# DHT-Based Range Query Processing for Web Service Discovery

Yiming Zhang[1], Ling Liu[2], Dongsheng Li[1], Xicheng Lu[1]
[1]*PDL, School of Computer, National University of Defense Technology, Changsha, 410073 China*
[1]*{ymzhang, dsli, xclu}@nudt.edu.cn*
[2]*College of Computing, Georgia Institute of Technology, Atlanta, 30332-0765 USA*
[2]*lingliu@cc.gatech.edu*

## Abstract

*DHTs are scalable, self-organizing, and adaptive to underlying topology changes, thus being a promising infrastructure for realizing efficient Web service discovery. Range queries play an important role in service discovery, and in recent years a number of DHT-based range query schemes have been proposed. However, most of them suffer from high query delay and high processing cost. This paper presents ERQ, an Efficient scheme for delay-bounded Range Query processing over DHTs. We first design a balanced Kautz (BK) tree to uniformly map the m-dimensional data space onto DHT nodes, and then present a novel algorithm that processes range queries in a parallel fashion, where an on-the-fly space pruning mechanism is adopted to reduce the processing cost. In a DHT with N nodes, ERQ can answer any multi-attribute range query in less than* $\log N(2\log\log N+1)$ *hops with low processing cost, irrespective of the queried range, the whole space size, or the number of queried attributes. The effectiveness of ERQ is demonstrated through extensive experiments.*

## 1. Introduction

Over the last decade, we see an increasing trend of hosting large number of Web services in decentralized networks organized by a distributed hash table (a.k.a. DHT [1-3]) model. DHTs are scalable, self-organizing, and adaptive to underlying topology changes, thus being a promising infrastructure for realizing efficient discovery of various Web services [4-6]. The basic functionality provided by DHTs is exact-match query, which might be enough in some simple applications. For example, a file sharing service [4] may use filenames as keywords for service publication and discovery. However, the simple exact-match interface is not flexible enough for many more complicated services. For example, in an Internet RAM service [6] a customer might wish to find RAM providers satisfying "*Memory* ≥ 2GB", and in a reputation management service a user might issue queries like "60 ≤ *Reputation Score* ≤ 80", and so on.

The above illustrated queries are called *range queries*, which play an important role in Web service discovery. Recently, a number of DHT-based schemes for range query processing have been proposed. One important category of DHT-based range query schemes is the *layered* schemes (e.g., [7-15]), which are built on top of existing DHTs and do not need to modify the underlying infrastructure, thus having a number of methodological merits such as being easy to design and error isolation. However, current layered schemes suffer from inefficient performance since they do not adapt the behavior of underlying DHTs to the requirement of range queries. In most proposed layered schemes, the query delay depends on both the size of the network (i.e. the number of nodes) and the properties of the query (such as the queried range, the whole space size and the number of queried attributes). As a result, these schemes cannot guarantee to return the results in a *bounded delay* (defined by [12]) that is relevant *only* to the size of the network.

In this paper we present ERQ, an Efficient layered scheme for delay-bounded Range Query processing over DHTs. 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 low congestion. ERQ supports efficient range queries and can return all the results in a bounded delay, irrespective of the queried range, the whole space size, or the number of queried attributes.

The contribution of this paper is three-fold.

- We propose the balanced Kautz (BK) tree model to uniformly map the *m*-dimensional data space onto the network via a 1-dimensional Z-curve.
- We present a parallel query processing algorithm that searches along the BK tree with an on-the-fly space pruning mechanism.
- We theoretically analyze the delay and cost of ERQ, and demonstrate the effectiveness of our proposals through extensive experiments.

ERQ can answer any range query in less than $\log_d N(2\log_d\log_d N+1)$ hops with low processing cost, where $N$ and $d$ are the size and base of the underlying DHT, respectively. This is very close to the asymptotic lower bound $\Omega(\log_d N)$ [12] for range queries over constant-degree DHTs.

