CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2010)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.1655

Efficient range query processing over DHTs based on the balanced Kautz tree

Yiming Zhang[1,2,*,†], Ling Liu[3], Xicheng Lu[1,2] and Dongsheng Li[1,2]

[1]*National Laboratory for Parallel and Distributed Processing (PDL), Changsha 410073, People's Republic of China*
[2]*School of Computer, National University of Defense Technology, Changsha 410073, People's Republic of China*
[3]*College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.*

SUMMARY

Distributed Hash Tables (DHTs) are scalable, self-organizing, and adaptive to underlying topology changes, thus being a promising infrastructure for hosting large-scale distributed applications. The ever-wider use of DHT infrastructures has found more and more applications that require support for range queries. Recently, a number of DHT-based range query schemes have been proposed. However, most of them suffer from high query delay or imbalanced load distribution. To address these problems, in this paper we first present an efficient indexing structure called Balanced Kautz (BK) tree that uniformly maps the m-dimensional data space onto DHT nodes, and then propose a BK tree-based range query scheme called ERQ that processes range queries in a parallel fashion and guarantees to return the results in a bounded delay. In a DHT with N nodes, ERQ can answer any range of query in less than $\log N(2\log\log N+1)$ hops in a load-balanced manner, irrespective of the queried range, the whole space size, or the number of queried attributes. The effectiveness of our proposals is demonstrated through experiments. Copyright © 2010 John Wiley & Sons, Ltd.

Received 30 September 2009; Revised 4 May 2010; Accepted 16 July 2010

KEY WORDS: distributed computing; range query; distributed hash table (DHT); DLG-Kautz

1. INTRODUCTION

Over the last decade, we see an increasing trend of hosting large-scale distributed systems in networks organized by a Distributed Hash Table (DHT) [1–3] model. DHTs are scalable, self-organizing, and adaptive to underlying topology changes, thus being a promising infrastructure for realizing efficient resource discovery and sharing. The basic functionality provided by DHTs

*Correspondence to: Yiming Zhang, National Laboratory for Parallel and Distributed Processing (PDL); School of Computer, National University of Defense Technology (NUDT), Changsha 410073, People's Republic of China.
†E-mail: ymzhang@nudt.edu.cn

is exact-match query, which might be enough for some simple applications. However, the ever-wider use of DHT infrastructures has found more and more systems that require support for range queries. For example, in the Internet RAM sharing system [4] a customer might wish to find RAM providers satisfying '$\mathit{Memory} \geq 2\,\mathrm{GB}$', and in the hotel booking system a customer might issue queries like '$\$100 \leq \mathit{Rate} \leq \$200$'.

Recently, a number of range query schemes have been proposed for DHT-based systems. One important category of these is the *layered* schemes (e.g. [5–14]), which are built on top of existing DHT infrastructures and do not need to modify the topology of DHTs. The layered design has a number of methodological merits such as being easy to design, error isolation, and direct inheritance of DHTs' scalability and maintenance. However, since current layered schemes do not adapt the behaviour of underlying DHTs to the requirement of range queries, they usually have inefficient query performance and suffer from the following problems.

(1) In most proposed layered schemes, the query delay depends not only on the size of the network (i.e. the number of nodes), but also on the properties of the query (such as the queried range, the whole space size and the number of queried attributes). As a result, the unpredictable query delay would affect many aspects such as synchronization and user experience.
(2) In real environments the distribution of resource attributes is usually skewed and dynamic. For example, in the Internet RAM sharing system [4] there are more RAM providers in the range '$0 \leq \mathit{Memory} \leq 1\,\mathrm{Gbyte}$' than in the range '$2\,\mathrm{Gbyte} \leq \mathit{Memory} \leq 3\,\mathrm{Gbyte}$', and the distribution might be varied as time evolves. However, many layered schemes lack the capability of adaptive load balancing under dynamic load distribution changes.

To address these problems, in this paper we first present an efficient indexing structure called Balanced Kautz (BK) tree based on the PHT [6] technique, and then propose a BK tree-based Efficient Range Query (ERQ) scheme, guaranteeing to return the results in a *bounded delay* (defined by Li *et al.* [5]) which is relevant *only* to the size of the network. ERQ is built on top of DLG-Kautz (DK) [3], a high-performance *constant-degree* DHT. ERQ does not need to modify the underlying DK infrastructure, and thus directly inherits the desirable properties of DK such as low diameter, constant degree and efficient maintenance.

ERQ can answer any range query in less than $\log_d N(2\log_d \log_d N+1)$ hops in a load-balanced manner, where N and d are the size and base of the underlying DHT, respectively. This is very close to the asymptotic lower bound $\Omega(\log_d N)$ [5] for range queries over constant-degree DHTs. For example, in a network of 1 million ($\approx 2^{20}$) nodes with $d=4$, $2\log_d \log_d N+1 \approx 4.32$ and the query delay is guaranteed to less than $4.32\log_d N \approx 43.2$ hops.

Till date the most relevant work to ERQ is Armada [5], a delay-bounded scheme that supports range queries over the FissionE DHT [15]. The advantages of ERQ over Armada mainly include the following: (i) Compared with Armada, ERQ can achieve adaptive load balancing under dynamic load distribution changes; and (ii) ERQ can increase the base d to reduce the delay, while keeping a relatively small maintenance overhead. In contrast, Armada can only have a fixed base $d=2$.

The remainder of this paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents the detailed design of BK tree and ERQ, followed by theoretical analysis and extensive evaluations in Sections 4 and 5, respectively. Section 6 discusses related work and Section 7 concludes the paper.

2. PRELIMINARIES

In this section we first introduce the DK DHT on which the BK tree and ERQ are based, and then present an overview of our range query scheme.

2.1. *The DK DHT*

DK is a Kautz [16] graph-based DHT that was proposed in our previous work [3]. Given the average node out-degree d, DK is known to have the minimum network diameter ($\leq 2\log_d N$) among all the existing DHTs, making it a preferable infrastructure for designing layered range query schemes.

Let $Z_d = \{0, 1, 2, \ldots, d-1\}$ be an alphabet of d letters. A Kautz string ξ of length k and base d is defined as $\xi = \xi_1\xi_2\ldots\xi_k$, where $\xi_i \in Z_{d+1}$ with $1 \leq i \leq k$ and $\xi_j \neq \xi_{j+1}$ with $1 \leq j \leq k-1$. The Kautz name space $KS(d,k)$ is the set that contains all Kautz strings of base d and length k. The *Kautz graph* $K(d,k)$ is a directed graph in which each node is labelled with a Kautz string in $KS(d,k)$ and has d out-neighbors: for each $\beta \in Z_{d+1}$ and $\beta \neq u_k$, node $u = u_1u_2\ldots u_k$ has one out-edge to node $u' = u_2u_3\ldots u_k\beta$ (denoted by $u \to u'$). Figure 1(a) shows an example of Kautz graph $K(2,2)$. The routing path in a Kautz graph from node $u = u_1u_2\ldots u_k$ to node $v = v_1v_2\ldots v_k$ is: $u = u_1u_2\ldots u_k \to u_2\ldots u_kv_1 \to \cdots \to u_kv_1\ldots v_{k-1} \to v_1v_2\ldots v_k = v$.

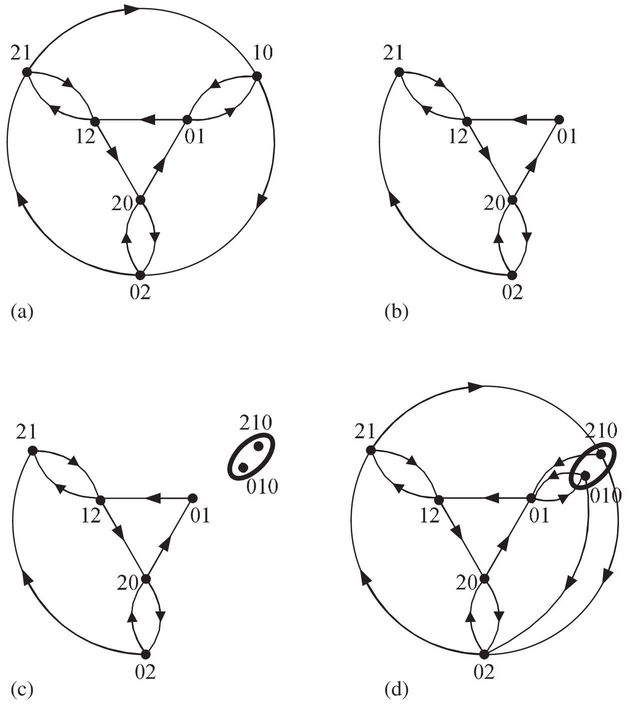


Figure 1. Kautz graph $K(2,2)$ and an example of the edge-node transition: (a) Kautz graph $K(2,2)$; (b) Delete '10' and its edges; (c) Add notes '010' and '210'; and (d) Add relevant edges.

