92 migrations (out of 45,000 indices in the network). Also, many splits and mergers result in just a few index migrations (less than 20). Therefore, EZSearch addresses the network dynamics very well, even when we increase the dimension from 3-D to 9-D. This study substantiates our use of the Zigzag hierarchy as a highly efficient communication architecture for information retrieval.

4.3. Index storage overhead

Load balancing is desirable in any distributed system. It is achieved in EZSearch as illustrated in Figs. 8 and 9.

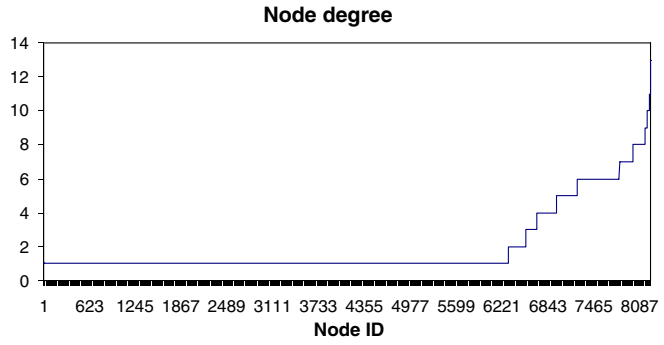

Fig. 4. The node degree for every node in the network.

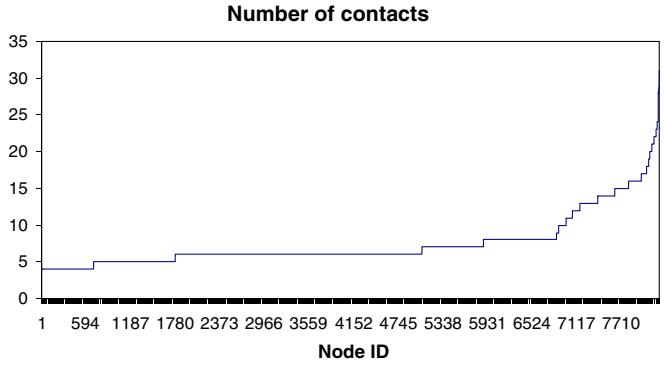

Fig. 5. The number of contact nodes for each node in the network.

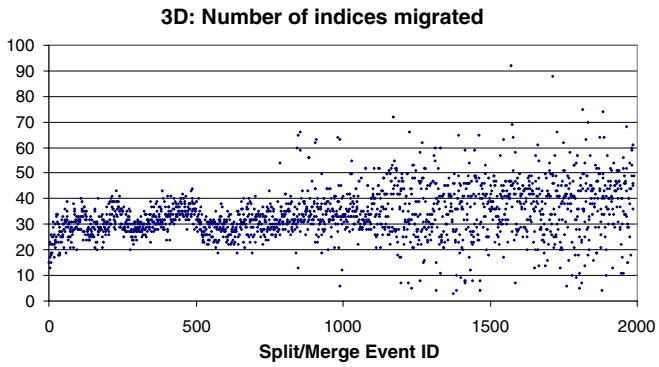

Fig. 6. 3-Dimension: index migration overhead.

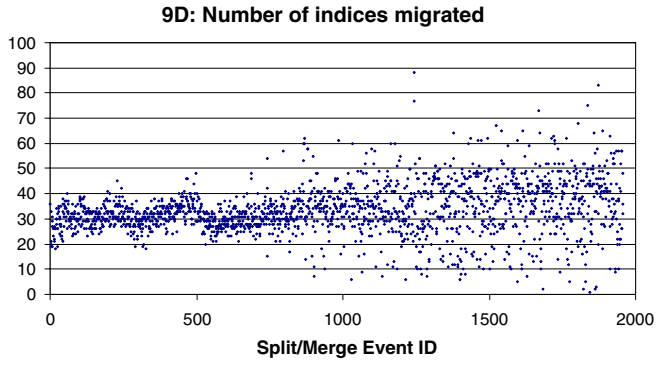

Fig. 7. 9-Dimension: index migration overhead.

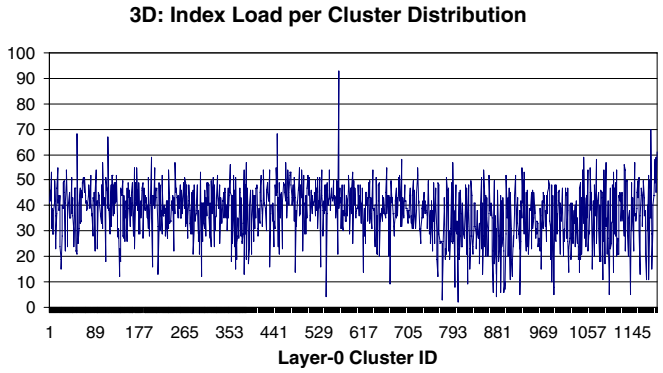

Fig. 8. 3-Dimension: the distribution of indices over all layer-0 clusters.