Up to now the most relevant work to ERQ is Armada [12], a novel delay-bounded scheme that supports range queries over the FissionE DHT [20]. The advantages of ERQ over Armada mainly include: (i) ERQ has a better load balancing property than Armada under dynamic load distribution changes, which are very common in real Internet environments; and (ii) ERQ can increase the base $d$ to reduce the delay, while keeping a relatively small routing table size ($2d$) and maintenance overhead. In contrast, Armada can only have a fixed base $d = 2$.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents the detailed design of 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

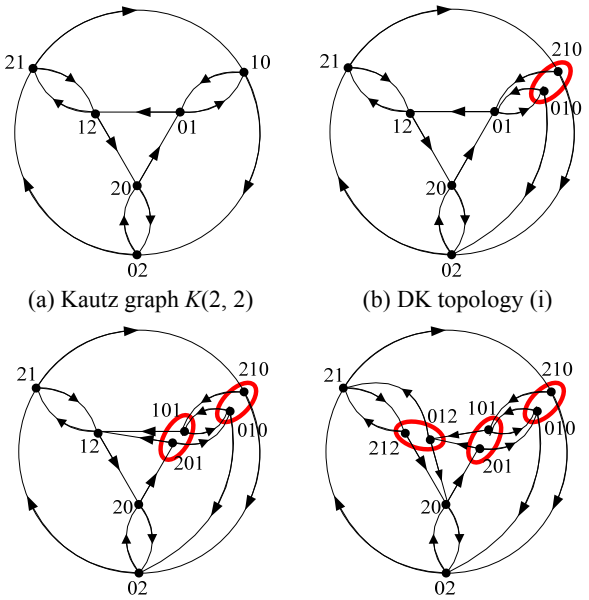
In this section we first introduce the background of DK on which ERQ is based, and then present an overview of ERQ.

### 2.1. Background: the DK DHT

DK is a Kautz graph-based DHT that was proposed in our previous work [3]. Let $Z_d = \{0, 1, 2, \ldots, d-1\}$ be an alphabet of $d$ letters. A *Kautz string* $a$ of length $k$ and base $d$ is defined as $a = a_1a_2\ldots a_k$, where $a_i \in Z_{d+1}$ with $1 \le i \le k$ and $a_j \ne a_{j+1}$ with $1 \le j \le k-1$. 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 labeled with a Kautz string in $KS(d, k)$ and has $d$ out-neighbors: for each $\beta \in Z_{d+1}$ and $\beta \ne u_k$, node $u = u_1u_2\ldots u_k$ has one out-edge to node $v = v_2v_3\ldots v_k\beta$ (denoted by $u \to v$). 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:

$$u = u_1u_2...u_k \to u_2...u_kv_1 \to \mathrm{L} \to v_1v_2...v_k = v.$$

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 current 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 $|u| \le |v|$ for all the neighbors $v$ of $u$. Let $u$ = 10 in this example. Then the edge from 21 to 10 will turn into node 210 that corresponds to node $p$, and the edge from 01 to 10 will turn into node 010 that corresponds to node $u$. The new topology is shown in Figure 1(b). By this means DK would get a series of topologies, the first three of which are illustrated in Figures 1(b), (c), and (d). All nodes in DK are organized into an approximate Kautz graph according to their identifiers. Each node has $d$ out-neighbors: node $u = u_1u_2\ldots u_m$ has one out-neighbor $u_t\ldots u_2u_3\ldots u_m\beta$ ($1 \le t \le 3$) for each $\beta \ne u_m$.




(a) Kautz graph $K(2, 2)$ (b) DK topology (i)

(c) DK topology (ii) (d) DK topology (iii)

Figure 1. **Kautz graph $K(2,1)$ and DK topologies.**

### 2.2. ERQ overview

To facilitate Web service look up and discovery, we model a service in terms of the set of attributes to specify the properties of the service. For example, consider a computing service with two attributes: the CPU frequency (*CPU*) and the memory capacity (*Memory*). All possible service instances form a two-dimensional data space and a specific service corresponds to a data point in the space. A service customer might lookup services satisfying "*CPU* ≥ 1GHz & *Memory* ≥ 2GB", which is a typical range query.

DHTs are a promising infrastructure for realizing efficient Web service discovery. However, most DHTs cannot directly support such range queries since their hash functions destroy the original order relationship (either "<" or ">") between attribute values. This paper addresses the problem and proposes an Efficient Range Query scheme (ERQ) that is layered on top of the DK DHT.

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, ERQ needs only to focus on the range query-related issues, namely, publishing and searching.