In a DK DHT with base d, node identifiers are Kautz strings with base d and the identifier lengths might be different. Let $|u|$ denote the identifier length of node u. DK utilizes a mechanism called *edge-node transition* to deal with node joining/departure. For example, suppose that the original topology of DK is as shown in Figure 1(a). When a new node p joins, it first looks for a DK node u that satisfies $|u| \leq |v|$ for all the neighbors v of u. Let $u = 10$ in this example and the remaining processing is as follows: (i) Delete node u and all the edges of u, as shown in Figure 1(b); (ii) Each in-edge of u in the original topology ($21 \rightarrow 10$ and $01 \rightarrow 10$) turns into a node (210 for p and 010 for u), as shown in Figure 1(c); and (iii) Add in-edges and out-edges for nodes u and p, as shown in Figure 1(d).

By this means all nodes in DK dynamically form an approximate Kautz graph as shown in Figure 1(d) according to their identifiers. Each node has d out-neighbors: node $u = u_1u_2 \ldots u_m$ has one out-neighbor $u_t \ldots u_2u_3 \ldots u_m\beta (1 \leq t \leq 3)$ for each $\beta \in Z_{d+1}$ and $\beta \neq u_m$. The routing path from a node $u = u_1u_2 \ldots u_m$ to another node $v = v_1v_2 \ldots v_n$ in DK is: $u = u_1u_2 \ldots u_m \rightarrow u_ru_{r+1} \ldots u_mv_1 \rightarrow u_tu_{t+1} \ldots u_mv_1v_2 \rightarrow \ldots \rightarrow v_1v_2 \ldots v_n = v$.

2.2. Overview

We model a resource S with m attributes in terms of an m-dimensional data point. For example, consider a RAM sharing system with two attributes: the CPU frequency (*CPU*) and the memory capacity (*Memory*). All possible RAM provider instances are represented by data points which form a two-dimensional data space D, and a specific provider corresponds to a data point in D. A customer might lookup RAM providers satisfying '*CPU* $\geq$ 1 GHz &*Memory* $\geq$ 2 GB', which is a typical range query.

The main advantage provided by the layered design is that the underlying DHT eases the burden of dealing with topology maintenance and message routing. Thus, we need only to focus on the query-related issues, i.e. load-balanced mapping and delay-bounded searching.

In the mapping phase, the RAM resources are mapped onto nodes by the following operations. (i) Map the multi-attribute data points to a one-dimensional curve; and (ii) Map the curve to the nodes organized by a BK tree. For example, suppose that the RAM sharing system is deployed in a network as shown in Figure 1(a) and node 21 is a RAM provider with attributes '*Memory* = 2 GB & *CPU* = 1.5 GHz'. After this phase, the information would be published to a specific node, say node 10, together with the provider address '*ProviderID* = 21'.

In the searching phase, the query node issues a range query. The query is propagated along the BK tree in a parallel fashion, where an on-the-fly space pruning mechanism is adopted to reduce the processing cost. For example, suppose that in the network as shown in Figure 1(a) node 02 wants to lookup RAM providers satisfying '*CPU* $\geq$ 1 GHz &*Memory* $\geq$ 2 GB'. This phase would propagate the query to node 10 and return the result ('*ProviderID* = 21').

3. DESIGN

In this section we present the detailed design, including the BK tree indexing structure and the query processing algorithm ERQ, of our scheme.

3.1. BK tree-based load-balanced mapping

This subsection in turn presents (i) the linearization method of multi-attribute values, (ii) the BK tree indexing structure, and (iii) the BK tree-based load-balanced mapping algorithm.

3.1.1. Mapping data onto a Z-curve. Range queries can be classified into single-attribute queries and multi-attribute queries [5]. Considering the hash function of DK destroys the original order relationship (either '<' or '>') between attribute values, we process both two kinds of range queries in a unified way by linearizing the multi-attribute values to a one-dimensional Z-curve [17–20], the procedure of which is described as follows.

Suppose that a data point X has m attributes $X_i = x_i$ with $0 \le i < m$, and X is denoted as a vector $X = <x_0, x_1, \ldots, x_{m-1}>$. Let the entire interval of attribute X_i be $x_{i(\min)} \le X_i < x_{i(\max)}$, denoted as $X_i \in [x_{i(\min)}, x_{i(\max)})$. We first use a k-digit base d number x_i' to normalize x_i as:

$$x'_i = \left\lfloor \frac{(d^k - 1) \times (x_i - x_{i(\min)})}{(x_{i(\max)} - x_{i(\min)})} \right\rfloor. \tag{1}$$

Clearly x'_i satisfies $0 \le x'_i < d^k$. By (1) each data point x is normalized to a new point in an m-dimensional hypercube space $\{[0, d^k), [0, d^k), \ldots, [0, d^k)\}$. Therefore, without the loss of generality, in the following we would simply assume that the entire range of each dimension has been normalized into $[0, d^k)$, i.e. for each data point $X = <x_0, x_1, \ldots, x_{m-1}>$ we have $0 \le x_i < d^k$ with $0 \le i < m$.

Definition 1

Suppose that a data point X has m attributes, namely $X_i = x_i$ with $0 \le i < m$. Let x_i be a k-digit base d integer $x_{i0}x_{i1} \ldots x_{i(k-1)}$, then

$$Z(X) = x_{00}x_{10} \ldots x_{(m-1)0}x_{01}x_{11} \ldots x_{(m-1)1} \ldots x_{0(k-1)}x_{1(k-1)} \ldots x_{(m-1)(k-1)} \tag{2}$$

is called **Z-mapping** from the m-dimensional space to a one-dimensional Z-curve, and $Z(X)$ is called Z-address.

This definition can be viewed as a variation of the Z-order curve [17–20]. Clearly $Z(X)$ is a km-digit base d number and satisfies $0 \le Z(X) < d^{km}$. Then, the m-dimensional normalized space is mapped into a one-dimensional Z-curve with range $[0, d^{km})$, and by (2) a data point X is mapped to a unique integer $Z(X)$ on the Z-curve.

Let $d = 2$, $m = 2$ and $k = 3$. Figure 2 depicts an example of a Z-curve with two data points $X = <2, 4> = <010, 100>$ and $Y = <5, 6> = <101, 110>$. By (2) we have $Z(X) = 011000 = 24$ and $Z(Y) = 110110 = 54$. Clearly, the Z-address calculation is reversible, and thus the original coordinate can be recovered from its corresponding Z-address.

3.1.2. BK tree design. Many algorithms have been proposed, such as ID selection [21], virtual servers [22], and item balancing [23], to support load-balanced data mapping in DHTs. But these algorithms can neither generate the Kautz strings required by DK nor guarantee to return the results in a bounded delay. This section proposes an efficient indexing structure for addressing these problems.

Intuitively, a simple way for mapping the Z-curve onto nodes is to uniformly and statically map the Z-curve onto Kautz space and then assign the Kautz space to DK nodes. Figure 3(a) shows an example of such mapping from the Z-curve (shown in Figure 2) onto $KS(2,2)$, where the range [0,64) are equally divided into six shares and assigned to the six Kautz strings.

However, this intuitive mapping is independent of the load distribution and thus cannot handling load imbalance issues. To address this problem, we borrow the idea of dynamic

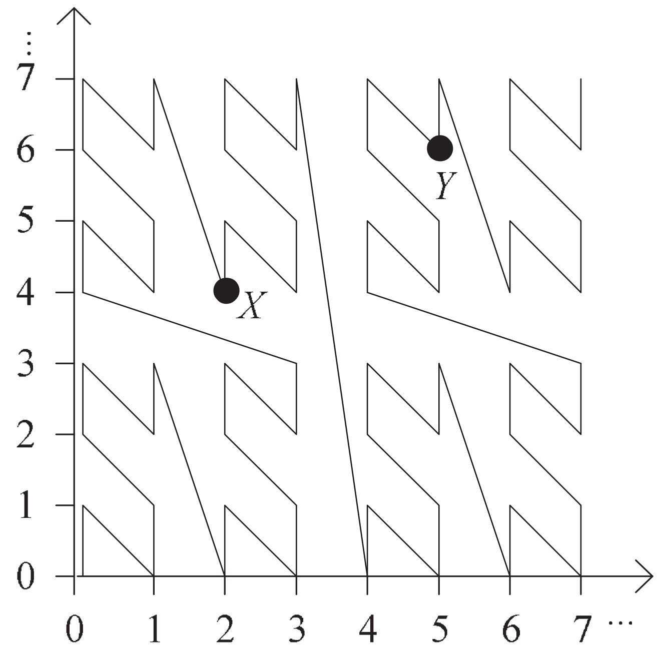


Figure 2. Example of a Z-curve.