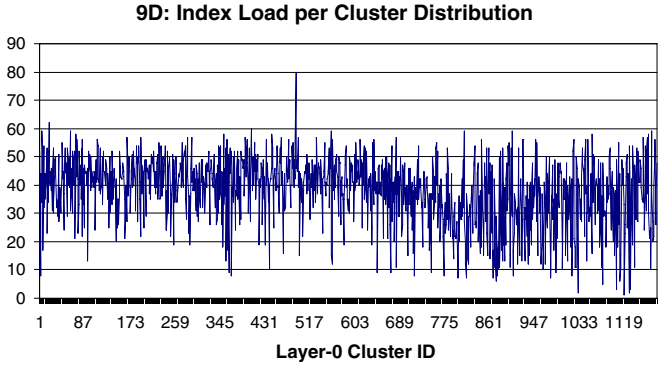

Fig. 9. 9-Dimension: the distribution of indices over all layer-0 clusters.

These two figures plot the number of indices stored at each layer-0 cluster for 3-D and 9-D, respectively. In both cases, the head of each cluster at layer-0 stores about 40 indices and never more than 90 indices. This is a tiny storage overhead if we consider the fact that there are more than 45,000 objects in the network. Similar to the study of index migration overhead, increasing the dimensionality does not affect the index load-balancing and small-overhead properties.

4.4. Search efficiency

In our evaluation study, EZSearch always attempts to return all the objects that satisfy each query (i.e., 100% precision and 100% recall). To measure search efficiency, we record the number of nodes visited by each query. Figs. 10–12 show the percentage of this number to the network size for the cases of range queries and kNN, respectively. There are 100 queries for each case (range query or kNN) and the result plotted in the figures is the average value over all these 100 queries.

4.4.1. Range queries

As expected, more nodes are visited if we increase either the range of the query or the dimensionality of the data space. This is a common problem in all multi-dimensional search techniques. However, EZSearch provides quite good results. When the query asks for 5% of the entire data

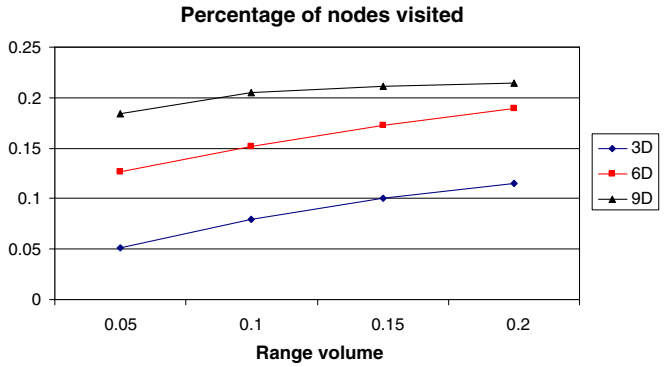

Fig. 10. Effect of volume on range queries: percentage of the network nodes that are visited during each range search.

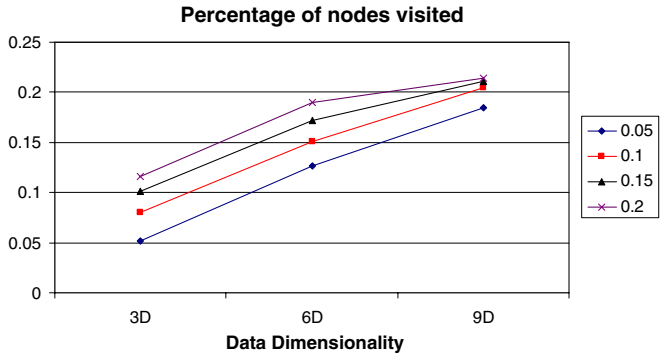

Fig. 11. Effect of dimensionality on range queries: percentage of the network nodes that are visited during each range search.