In the publishing phase, the services are published by the following operations. (i) Map the multi-attribute data points to a 1-dimensional Z-curve; and (ii) Map the Z-curve to the nodes organized by a BK tree. For example,

suppose that the computing service is deployed in a network as shown in Figure 1(a) and node 21 is a provider with attributes "*Memory* = 3GB & *CPU* = 1.5GHz". After the publishing phase, this information would be published to its responsible node, say node 10, together with the provider address "*ProviderID* = 21".

In the searching phase, the query node issues a range query and ERQ propagates the query 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 look up services satisfying "*CPU* ≥ 1GHz & *Memory* ≥ 2GB". The searching phase would propagate the query to node 10 and return the result ("*ProviderID* = 21").

## 3. Design

This section presents the detailed design, including service publication and query processing, of ERQ.

### 3.1. Service publication

This subsection first discusses the linearization of multi-attribute values, then presents the balanced Kautz tree, and at last proposes the service publication algorithm.

**3.1.1. Linearization of multi-attribute values.** Range queries can be classified into single-attribute queries and multi-attribute queries [12]. ERQ processes both single-attribute and multi-attribute queries in a unified way by linearizing the multi-attribute values to a one-dimensional Z-curve [21].

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

$$x_i' = \frac{(d^k - 1) \times (x_i - x_{i(\min)})}{(x_{i(\max)} - x_{i(\min)})}. \quad (1)$$

Clearly $x_i'$ satisfies $0 \le x_i' < d^k$. For simplicity, in the following we will not distinguish $x_i$ and $x_i'$, and assume that all attributes have the same entire interval $[0, d^k)$.

DEFINITION 1. **Z-mapping** [22,23]**.** 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 $x_{i0}x_{i1}...x_{ik}$, then

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

is called *Z-mapping* from the $m$-dimensional space to a one-dimensional Z-curve.

Let $n$ represent the length of $Z(X)$. Clearly $n = k \times m$ and $0 \le Z(X) < d^n$. Then, a data point $X$ is mapped by (2) to a unique $n$-digit integer $Z(X)$ on the Z-curve.

**3.1.2. Balanced Kautz tree.** This subsection designs the BK tree by emulating the PHT structure [13] in DK.

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

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

Each node in the tree has 0 or $d$ children. If node $A$ has $d$ children, say nodes $B_j$ with $0 \le j < d$, then node $B_j$ will correspond to the concatenation of string $s$ and letter $j$ (denoted as $s \circ j$) and is labeled as $KHash(s \circ j)$. Note that node $B_j$ corresponds to a prefix one more digit than $s$. Let $i = h \bmod m$, then the $i$th-dimension of the data space represented by node $A$ is divided into $d$ equal shares, each of which is assigned to a children $B_j$. The intervals of other $d-1$ attributes $X_{i'}$ ($i' \ne i$) represented by the children of node $A$ are the same as that represented by $A$.

In the following we will use the term *tree node* (*inner node* or *leaf node*) to represent a node in the BK 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 $s$. $R$ is emulated by the DK node that is responsible for the label ($KHash(s)$) according to the following policy [3].

Let the DK node be $u = u_1u_2...u_m$ and the label $s = s_1s_2...s_n$. Define $M(u,s)$ as the maximum value of all integers $i$ ($0 \le i \le \min(m, n)$) that satisfy $u_{m-i+j} = s_j$ for any $j$ ($0 \le j \le i$). E.g., $M(10\underline{\mathbf{0121}}, \underline{\mathbf{0121}}20) = 4$, $M(10\underline{\mathbf{121}}, \underline{\mathbf{121}}20) = 3$. The tree node $R$ is emulated by a DK node $u$, iff

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

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

In the following we will use $r(x)$ to denote the DK node that emulates the tree node with label $x$. For example, suppose that the current topology of DK is as shown in Figure 1(b), 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. Publication in 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 node, say node $A$, contains more than *MAX* points since new points are published 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.

All service providers periodically invoke the procedure and a service is considered stopped when the leaf node can no longer receive any messages from its provider. Note that 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 and is omitted here.