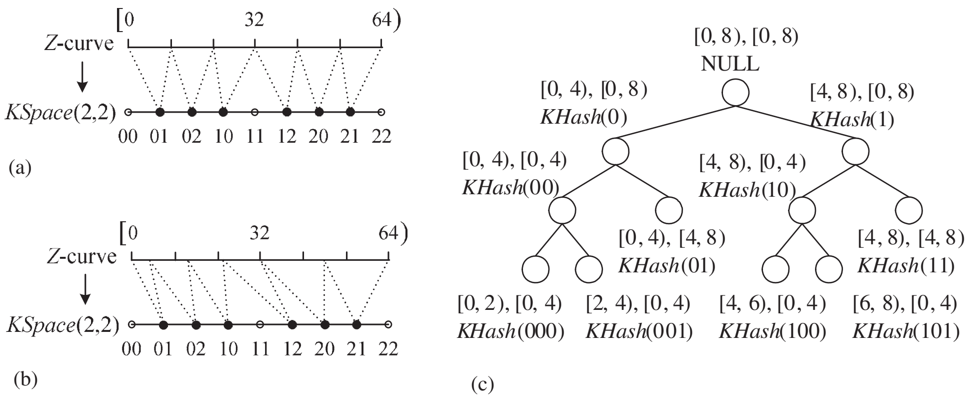


Figure 3. Examples of Z-curve mapping and BK tree: (a) static mapping; (b) dynamic mapping; and (c) example of a BK tree.

mapping from the PHT technique [6] and adopt the PHT technique to assign load among nodes in the DK network. For example, if there are more data points in the range [0,32) than in [32,64) on the Z-curve, the mapping would be dynamically adjusted and the range [0,32) is mapped onto more Kautz strings as shown in Figure 3(b). This balanced indexing structure for mapping data onto DK nodes is referred to as BK tree and is described as follows.

A node (including inner nodes and leaves) in a BK tree represents an m-dimensional space S and corresponds to a string p that is a common prefix of $Z(P)$, where P represents all possible data points in S. The node will be labelled as $s = \text{KHash}(p)$, the Kautz hash value [3] of string p.

The root at layer 0 represents the whole m-dimensional space $0 \leq X_i < d^k$ with $0 \leq i < m$, which is denoted as $[0, d^k)^m$. The root corresponds to a null string, thus being labelled as 'NULL'. Suppose that a node A at layer h corresponds to a common prefix p and is labelled as $\text{KHash}(p)$. Then A represents a multi-attribute value space that consists of all data points Y, which satisfy p is a prefix of $Z(Y)$. We also say that node A represents prefix p.

Each node in the tree has 0 or d children. If node A has d children, say nodes B_j with $0 \leq j < d$, then node B_j will correspond to the concatenation of string p and letter j (denoted as $p \circ j$) and is labelled as $\text{KHash}(p \circ j)$. Note that node B_j corresponds to a prefix one more digit than p. Let $i = h \bmod m$, then the ith-dimension of the data space represented by node A is divided into d equal shares, each of which is assigned to children B_j. The intervals of other $d-1$ attributes $X_{i'}(i' \neq i)$ represented by the children of node A are the same as that represented by A. Figure 3(c) shows an example of a 4-layer BK tree with base $d = 2$. The root represents a two-dimensional space $[0, 8)^2$. In the following we will use the term *tree node* (including *inner node* or *leaf*) to represent a node in the tree, and use the term *DK node* (*node* for short) to represent a real node in the DK DHT.

Suppose that a tree node R represents an m-dimensional data space S and the corresponding common prefix p. R is emulated by the DK node that is responsible for the label $s = \text{KHash}(p)$ according to the following policy. Let the DK node be $u = u_1 u_2 \ldots u_m$ and suppose that $s = s_1 s_2 \ldots s_n$. Define $M(u, s)$ as the maximum value of all integers $i (0 \leq i \leq \min(m, n))$ that satisfy $u_{m-i+j} = s_j$ for any $j (0 \leq j \leq i)$. For example, $M(1\mathbf{0121}, \mathbf{0121}20) = 4$, $M(10\mathbf{121}, \mathbf{121}20) = 3$. The tree node R is emulated by a DK node u, if and only if

$$M(u, s) = |u| \quad \text{or} \quad M(u, s) = |u| - 1 \wedge u_1 = s_*, \tag{3}$$

where $s_* = s_n$ if $s_n \neq u_2$, or $s_* = s_{n-1}$ if $s_n = u_2$.

Note that simple prefix (or suffix) matching policies, e.g. policies adopted in [15, 24], do not work here. We discuss the reason, as well as the correctness of our proposed policy, in the extended version of this paper [25]. In the following we will use $r(x)$ to denote the DK node that emulates the tree node with label x. E.g. suppose that the current topology of DK is as shown in Figure 1(d), and node $u = 010$, $u' = 210$, label $t = 010201$ and $s = 101202$. Then $r(t) = u$, i.e. the tree node of t is emulated by u, since $M(u, t) = 3 = |u|$; and $r(s) = u$' since $M(u', s) = 2 = |u| - 1$ and $s_n = 2 = u'_2 \wedge s_n = 2 = u'_1$.

3.1.3. Mapping Z-curve onto the BK tree.

3.1.3. Mapping Z-curve onto the BK tree. A data point is assigned to a leaf in the BK tree that represents the space containing the point. A leaf contains at most *MAX* points to limit the maximum load of a node. If a leaf, say A, contains more than *MAX* points since new points are mapped onto it, node A will generate d children as described above and divide its load to its d children according to their Z-mapping values: a point X will be assigned to the child that corresponds to a prefix of $Z(X)$. Then, A will contain no points and it will turn into an inner node. Moreover, node

A will add d links to its routing table, each of which points to a child. This algorithm (named Z-Hash) is summarized as follows.

```
Procedure Z Hash(DataPoint X)
01 $z = Z(X)$;                                            // Get the key by Z-Hash
02 Get the corresponding leaf ($A$) and string ($s$) of $z$;
03 Map $X$ onto $A$;
04 $n$ = NoOfPoints($A$);                                 // How many points in $A$
05 if ($n > MAX$) {                                       // Overflow ?
06 Generate $d$ new children for $A$;
07 Add links from $A$ to its $d$ children;
08 Get the length ($l$) of string $s$;
09     for each $Y \in$ Point ($A$) {                     // Reallocate points
10         $j = Z(Y)_{l+1}$                               // Get the ($l+1$)th digit
11         $B = s \circ j$                                // $B$ is a child of $A$
12 Move $Y$ from $A$ to $B$;
13 return;
```

All resource providers periodically invoke the procedure and a provider is considered failed when the leaf node can no longer receive any messages from its provider. If the total number of points of the d children is less than *MAX* they will move their points to their parent A and A will again become a leaf node, the procedure of which can be viewed as a counterpart of the split procedure (Lines 7–13) and is omitted here.

Note that Line 6 guarantees that there are no nodes containing more than *MAX* data points, and thus we could achieve load-balanced mapping even if there exist dynamic load distribution changes. We will further discuss the load-balancing property of the BK tree through experiments in Section 5.

3.2. Delay-bounded range query processing

Suppose that the queried range is $x_i^{(1)} \leq X_i < x_i^{(2)}$ with $0 \leq i < m$, and is denoted as $[X^{(1)}, X^{(2)})$ with $X^{(1)} = <x_i^{(1)}>$ and $X^{(2)} = <x_i^{(2)}>$. Let $Z_{\min} = Z(X^{(1)})$, $Z_{\max} = Z(X^{(2)})$, and let *ComPrefix* $(Z_{\min}, Z_{\max})$ denote the common prefix of $Z_{\min}$ and $Z_{\max}$. In order to realize delay-bounded range query processing in the BK tree, the query processing of ERQ consists of two phases, namely, (i) locating the representing node A for *ComPrefix*$(Z_{\min}, Z_{\max})$, and (ii) searching down the BK tree from A.

3.2.1. Locating the representing node. Let *ComPrefix*$(Z_{\min}, Z_{\max}) = z_1 z_2 \ldots z_p$. Then there are $p+1$ possible strings to be a prefix of $z_1 z_2 \ldots z_p$, namely, $z_1 z_2 \ldots z_p$, $z_1 z_2 \ldots z_{p-1}$, $\ldots$, z_1 and null. An intuitive method to locate the representing node is to travel a bottom-up path along the BK tree, i.e. $r(\text{KHash}(z_1 z_2 \ldots z_p)) \rightarrow r(\text{KHash}(z_1 z_2 \ldots z_{p-1})) \rightarrow \ldots$, until it finds a tree node. However, clearly in this way the tree root tends to be a performance bottleneck when there are large amounts of queries.