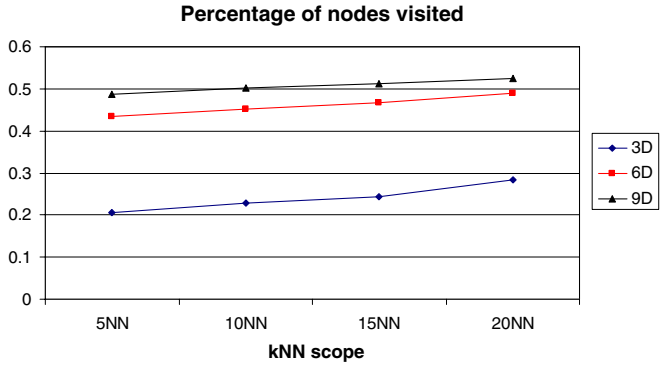

Fig. 12. Effect of k on kNN queries: percentage of the network nodes that are visited during each range search.

space, it visits only 5% of all the nodes in the network when the data dimensionality is 3, and only 18% when the dimensionality is 9. When the query asks for 20% of the data space, only 12% of the node is visited in the 3-D case and 22% in the 9-D case. This study illustrates that the search efficiency decreases only linearly with the dimensionality and the query range, rather than exponentially as in many other search techniques that suffer the curse of dimensionality problem.

4.4.2. KNN queries

The curse of dimensionality does not seem to be a severe problem either for kNN queries as illustrated by Fig. 12.

When the dimensionality is low (3-D), the search for kNN queries is quite efficient. A 5NN query visits only 20% of the nodes while a 20NN query visits less than 30% of the nodes. When the dimension is 9, a 20NN query visits approximately a half of the network. This is not as desirable as we want. This is probably because we use simplified algorithms for cluster split and merging (see Section 3.4.1 and Section 3.4.2). Enhancing these algorithms will certainly improve the efficiency of the search in general and especially for kNN queries in high-dimensionality Fig. 13.

4.5. Search time

To measure search time, we record the time it takes to process each query from the time it is posted until when all the results are returned. The search time is averaged over all 100 queries for each type. Our simulation is run on a HP Worsktation 6200 Intel Pentium 4 3Ghz CPU 1GB DRAM with Debian Linux. The simulation is centralized and the absolute search time does not reflect the true search time in a distributed setup because communication time is not included. However, in a real-world setup, we expect the communication time to be small because of the short routing path between every two nodes. Also, studying the relative differences between search times may be meaningful for implying the effect of dimensionality and query range.

Similar to the case of search overhead, increasing the range of the query also increases the search time because we search for more objects (see Fig. 14 for range queries and Fig. 15 for kNN queries). However, it is noted that increasing the dimensionality plays only a small impact on the search time (see Fig. 16 for range queries and Fig. 17 for kNN queries). This is a desirable property that shows the robustness of EZSearch under the effect of dimensionality and query range.

4.6. Comparison with DPTree

To the best of our knowledge, the hierarchical P2P search approach most related to EZSearch is DPTree [8].

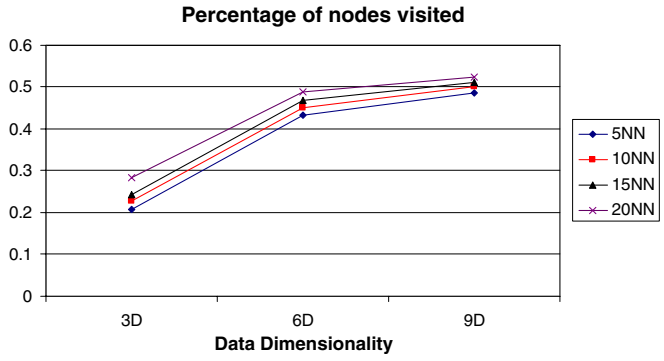

Fig. 13. Effect of dimensionality on kNN queries: Percentage of the network nodes that are visited during each range search.

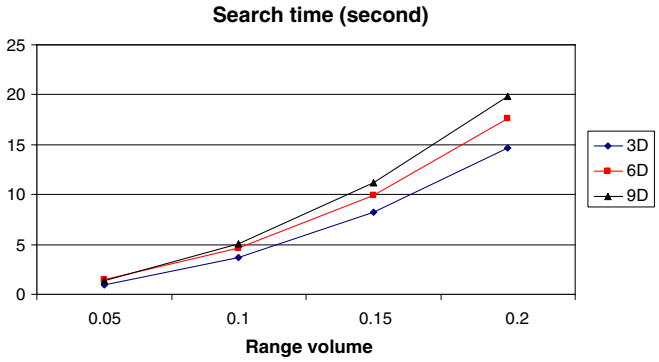

Fig. 14. Effect of volume on range queries: search time for each query.