## 3.2. Range query processing

Suppose that the queried range is $x_i^{(1)} \le X_i < x_i^{(2)}$ with $0 \le i < m$, which 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 $Z_{\max}$. To reduce the search space, ERQ's query processing 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* ($Z_{\min}$, $Z_{\max}$) = $z_1z_2...z_p$. Then there are $p$+1 possible strings to be a prefix of $z_1z_2...z_p$, namely, $z_1z_2...z_p$, $z_1z_2...z_{p-1}$, …, $z_1$ and null. An intuitive method to locate the representing node is to travel a top-down path along the BK tree: root $\rightarrow$ $r(KHash(z_1))$ $\rightarrow$ $r(KHash(z_1z_2))$ $\rightarrow$ …, until the current node is $r(KHash(z_1z_2...z_p))$ or a leaf. 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, $r(KHash(z_1z_2...z_a))$, $r(KHash(z_1z_2...z_{2a}))$, …, $r(KHash(z_1z_2...z_{da}))$, 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, $r(KHash(z_1z_2...z_b))$, $r(KHash(z_1z_2...z_{2b}))$, etc., 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 algorithm for locating representing node is referred to as *d-Search*.

Note that if the bandwidth allows checking all possible strings simultaneously, ERQ can locate the representing node in one step. Similarly, 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.** If the representing node (say node $A$) for *ComPrefix*($Z_{\min}$, $Z_{\max}$) 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 $A$.

As shown in Theorem 1 (presented in the next section), for 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.

```
Procedure Parallel-Search (Value X(1), Value X(2))
01 A = d-Search (X(1), X(2));
02 if (A==NULL) {return; }
03 if (A is a leaf node) {
04     LocalSearch (X(1), X(2)); return; }
05 Prune (A, X(1), X(2));
06 return;

Procedure Prune (Node B, Value X(1), Value X(2))
01 if (B is a leaf node) {
02     LocalSearch (X(1), X(2)); return; }
03 else {
04     s = GetString (B); len = Length (s);
05     for each C ∈ Children (B) {
06         if (Check(C, len, X(1), X(2)) ) {
07             Prune (C, X(1), X(2)); } } }
08 return;
```

Figure 2. **Range query processing and on-the-fly space pruning.**

The complete algorithm for range query processing in ERQ (named *Parallel-Search*) is summarized in Figure 2, together with the space pruning algrithm (named *Prune*).

# 4. Analysis

This section analyzes the properties of query delay and processing cost in ERQ.

For two data points $X = <x_i>$ and $Y = <y_i>$ with $0 \le i < m$, we say that $X$ is *smaller than* $Y$ (denoted as "$X < Y$") if there exists an integer $t$ ($0 \le t < m$) satisfying: for each $j$ ($0 \le j < t$), $x_j = y_j$ and $x_t < y_t$.

The following Theorem 1 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 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 in Section 3.2.1, the $d$-division 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.

According to [5], the diameter of DK is less than $2\log_d N$, thus one step corresponds to at most $2\log_d N$ DHT hops and the delay ($D_1$) of the first phase (locating) satisfies

$$D_1 \le 2\log_d N \log_d \log_d N . \quad (4)$$

On the other hand, since the height of ZK 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

$$D_2 < \log_d N . \quad (5)$$

By (4) and (5), the delay ($D$) of any range queries in ERQ satisfies

$$D < \log_d N(2\log_d \log_d N+1) . \quad (6)$$

Therefore, Theorem 1 holds. ■

From Theorem 1 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 $O(\log_d N)$ for range queries over constant-degree DHTs.

The following Theorem 2 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 $\beta$ 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 the definition of Z-mapping, 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

$$C_1 \le 2d \log_d N \log_d \log_d N . \quad (7)$$

Suppose that the ZK 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

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

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

$$k - h \approx \log_d \beta N . \quad (9)$$

By (7) ~ (9), 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 Theorem 2 we conclude that the average cost of single-attribute range query processing in ERQ is close to the asymptotic lower bound $O(\log N)+\beta N$ [12]. We will evaluate the average cost of multi-attribute range queries through experiments in the next section.

## 5. Evaluation

This section evaluates ERQ by extensive experiments.

### 5.1. Methodology

We evaluate the properties of ERQ by modifying the DK simulator [3]. Among the well-known layered range query schemes, only Armada [12], PlaceLab [13] and DCF-CAN [14] (see Section 6) can support range queries over *constant-degree* DHTs. Since the performance of PlaceLab is much worse than others, we only compare ERQ with Armada and DCF-CAN in this section.