To address this problem, ERQ utilizes a variation of the binary search algorithm as follows. Let $a=\lceil p/d \rceil$, the query node first issues d queries to check whether some of the d DK nodes, namely, $r(\text{KHash}(z_1 z_2 \ldots z_a))$, $r(\text{KHash}(z_1 z_2 \ldots z_{2a}))$, $\ldots$, $r(\text{KHash}(z_1 z_2 \ldots z_{(d-1)a})$, $r(\text{KHash}(z_1 z_2 \ldots z_{da}))$, exist in the network. If so, the node with the maximum identifier length will be the representing node for *ComPrefix*$(Z_{\min}, Z_{\max})$ and the first phase is finished. Otherwise, Let $b=\lceil a/d \rceil$ and the query node will start a similar procedure to check whether some of the d nodes, namely, $r(\text{KHash}(z_1 z_2 \ldots z_b))$, $r(\text{KHash}(z_1 z_2 \ldots z_{2b}))$, $\ldots$, $r(\text{KHash}(z_1 z_2 \ldots z_{db}))$, exist. By parity of reasoning, this algorithm ensures that the representing node for *ComPrefix*$(Z_{\min}, Z_{\max})$ can be found in at most $\log p \le \log_d \log_d N$ steps. The above algorithm is referred to as *d-Search* and summarized as follows.

```
Procedure d-Search(Min $X^{(1)}$, Max $X^{(2)}$)
01 Get the length ($p$) of the common prefix of $Z(X^{(1)})$ and $Z(X^{(2)})$;
02 $tmp=p$;                                          // Cache the value
03 for ($i=0; i<=\log p; i++$) {                     // At most loglog$N$
04    $a=\lceil tmp/d \rceil$;                        // $d$ times smaller
05    TestSet = $\{z_1 z_2 \ldots z_a, z_1 z_2 \ldots z_{2a}, \ldots, z_1 z_2 \ldots z_{da}\}$;
06    for each string $\in$ Testset {                 // Simultaneously
07       $s=$ KHash(string);                          // Get Kautz string
08       if (($A$ = RouteTo($s$))!= NULL) {
09          return $A$; } }
10 $tmp=a$; }                                         // Prepare next loop
11 return NULL;                                       // Find nothing
```

Note that if the network bandwidth allows checking all possible strings simultaneously, ERQ can locate the representing node for *ComPrefix*$(Z_{\min}, Z_{\max})$ in one step. On the contrary, if the bandwidth is limited that ERQ can only check all possible strings one by one, this phase can be finished in at most $p \le \log_d N$ steps.

3.2.2. Parallel searching with on-the-fly pruning. After finding the representing node (say node A) for *ComPrefix*$(Z_{\min}, Z_{\max})$, if A is a leaf node, then ERQ can easily finish the processing at A. Otherwise ERQ needs to perform a top-down search along the BK tree from the inner node A.

By Corollary 1 (presented in the next section), for any subinterval $[X^{(1)}, X^{(2)})$ in the m-dimensional space the corresponding interval of the Z-mapping might be only a subset of $[Z(X^{(1)}), Z(X^{(2)}))$. Therefore, from an inner node, say node B, an on-the-fly space pruning mechanism can be conducted as follows. (i) For each child of B, say node C, check whether its represented space intersects with the queried range. (ii) If there are some intersections, then the query will be sent to child C; Otherwise C and its sub-tree will not be queried any more.

The complete algorithm for range query processing in ERQ (called *ERQ-Search*) is summarized as follows. We would discuss the delay and cost of this algorithm through theoretical analysis and experimental evaluations in Sections 4 and 5.

```
Procedure ERQ-Search(Min $X^{(1)}$, Max $X^{(2)}$)
01 $A$ = d-Search($X^{(1)}$, $X^{(2)}$);                // The first phase
02 if ($A$ == NULL) {return; }                           // Range query fails
03 if ($A$ is a leaf node) {                             // All points are here
04     Search locally return; }                          // Search locally
05 Prune($A$, $X^{(1)}$, $X^{(2)}$);                      // The 2nd phase
06 return;
Procedure Prune(Node $B$, Min $X^{(1)}$, Max $X^{(2)}$)
01 if ($B$ is a leaf node) {
02     Search locally return; }                          // Search locally
03 else {                                                // Inner node
04     Get the length (len) of Node $B$'s string;
05     for each $C \in$ Children($B$) {
06         if (Check($C$, len, $X^{(1)}$, $X^{(2)}$) ) {
07             Prune($C$, $X^{(1)}$, $X^{(2)}$); } } }   // New prune search
08 return;
```

4. ANALYSIS

This section in turn analyzes the properties of Z-mapping, query delay, and processing cost in ERQ. We first introduce the following definitions proposed by References [5, 19].

For two data points $X = <x_0, x_1 \ldots x_{m-1}>$ and $Y = <y_0, y_1, \ldots, y_{m-1}>$, we say that X is *smaller than* Y (denoted as '$X < Y$') if and only if $x_i \leq y_i$ for each $0 \leq i < m$.

Definition 2
Suppose that F is a surjection function from an m-dimensional space D to a 1-dimensional space E. F is order-preserving, if and only if for any two data points X and Y in D, if $X < Y$, then $F(X) < F(Y)$.

Definition 3
Suppose that F is an ordering–preserving function from an m-dimensional space D to a 1-dimensional space E. F is interval-preserving, if and only if for any subinterval $[X^{(1)}, X^{(2)})$ of D, the corresponding interval of F is $[F(X^{(1)}), F(X^{(2)}))$.

The following Corollary 1 presents the properties of Z-mapping.

Corollary 1
The Z-mapping is interval-preserving if $m = 1$; Otherwise the Z-mapping is order-preserving, but not interval-preserving.

Since Z-mapping is a variation of the Z-order curve, this corollary can be derived by analogy with similar Z-curve properties in References [5, 19, 20] and detailed proof is omitted here due to lack of space, which can be found in the extended version of this paper [25].

The following theorem gives the upper bound for the range query delay in ERQ.

Theorem 1
Any range queries in ERQ can be answered in less than $\log_d N(2\log_d \log_d N+1)$ hops, where N and d are the size and base of the DK DHT, respectively.

Proof
Let the queried range be $[X^{(1)}, X^{(2)})$. Let $Z_{\min}=Z(X^{(1)})$, $Z_{\max}=Z(X^{(2)})$. As described before, the d-Search algorithm can locate the tree node representing *ComPrefix*$(Z_{\min}, Z_{\max})$ within at most $\log_d p \le \log_d \log_d N$ steps.

The diameter of DK is less than $2\log_d N$ [3], thus one step corresponds to at most $2\log_d N$ DHT hops and the delay (D_1) of the first phase (locating) satisfies

$$D_1 \le 2\log_d N \log_d \log_d N. \tag{4}$$

On the other hand, since the height of a BK tree is at most $\log_d N$ and for each branch in the tree there is a link, the delay (D_2) of the second pruning phase satisfies

$$D_2 < \log_d N. \tag{5}$$

By (1) and (2), the delay (D) of any range queries in ERQ satisfies

$$D < \log_d N(2\log_d \log_d N+1). \tag{6}$$

This completes the proof. □

From this theorem it is easy to see that ERQ guarantees to return the results within a bounded delay, which is only relevant to the size of the network but independent of the queried range, the whole space size or the number of queried attributes. This delay is very close to the asymptotic lower bound $\Omega(\log_d N)$ [5] for range queries over constant-degree DHTs.

The following theorem gives the average message cost of single-attribute range query processing in ERQ.

Theorem 2
The average message cost of single-attribute range query processing in ERQ is no more than $O(\log_d N \log_d \log_d N+\beta N)$, where N and d are, respectively, the size and base of the DHT, and β represents the ratio of the queried range to the entire interval.

Proof
Let the queried range be $S=[X^{(1)}, X^{(2)})$. Let $Z_{\min}=Z(X^{(1)})$, $Z_{\max}=Z(X^{(2)})$. By Corollary 1, for $Z(X)$ and any two values X and Y in S, if $X<Y$ then $Z(X)<Z(Y)$; and for S the corresponding interval of Z is $[Z(X^{(1)}), Z(X^{(2)}))$.

Therefore, the range query for S is equal to searching a set of neighboring tree nodes, which represent a consecutive interval $[Z(X^{(1)}), Z(X^{(2)}))$.