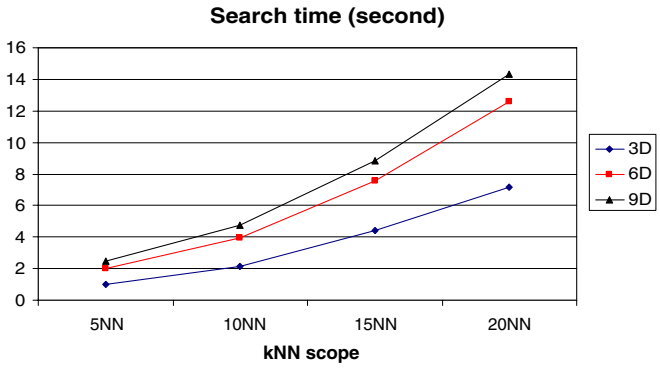

Fig. 15. Effect of k on kNN queries: search time for each query.

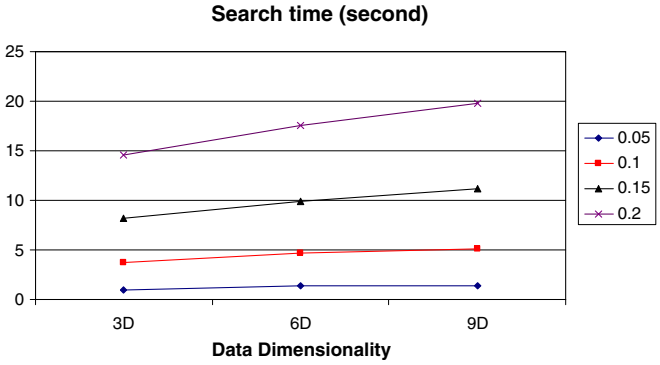

Fig. 16. Effect of dimensionality on range queries: search time for each query.

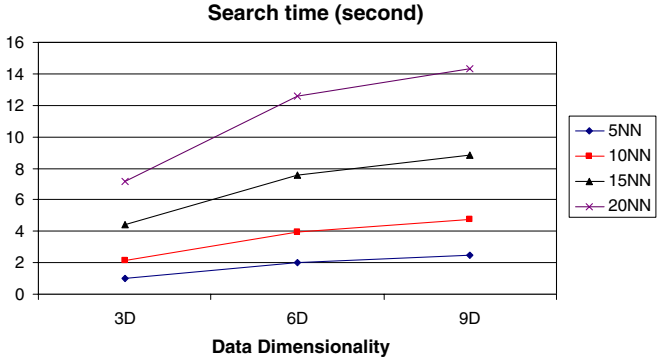

Fig. 17. Effect of dimensionality on kNN queries: search time for each query.

While EZSearch uses Zigzag, DPTree uses Skipnet [7] as the communication overlay and an R-tree-like balanced tree as the indexing architecture, mapping each node of the overlay to a branch of the index tree. A nice property of DPTree is load balancing; however, it is a complex structure with many details undisclosed. Its indexing robustness under network dynamics and its scalability with data dimensionality have yet to be evaluated. For example, DPTree was evaluated only for 2 dimensions and it remains unclear about the cost of rebuilding the index tree and migrating indices upon structural changes in the overlay. Also, DPTree's effectiveness may vary depending on its parameter setting. Therefore, it is difficult for us to compare EZSearch with DPTree experimentally on a fair basis. We will nevertheless, in our future work, try to obtain further information on DPTree for a meaningful comparison.

5. Conclusions

We have presented EZSearch – a system design for multidimensional search in P2P networks. The fundamental uniqueness of EZSearch is its use of the Zigzag hierarchy for connecting the nodes and so EZSearch inherits from Zigzag a highly efficient foundation for communication purposes. EZSearch implements an indexing architecture on top of Zigzag. We have shown that this indexing architecture is robust under the network dynamics. It distributes the index storage overhead fairly over the network nodes. Equally importantly, it allows fast range and kNN query searches with the search path length logarithmic with the network size. EZSearch keeps the search overhead reasonably small and is scalable with the query range and data dimensionality.

EZSearch can be enhanced in several ways. For example, since EZSearch is a hierarchical approach, high-layer nodes likely have to process more query load the low-layer nodes. EZSearch can be extended with a role-switch algorithm that switches the positions of high-layer nodes with low-layer nodes to achieve better fairness. Or, better algorithms for cluster split and merge events may be devised to improve the effectiveness of EZSearch's handling kNN queries in high dimensionality. We would also like to compare EZSearch with other hierarchical techniques such as DPTree [8] with real data traces.

Acknowledgements

The authors thank the US National Science Foundation for sponsoring this research (Grant CNS-0615055, PI: D. A. Tran). We are also thankful to the reviewers for their valuable comments on our work.