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 ($\beta$), and the number of queried attributes ($m$). In our experiments,

- The network size is varied from 1K to 10K;
- The entire interval in each dimension in the data space is [0, 1000);
- The ratio is varied from 0.05 to 0.4; and
- The number of queried attributes is set to 1 (single-attribute) and 6 (multi-attribute).

In the rest of this section we vary these parameters one at a time, and in turn evaluate the query delay and processing cost 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 (as a function of the network size) in ERQ, Armada and DCF-CAN, which are designed on top of DK [3], FissionE [20] and CAN [2] respectively. In the three underlying DHTs, the base ($d$) of DK and CAN can be set to any integer ($\ge 2$), while the base of FissionE can only be 2. Since all the three DHTs have the same

average degree $2d$, we can say that they have the same routing table size if their base is equal. In our experiments the base of DK and CAN is set to $d = 2$ and $d = 4$, and the base of FissionE 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 $\beta = 0.2$. The results are shown in Figure 3.

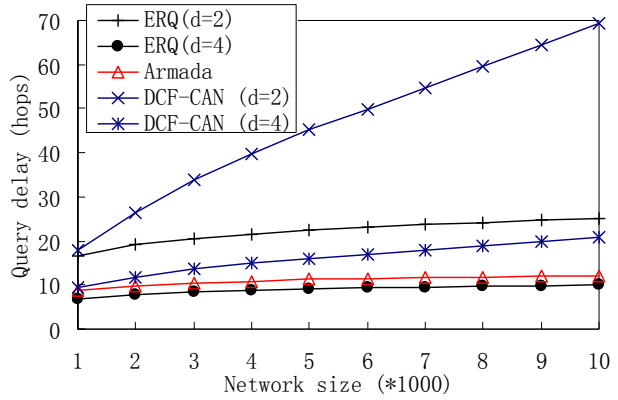



Figure 3. **Average query delay (m = 1).**

Our conclusion for Figure 3 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 a 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 (for example, millions of nodes).

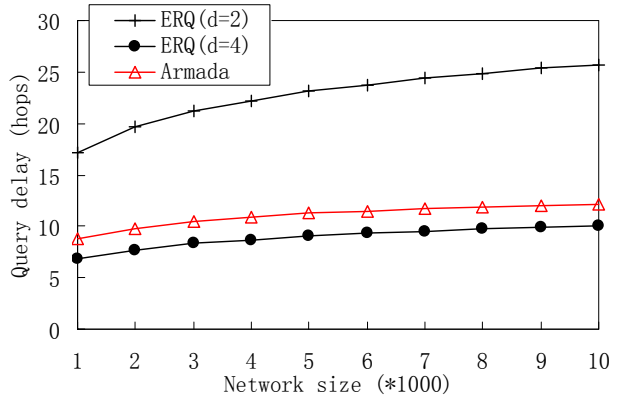



Figure 4. **Average query delay (m = 6).**

We then evaluate the average delay of multi-attribute range queries with $m = 6$. Other parameters ($N$, $S$, $\beta$ and $d$) are the same as the previous experiment. Since DCF-CAN can only support single-attribute range queries, only ERQ and Armada are evaluated.

The results are shown in Figure 4. From this figure we can deduce similar conclusions to Figure 3, which are omitted here due to lack of space.

We then evaluate the impact of the queried range ratio ($\beta$) on the performance of ERQ (with $d = 2$ and $d = 4$). The network size ($N$) is fixed to 6000, the number of attributes ($m$) is 6, and the ratio in all dimensions is simultaneously varied from 0.05 to 0.4. The results are shown in Figure 5.

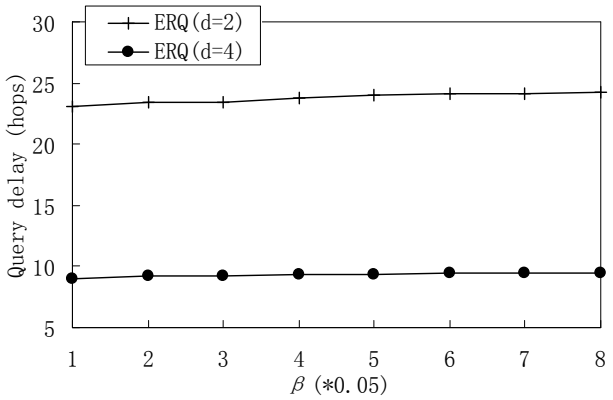



Figure 5. **Impact of queried range ratio on the query delay of ERQ.**

From Figure 5 we can see that the query delay of ERQ almost keeps a constant regardless of the ratio ($\beta$). We conclude that the queried range ratio affects little the performance of ERQ, which demonstrates ERQ's delay-bounded property.