As discussed in the proof of Theorem 1, the delay of the first phase (locating) is no more than $2\log_d N \log_d \log_d N$, and thus the cost (C_1) of the first phase satisfies

$$C_1 \le 2d \log_d N \log_d \log_d N. \tag{7}$$

Suppose that the BK tree has k layers. On average we have $k = \log_d N$. Suppose that the tree node representing *ComPrefix*($Z_{\min}$, $Z_{\max}$) is at layer h. Then in the second phase (pruning search) all the message forwarding paths can be approximated as a $k-h$ layer base-d tree, in which the number of tree nodes is equal to the cost (C_2) of the second phase. Clearly, C_2 satisfies

$$C_2 \approx 1 + d + d^2 + \cdots + d^{k-h} = \frac{d^{k-h+1}-1}{d-1}. \tag{8}$$

Since the ratio of the queried range size to the entire interval is β, on average there are βN nodes involved in representing the interval $[Z(X^{(1)}), Z(X^{(2)}))$. Thus we have

$$k - h \approx \log_d \beta N. \tag{9}$$

By (4) $\sim$ (6), it is easy to infer that the average cost is no more than $O(\log_d N \log_d \log_d N + \beta N)$ messages, and this completes the proof. □

From this theorem we conclude that the average cost of single-attribute range query processing in ERQ is close to the asymptotic lower bound $\Omega(\log N) + \beta N$ [5]. We will evaluate the average cost of multi-attribute range queries through experiments in the next section.

5. EVALUATION

5.1. Methodology

We evaluate various aspects of ERQ by modifying the DK simulator. Among the well-known layered range query schemes, only Armada [5], PlaceLab [6] and directed controlled flooding (DCF)-CAN [7] (see Section 6) can support multi-attribute range queries over *constant-degree* DHTs. Since the performance of PlaceLab is much worse than others [5], we only compare ERQ with Armada and DCF-CAN.

There are four configurable parameters involved in our experiments, namely, the network size (N), the size of the whole space (S), the ratio of the queried range to the entire interval in each dimension (β), and the number of queried attributes (m). In our experiments,

- The network size is varied from 1000 to 10 000;
- The entire interval in each dimension in the data space is [0, 1000);
- The ratio is varied from 0.01 to 0.4; and
- The number of attributes is set to 1 (for single-attribute queries) and 6 (for multi-attribute queries).

In the remainder of this section we vary these parameters one at a time, and in turn evaluate the query delay, query processing cost, BK tree maintenance overhead, and load-balancing property of ERQ. The experiment for each property is repeated at least 1000 times.

5.2. Query delay

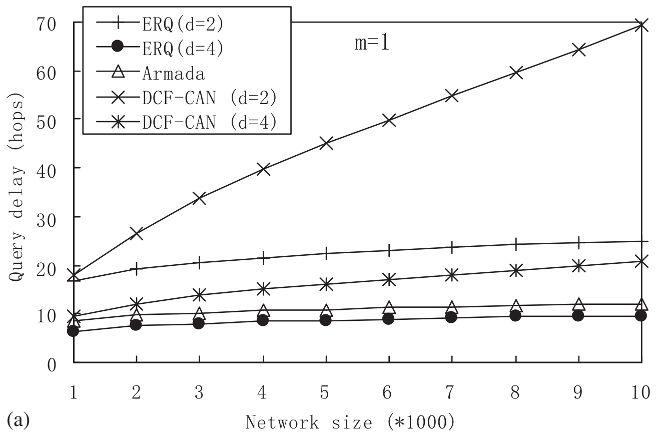
We first evaluate the average delay of single-attribute range queries in ERQ, Armada and DCF-CAN, as a function of the network size. ERQ, Armada and DCF-CAN are designed on top of DK [3], FissionE [15] and CAN [2], respectively. In these underlying DHTs, the base (d) of DK and

CAN can be set to any integer (≥ 2), while the base of FissionE can only be 2. In our experiments the base for ERQ and DCF-CAN is set to $d=2$ and $d=4$, and the base for Armada is set to $d=2$.

In each experiment we randomly select a node to initiate a range query. The queried range is randomly selected from [0, 1000) with a fixed ratio $\beta=0.02$. The network size (N) is varied from 1000 to 10 000. The results are shown in Figure 4(a).

Our conclusion for this figure is two-fold. First, the query delay of ERQ is considerably less than that of DCF-CAN when they have the same base. This is mainly because ERQ and DCF-CAN have different delay functions ($O(\log_d N \log_d \log_d N)$ and $O(N^{1/d})$, respectively) of the network size. Second, the query delay of ERQ is little greater than that of Armada when $d=2$, but ERQ outperforms Armada when $d=4$. This result proves the second advantage of ERQ over Armada (discussed in Section 1), that is, ERQ benefits from the configurable base of its underlying DK DHT, in contrast Armada can only have a fixed base $d=2$. Note that (although not shown in this figure) ERQ can adopt a larger base d to further reduce the delay, and the effect would become more pronounced for considerably large values of the network size (such as millions of nodes).

We also evaluate the average delay of multi-attribute range queries with $m=6$. The ratio in each dimension is set to $\beta=0.2$. Other parameters (N, S, and d) are the same as the previous



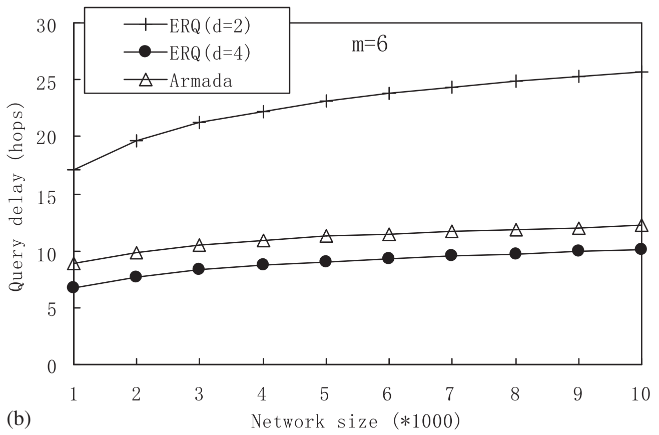


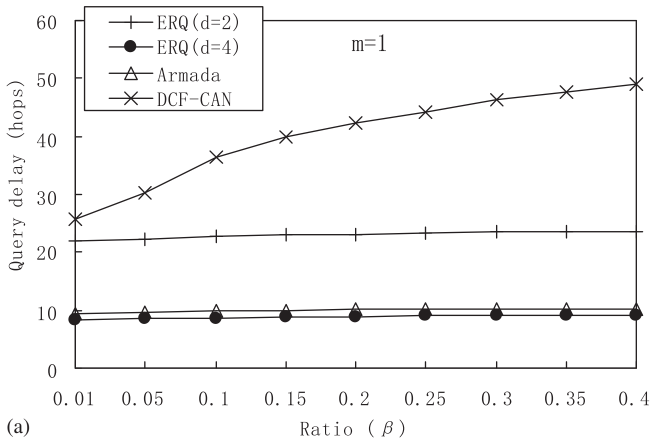
Figure 4. Average query delay of ERQ ($m=1, m=6$), Armada and DCF-CAN.

experiment. Since DCF-CAN cannot support multi-attribute range queries, only ERQ and Armada are evaluated in this experiment. The results are shown in Figure 4(b). From this figure we can deduce similar conclusions to the above-mentioned single-attribute case shown in Figure 4(a).

We then evaluate the impact of the queried range ratio ($beta$) on the single-attribute query delay in ERQ, Armada and DCF-CAN. The base for ERQ is set to $d=2$ and $d=4$, respectively. The network size (N) is fixed at 2000, and the ratio is varied from 0.01 to 0.4. The results are shown in Figure 5(a).

From this figure we observe that the query delay of ERQ almost keeps a constant regardless of the ratio (β). Thus, we conclude the queried range ratio affects little the performance of ERQ, which demonstrates ERQ's delay-bounded property. Again ERQ benefits from the configurable base and outperforms Armada when adopting base $d=4$.

We also evaluate the impact of the queried range ratio (β) on the multi-attribute query delay in ERQ and Armada. The number of attributes is set to $m=6$ and the network size (N) is fixed at 6000. Other parameters (β, S, and d) are the same as the previous experiment. The results are



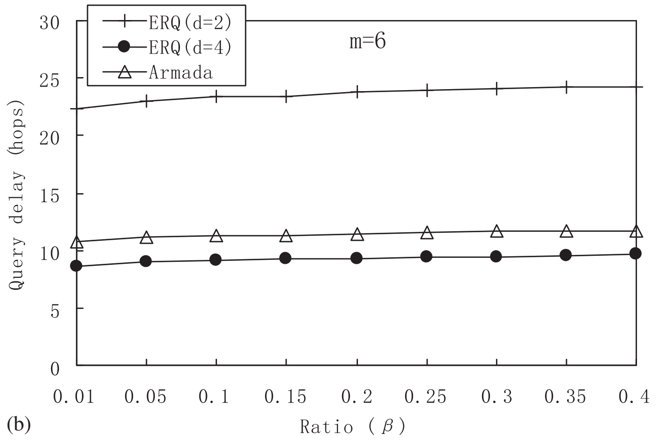


Figure 5. Impact of queried range ratio (β) on the query delay of ERQ, Armada and DCF-CAN.

shown in Figure 5(b). From this figure we can deduce similar conclusions to the single-attribute case shown in Figure 5(a).