References

[1] F. Banaei-Kashani, C. Shahabi, SWAM: A family of access methods for similarity-search in peer-to-peer data networks, in: ACM International Conference on Information and Knowledge Management, Washington, DC, November 2004.
[2] J.L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.
[3] A.R. Bharambe, M. Agrawal, S. Seshan, Mercury: Supporting scalable multi-attribute range queries, in: ACM SIGCOMM, Portland, OR, August–September 2004.
[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making gnutella-like p2p systems scalable, in: ACM SIGCOMM, Karlsruhe, Germany, 2003.
[5] P.B. Godfrey, S. Shenker, I. Stoica, Minimizing churn in distributed systems, in: ACM Sigcomm, Pisa, Italy, December 2006.
[6] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: ACM SIGMOD Conference on Management of Data, 1984, pp. 47–57.
[7] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, A. Wolman, Skipnet: A scalable overlay network with practical locality properties, in: USENIX Symposium on Internet Technologies and Systems, Seattle, WA, March 2003.
[8] M. Li, W.-C. Lee, A. Sivasubramaniam, DPTree: A balanced tree based indexing framework for peer-to-peer networks, in: IEEE International Conference on Networking Protocols, Boston, MA, November 2006.
[9] M. Li, W.-C. Lee, A. Sivasubramaniam, D.L. Lee, A small world overlay network for semantic based search in p2p systems, in: IEEE International Conference on Network Protocols, Berlin, Germany, October 2004.
[10] A. Mondal, Yilifu, M. Kitsuregawa, P2PR-tree: An r-tree-based spatial index for peer-to-peer environments, in: ICDE/EDBT PhD Workshop, Crete, Greece, 2004.
[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in: ACM SIGCOMM, San Diego, CA, August 2001, pp. 161–172.
[12] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001, pp. 329–350.
[13] O.D. Sahin, A. Gulbeden, F. Emekci, D. Agrawal, A.E. Abbadi, PRISM: Indexing multi-dimensional data in p2p networks using reference vectors, in: ACM Multimedia Conference, Singapore, November 2005.
[14] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufman Publishers, 2006.
[15] N. Sarshar, P.O. Boykin, V.P. Roychowdhury, Percolation search in power law networks: Making unstructured peer-to-peer networks scalable, in: IEEE Conference on P2P Computing, Zurich, Switzerland, August 2004.
[16] C. Schmidt, M. Parashar, Flexible information discovery in decentralized distributed systems, in: IEEE International Symposium on High-Performance Distributed Computing, Seattle, WA, June 2003.
[17] I. Stoica, R. Morris, D. Karger, M. Kaashock, H. Balakrishman, Chord: A scalable peer-to-peer lookup protocol for internet applications, in: ACM SIGCOMM, San Diego, CA, August 2001, pp. 149–160.
[18] C. Tang, Z. Xu, S. Dwarkadas, Peer-to-peer information retrieval using self-organizing semantic overlay networks, in: ACM SIGCOMM, Karlsruhe, Germany, 2003.
[19] D.A. Tran, Hierarchical semantic overlay approach to p2p similarity search, in: USENIX Annual Technical Conference, Anaheim, CA, April 2005.
[20] D.A. Tran, K. Hua, T. Do, A peer-to-peer architecture for media streaming, IEEE Journal on Selected Areas in Communications – Special Issue on Advances in Service Overlay Networks 22 (1) (2004).
[21] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment, IEEE Journal on Selected Areas in Communications 22 (1) (2004).

Duc A. Tran is an Assistant Professor of Computer Science at the University of Massachusetts in Boston, where he supervises the Network Information Systems Lab. Dr. Tran conducts research in the areas of Computer Networks, Distributed Systems, and Multimedia Systems. His work has been funded by the NSF and the Ohio Board of Regents. Dr. Tran has served as a Vice Program Chair for IEEE AINA 2007, journal guest-editor, TPC member for 14 international conferences, and referee for numerous ACM/IEEE publications. His Ph.D. degree in Computer Science was from the University of Central Florida in 5/2003, where he also received the Distinguished Doctoral Research Award, IEEE-Orlando Outstanding Graduate Student Award, and the Order of Pegasus Award.

Thinh Nguyen has been an Assistant Professor at Oregon State University since 2004. He earned a B.S. from the University of Washington, and an M.S. and Ph.D. from U.C. Berkeley in 2000 and 2003, respectively. During 2003–2004, he was a post-doctoral research associate at Lawrence Livermore National Laboratory. During 1996–1998, he was a graphics researcher at Intel's Microcomputer Research Lab. He also spent 6 months at Microsoft, optimizing DirectX6 for Pentium III. Dr. Nguyen's current research interests include networking, signal processing, computer graphics, machine learning, data analysis and data mining.