5.3. Query processing cost

In this subsection we first evaluate the average processing cost of single-attribute query in ERQ, as a function of the network size, and compare it with Armada and DCF-CAN. The queried range ratio is set to $\beta = 0.02$. The base for ERQ is set to $d = 4$. The network size (N) is varied from 1000 to 10 000. The results are shown in Figure 6(a).

Our conclusion for this figure is two-fold. First, from this figure we observe that the average cost of ERQ increases slowly with the network size, which illustrates ERQ can efficiently processing queries with a low overhead. Second, the average cost of ERQ is always less than that of Armada and DCF-CAN. This is mainly because ERQ's BK tree has fewer levels compared with Armada's forwarding tree, which demonstrates ERQ's advantage of the configurable base d.

We also evaluate the average processing cost of multi-attribute query in ERQ and Armada. The number of attributes is set to $m = 6$ and the queried range ratio is set to $\beta = 0.2$. Other parameters

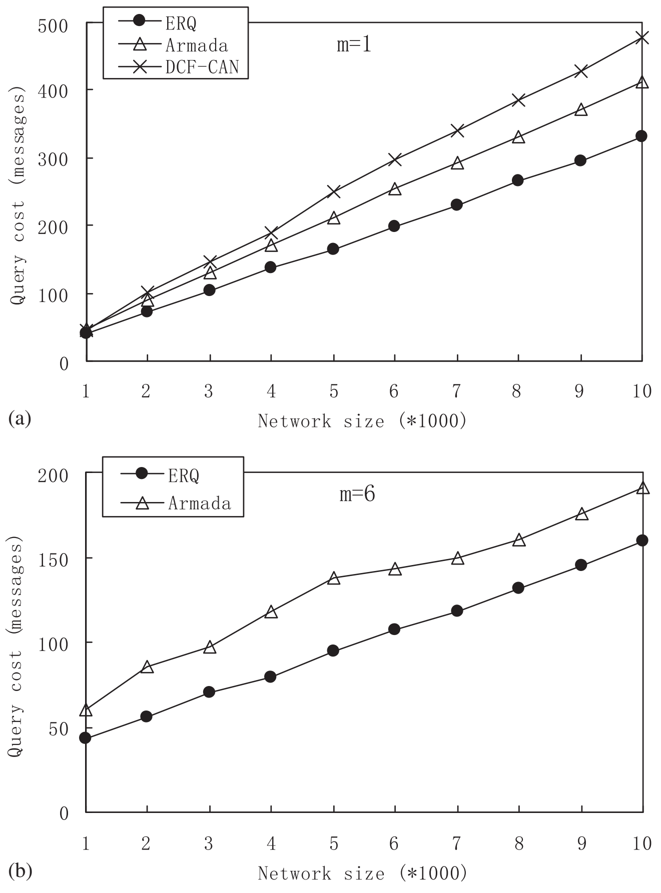


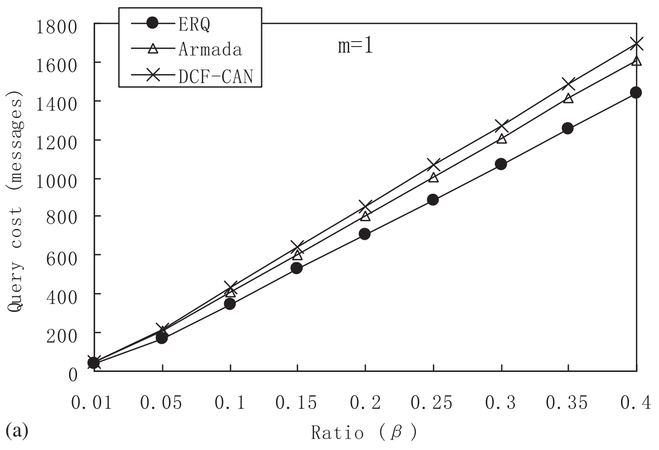
Figure 6. Average query processing cost of ERQ ($m = 1, m = 6$), Armada and DCF-CAN.

(N, S, and d) are the same as the previous experiment. The results are shown in Figure 6(b). From this figure we can deduce similar conclusions to the single-attribute case shown in Figure 6(a).

We then evaluate the impact of the queried range ratio (β) on the processing cost of single-attribute query in ERQ, Armada and DCF-CAN. The base for ERQ is set to $d=2$ and $d=4$, respectively. The network size (N) is fixed at 2000, and the ratio ($beta$) in all dimensions is simultaneously varied from 0.01 to 0.4. The results are shown in Figure 7(a).

From this figure we observe that ERQ again outperforms Armada and DCF-CAN. This is mainly due to the lower BK tree and the less number of involved nodes.

We also evaluate the impact of the queried range ratio (β) on the processing cost of multi-attribute query in ERQ and Armada. The number of attributes is set to $m=6$ and the network size (N) is fixed to 6000. Other parameters (β, S, and d) are the same as the previous experiment. The results are shown in Figure 7(b). From this figure we observe that the processing cost of ERQ increases nearly exponentially with the queried range ratio (β), which is mainly because the number of nodes involved in the queried range is exponential to β.



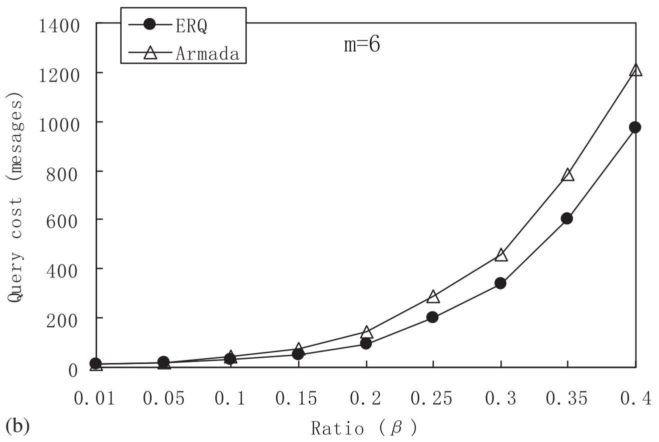


Figure 7. Impact of queried range ratio (β) on the query processing cost of ERQ, Armada and DCF-CAN.

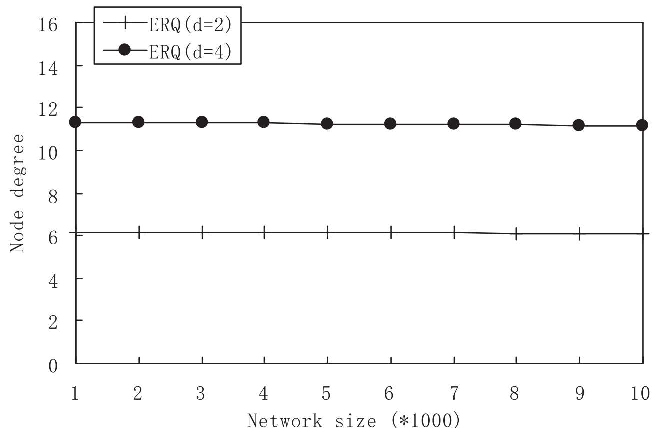


Figure 8. Average node degree of the BK tree.

5.4. *BK tree maintenance overhead*

In order to achieve load-balanced service publication, we designed the BK tree by emulating the PHT structure in DK. In this subsection we evaluate the maintenance overhead of the BK tree as a function of the network size.

As in References [1–3], we use the average node degree for the measurement of maintenance overhead. In our experiments, the base is set to $d=2$ and $d=4$, respectively, and the number of attributes is set to $m=1$. There are 600 000 random data points uniformly distributed in the entire data space [0, 1000). The network size is varied from 1000 to 10 000. The results are shown in Figure 8.

From this figure we observe that the average node degree in the BK tree almost keeps a constant, irrespective of the number of nodes in the network. The reason is that in our experiments the average degree of the underlying DK DHT and the branching factor of the BK tree are both constants, which decide the average node degree in the BK tree to be a constant. Since a node might correspond to several *tree nodes* in the BK tree, the experimental average degree is not exactly equal to a constant.

5.5. *Load balancing*

An important advantage of ERQ over Armada is that ERQ has a better load-balancing property than Armada under dynamic load distribution changes. Here the load of a node is defined as the number of data points on that node. We evaluate the dynamic load-balancing property of ERQ and compare it with Armada. In our experiments, the base and the number of attributes are set to $d=4$ and $m=1$, respectively. The network size is fixed at 2000, and there are 600 000 random data points that follow specific distribution functions as described below.

At first, all the data points are distributed in the one-dimensional space by following a uniform distribution. In ERQ the points are published onto nodes according to the Z-Hash algorithm as described in Section 3.1, while in Armada the points are published with a probability-based load-balancing mechanism called POBM by using the Newton interpolation method [26]. POBM requires prior knowledge about the load distribution to get a balanced load distribution. Experiments

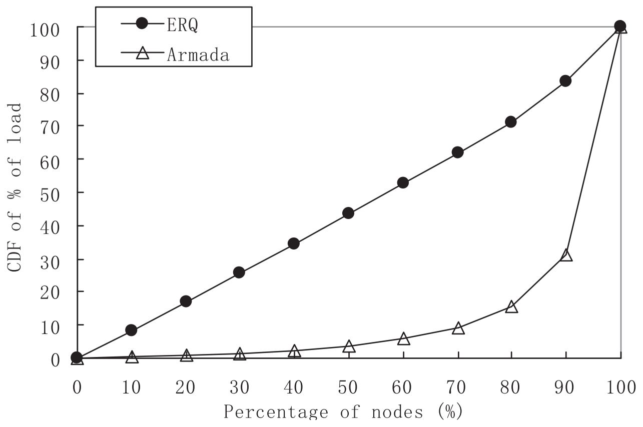


Figure 9. Load-balancing property.

show that at this point both ERQ and Armada can uniformly distribute the load, and the detailed results are omitted here due to lack of space.

Then, we change to adopt a Zipf distribution with the PDF function $\rho(x)=c\times a^{-x}$, where $a=2.5$ and c is a normalized constant. Clearly, the load becomes skewed both in ERQ and in Armada. However, ERQ can dynamically move the load between nodes and then recover to the balanced state, while Armada cannot adapt to the dynamic change of load distributions. We calculate the number of data points on each node, sort the nodes in an increasing load order, and respectively, calculate the total load of the first $10, 20, \ldots, 90$, and 100% nodes. The results are shown in Figure 9.

From this figure we can see that in ERQ the load distributes almost uniformly among the nodes, while in Armada the load distribution among nodes is obviously skewed. This result proves the first advantage of ERQ over Armada (discussed in Section 1): ERQ inherits the load-balancing property from the PHT technique even if there are dynamic load distribution changes.

6. RELATED WORK

DHT-based range query schemes could be classified into two categories, namely, *layered* schemes and *customized* schemes. The layered schemes refer to the ones that are layered over existing DHTs and do not need to modify the topology or behaviour of underlying DHTs. In contrast, the customized schemes refer to the ones that have a clean-slate design of the underlying DHTs, which are tightly coupled with their query processing methods.

6.1. Layered schemes

Squid [8] provides range query functionality based on Chord [1]. Squid uses a space-filling curve (SFC) [27] to map data points to nodes and performs range queries by searching SFC clusters recursively. The query delay of Squid is about $O(h\times\log N)$, where h is related to the depth of SFC clusters and the specific query. Gupta *et al.* [9] proposes a probabilistic scheme that uses locality sensitive hashing to support single-attribute range queries on Chord. SCRAP [10] uses the SFC to support multi-attribute range queries on Skip graph, but its query delay remains to be $O(\log N+n)$. March *et al.* [11] focuses on Chord-based multi-attribute range query processing over read-only DHTs. SkipNet [12] is a DHT that directly supports single-attribute range queries based on skip graphs [28] and has a query delay of $O(\log N+n)$, where n is the queried range size. Family tree [13] and Rainbow skip graph [14] are both constant-degree overlays that can support single-attribute range queries. Family tree combines the techniques of Viceroy [24] and SkipNet, and has an average range query delay of $O(\log N+n)$ and a delay bound of $O(\log^2 N+n)$ with high probability. Rainbow skip graph is an adaptation of the Skip graph and has a range query delay of $O(\log N)$ with high probability.

Among the well-known layered multi-attribute range query schemes, only Armada [5], PlaceLab [6], and DCF-CAN [7] are built on top of *constant-degree* DHTs.

Armada [5] is a delay-bounded range query scheme based on the FissionE DHT [15]. Armada can return the results for any range query within $2\log_2 N$ hops in a network of N nodes, with an average processing cost of $O(\log_2 N)$. As shown in Section 5, however, the fixed base $d=2$ prevents Armada from further reducing the query delay even if the bandwidth allows a larger base. Moreover, Armada utilizes historical statistical information to predict the load distribution, inevitably inducing inaccuracy problems under dynamic load changes, which are common in real Internet applications. Reference [19] proposes A-FissionE (an improvement of FissionE for handling churn) and designs MR-FissionE on top of A-FissionE. MR-FissionE is a variation of Armada and supports multi-attribute range queries with delay of $\mathrm{O}(\log^2 N)$. Armada and MR-FissionE can be viewed as the preliminary work of this paper. ERQ outperforms them in query delay, query cost, maintenance overhead and load balancing, mainly because (i) it is built based on the DK network that has the lowest diameter among all the existing DHTs, and (ii) it directly inherits the load balancing property from the PHT technique.

DCF-CAN [7] uses CAN [2] as the underlying DHT. When a node P invokes a range query $[l, u)$ in DCF-CAN, it first routes the query to the node in charge of the median value, i.e. $(l+u)/2$, and then starts two 'waves' of propagation. In the first wave, the current node propagates the query only to the neighbors that intersect the query and have a 'higher' interval than the current node. Then, the current node propagates the query to the neighbors with a 'lower' interval. The DCF mechanism can achieve a good tradeoff between query delay and overhead. DCF-CAN can only support single-attribute range queries.

Chawathe *et al.* proposed PlaceLab [6], which adopted the PHT structure to support range queries in a DHT. The PHT structure is a prefix hash tree in which leaf nodes are keys and each internal node corresponds to a distinct prefix, which is emulated by the tree. The BK tree differs from the PHT structure mainly in two aspects: (i) the BK tree node labels are Kautz strings; and (ii) the inner nodes have direct DK links to its children. PlaceLab achieves good load balancing by branching the leaves where attribute values are densely populated. However, each hop in the tree in PlaceLab corresponds to a DHT routing, and the diameter and average degree of the underlying DHT of PlaceLab are both $O(\log N)$. Thus, the query delay in PlaceLab is about $O^2(\log N)$. Recently Tang and Zhou proposed LHT [29] that redesigned the indexing scheme of PHT to reduce the maintenance overhead. LHT and PHT have similar range query delay.

6.2. *Customized schemes*

Mercury [30] and SWORD [31] support multi-attribute range queries by indexing the data set along each individual attribute. Liu *et al.* [32] propose NR-tree, which extends $R*$-tree index to support range queries and k-nearest neighbor queries in super-peer P2P systems; P-tree [33] builds specific P2P networks to support range queries based on B^+-tree. Brushwood [34] and Znet [35] modified SkipNet [12] and provide the multi-attribute range query functionality on skip graphs. Since customized range query schemes require a clean-slate design of underlying DHTs which are complicated and error-prone, our ERQ has an obvious methodological advantages over these schemes.

Aspnes *et al.* [36] and Ganesan *et al.* [37] propose mechanisms to improve the load balance of range queries. Sahin *et al.* [38] present a scheme for caching range queries. VBI-Tree [39] builds a framework based on a binary balanced tree structure to support both point queries and range queries efficiently. Skip-webs [40] improve randomized distributed data structures and present a framework for designing efficient distributed data structures for single-attribute or multi-attribute range queries. However, all the above methods are designed for customized schemes which need to make specific modifications to the underlying P2P networks.

7. CONCLUSIONS AND FUTURE WORK

This paper first presents an efficient indexing structure called BK tree that uniformly maps the m-dimensional data space onto DHT nodes, and then proposes a BK tree-based range query scheme called ERQ that processes range queries in a parallel fashion and guarantees to return the results in a bounded delay.

ERQ shows that the BK tree is an efficient indexing structure for distributed complex queries. So, in the future we would extend ERQ to support other BK tree-based complex queries, such as top-k queries, skyline queries, and nearest neighbor queries. On the other hand, in this paper ERQ utilizes the d-Search algorithm to locate the representing node for the common prefix $\mathit{ComPrefix}(Z_{\min}, Z_{\max})$. It would be interesting to develop an adaptive algorithm to automatically adjust its search patterns (one-by-one, d-Search, or fully-parallel search) according to the bandwidth limitation.

ACKNOWLEDGEMENTS

This paper is sponsored in part by the National Basic Research Program of China (973) under Grant No. 2005CB321801 and 2009CB320503, the National Natural Science Foundation of China (NSFC) under Grant No. 60903205 and 60703072, the Research Fund for the Doctoral Program of Higher Education (RFDP) under Grant No. 20094307110008, and the Foundation for the Author of National Excellent Doctoral Dissertation of PR China (FANEDD) under Grant No. 200953. The second author gives thanks for the partial support of her research by the NSF grants under the NetSE program and the CyberTrust program, an IBM SUR grant, and a grant from Intel Research Council. The authors thank the reviewers and the iVCE Group for their insightful comments. Some preliminary results of this work were presented at the ICWS '09 conference [41].

REFERENCES

1. Stoica I, Morris R, Nowell DL, Karger DR, Kaashoek MF, Dabek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking* 2003; **11**(1):17–32.
2. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. *Proceedings of the ACM SIGCOMM*, San Diego, CA, U.S.A., 2001; 149–160.

3. Zhang YM, Liu L, Li DS, Lu XC. Distributed line graphs: A universal framework for building DHTs based on arbitrary constant-degree graphs. *Proceedings of the IEEE International Conference on Distributed Computing Systems* (*ICDCS*), Beijing, China, 2008; 152–159.
4. Zhang YM, Li DS, Chu R, Xiao N, Lu XC. PIBUS: A network memory-based Peer-to-Peer IO buffering service. *Proceedings of the IFIP Networking*, Atlanta, GA, U.S.A., 2007.
5. Li DS, Cao JN, Lu XC, Chan KCC. Efficient range query processing in peer-to-peer systems. *IEEE Transactions on Knowledge and Data Engineering* 2009; **21**(1):78–91.
6. Chawathe Y, Ramabhadran S, Ratnasamy S, LaMarcay A, Shenker S, Hellersteinz J. A case study in building layered DHT applications. *Proceedings of the ACM SIGCOMM*, Philadelphia, PA, U.S.A., 2005; 97–108.
7. Andrzejak A, Xu ZC. Scalable efficient range queries for grid information services. *Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing* (*P2P*), Linköping, Sweden, 2002.
8. Schmidt C, Parashar M. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing* 2004; **8**(3):19–26.
9. Gupta A, Agrawal D, Abbadi AE. Approximate range selection queries in Peer-to-Peer systems. *Proceedings of the 1st Biennial Conference Innovative Data Systems Research* (*CIDR*), Asilomar, CA, U.S.A., 2003.
10. Ganesan P, Yang B, Garcia-Molina H. One torus to rule them all: Multidimensional queries in P2P systems. *Proceedings of the 7th International Workshop on Web and Databases* (*WebDB*), Maison de la chimie, Paris, France, 2004.
11. March V, Teo YM. Multi-attribute range queries on read-only DHT. *Proceedings of the 15th International Conference on Computer Communications and Networks* (*ICCCN*), Arlington, VA, U.S.A., 2006.
12. Harvey NJA, Jone MB, Saroiu S, Theimer M, Wolman A. SkipNet: A scalable overlay network with practical locality properties. *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems* (*USITS*), Seattle, WA, U.S.A., 2003.
13. Zatloukal KC, Harvey NJA. Family trees: An ordered dictionary with optimal congestion, locality, degree, and search time. *Proceedings of the 15th Anniversary ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), New Orleans, LA, U.S.A., 2004.
14. Goodrich MT, Nelson MJ, Sun JZ. The rainbow skip graph: A fault-tolerant constant-degree distributed data structure. *Proceedings of the 16th Anniversary ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), Miami, FL, U.S.A., 2006.
15. Li DS, Lu XC, Wu J. FissionE: A scalable constant degree and low congestion DHT scheme based on Kautz graphs. *Proceedings of the IEEE INFOCOM*, Miami, FL, U.S.A., 2005; 1677–1688.
16. Fiol MA, Yebra JLA, de Miquel IA. Line digraph iterations and the (d,k) digraph problem. *IEEE Transactions on Computers* 1984; **33**(2):400–403.
17. Jagadish HV. Linear clustering of objects with multiple attributes. *Proceedings of the ACM SIGMOD*, Atlantic City, NJ, U.S.A., 1990; 332–342.
18. Orenstein JA, Merrett TH. A class of data structures for associative searching. *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (*PODS*), Waterloo, Ontario, Canada, 1984.
19. Zhang YM, Li DS, Lu XC. Scalable distributed resource information service for internet-based virtual computing environment. *Journal of Software*, 2007; **18**(8):1933–1942 (in Chinese).
20. Lee KCK, Zheng B, Li H, Le W-C. Approaching the skyline in Z order. *Proceedings of the 33rd International Conference on Very Large Data Bases* (*VLDB*), Vienna, Austria, 2007.
21. Manku GS. Balanced binary trees for ID management and load balance in distributed hash tables. *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing* (*PODC*), St. John's, Newfoundland, Canada, 2004.
22. Godfrey PB, Stoica I. Heterogeneity and load balance in distributed hash tables. *Proceedings of the IEEE INFOCOM*, Miami, FL, U.S.A., 2005.
23. Karger D, Ruhl M. Simple efficient load balancing algorithms for peer-to-peer systems. *Theory of Computing Systems* 2006; **39**(10):787–804.
24. Malkhi D, Naor M, Ratajczak D. Viceroy: A scalable and dynamic emulation of the butterfly. *Proceedings of the ACM Symposium on Principles of Distributed Computing* (*PODC*), 2002.
25. Malkhi D, Naor M, Ratajczak D. Viceroy: A scalable and dynamic emulation of the butterfly. *Proceedings of the ACM Symposium on Principles of Distributed Computing* (*PODC*), 2002.
25. Zhang YM. Efficient range query processing over DLG-Kautz. *Technical Report NUDT-CS-20100725*. Available at: http://kylink.com/Papers/BKT.pdf.
26. Schatzman M. *Numerical Analysis*: *A Mathematical Introduction*. Clarendon Press: Oxford, 2002.

27. Asano T, Ranjan D, Roos T, Welzl E, Widmaier P. Space filling curves and their use in geometric data structures. *Theoretical Computer Science* 1997; **181**(1):3–15.
28. Aspnes J, Shah G. Skip graphs. *Proceedings of the 14th Anniversary ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), Baltimore, MD, U.S.A., 2003.
29. Tang Y, Zhou S. LHT: A low-maintenance indexing scheme over DHTs. *Proceedings of the IEEE International Conference on Distributed Computing Systems* (*ICDCS*), Beijing, China, 2008; 141–151.
30. Bharambe AR, Agrawal M, Seshan S. Mercury: Supporting scalable multi-attribute range queries. *Proceedings of the ACM SIGCOMM*, Portland, OR, U.S.A., 2004; 353–366.
31. Oppenheimer D, Albrecht J, Patterson D, Vahdat A. Distributed resource discovery on Planetlab with SWORD. *Proceedings of the 1st Workshop on Real*, *Large Distributed Systems* (*WORLDS*), San Francisco, CA, U.S.A., 2004.
32. Liu B, Lee WC, Lee DL. Supporting complex multi-dimensional queries in P2P systems. *Proceedings of the IEEE International Conference on Distributed Computing Systems* (*ICDCS*), Columbus, OH, U.S.A., 2005; 155–164.
33. Crainiceanu A, Linga P, Gehrke J, Shanmugasundaram J. P-Tree: A P2P index for resource discovery applications. *Proceedings of the ACM WWW*, New York, NY, U.S.A., 2004; 390–391.
34. Zhang C, Krishnamurthy A, Wang RY. Brushwood: Distributed trees in Peer-to-Peer systems. *Proceedings of the 4th International Workshop on Peer-to-Peer Systems* (*IPTPS*), Ithaca, NY, U.S.A., 2005.
35. Shu Y, Ooi BC, Tan KL, Zhou A. Supporting multi-dimensional range queries in Peer-to-Peer systems. *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing* (*P2P*), Konstanz, Germany, 2005.
36. Aspnes J, Kirsch J, Krishnamurthy A. Load balancing and locality in range-queriable data structures. *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing* (*PODC*), St. John's, Newfoundland, Canada, 2004.
37. Ganesan P, Bawa M, Garcia-Molina H. Online balancing of range-partitioned data with applications to Peer-to-Peer systems. *Proceedings of the 30th International Conference on Very Large Data Bases* (*VLDB*), Toronto, Canada, 2004.
38. Sahin OD, Gupta A, Agrawal D, Abbadi AE. A Peer-to-Peer framework for caching range queries. *Proceedings of the 20th IEEE International Conference on Data Engineering* (*ICDE*), Boston, MA, U.S.A., 2004.
39. Jagadish HV, Ooi BC, Vu QH, Zhang R, Zhou A. VBI-Tree: A Peer-to-Peer framework for supporting multi-dimensional indexing schemes. *Proceedings of the 22nd IEEE International Conference on Data Engineering* (*ICDE*), Atlanta, GA, U.S.A., 2006.
40. Arge L, Eppstein D, Goodrich MT. Skip-webs: Efficient distributed data structures for multi-dimensional data sets. *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing* (*PODC*), Las Vegas, NV, U.S.A., 2005.
41. Zhang YM, Liu L, Li DS, Liu F, Lu XC. DHT-based range query processing for Web service discovery. *Proceedings of the IEEE International Conference on Web Services* (*ICWS*), Los Angels, CA, U.S.A., 2009.