### 5.3. Processing cost

We evaluate the average processing cost of ERQ, as a function of the network size, and compare it with Armada. In our experiments, the base, the number of attributes and the queried range ratio are set to $d = 4$, $m = 6$ and $\beta = 0.2$, respectively.

The results are shown in Figure 6. 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 range queries with a low overhead.

Second, the average cost of ERQ is always less than that of Armada. This is because ERQ's BK tree has fewer levels compared with Armada's forwarding tree, which demonstrates ERQ's advantage of the configurable base $d$.

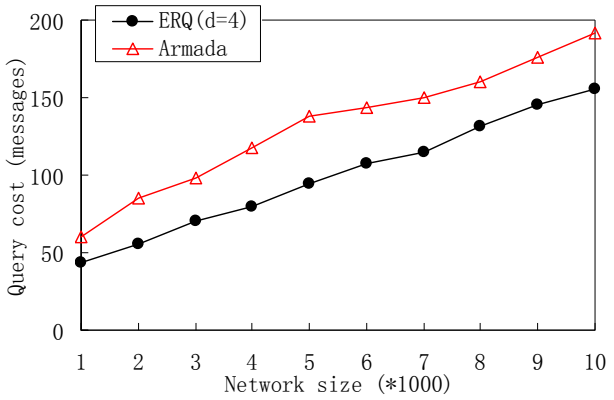



Figure 6. **Average processing cost (m = 6).**

We evaluate the impact of the queried range ratio ($\beta$) on the processing cost of ERQ with $d = 4$, $N = 6000$ and $m = 6$. The ratio in all dimensions is simultaneously varied from 0.05 to 0.4. The results are shown in Figure 7.

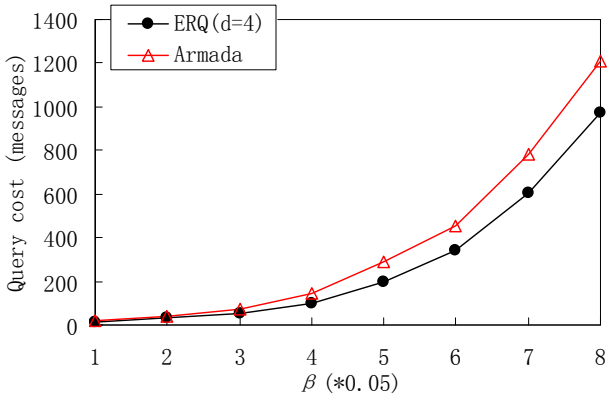



Figure 7. **Impact of queried range ratio on the processing cost of ERQ and Armada.**

From this figure we can see that the processing cost of ERQ increases nearly exponentially with the queried range ratio ($\beta$), which is mainly because the number of nodes involved in the queried range is exponential to $\beta$. In this experiment ERQ again outperforms Armada due to the lower Balanced Kautz tree and the less number of involved nodes.

## 6. Related work

DHT-based range query schemes could be classified into two categories, namely, *layered* schemes and *customized* schemes. The layered schemes [7-15] refer to the ones that are layered over existing DHTs and do not need to modify the topology or behavior of underlying DHTs. In contrast, the customized schemes [16-19] 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 [7] provides range query functionality based on Chord [1]. Squid uses a space-filling curve (SFC) to map data points to nodes and performs range queries by searching SFC clusters recursively. The query delay of Squid is about $O(h*\log N)$, where $h$ is related to the depth of SFC clusters and the specific query. Gupta *et al*. [8] proposes a probabilistic scheme that uses locality sensitive hashing to support single-attribute range queries on Chord. SkipNet [9] is a DHT that directly supports single-attribute range queries, and has a query delay of $O(\log_d N + n)$, where $n$ is the queried range size. Brushwood [10] provides multi-attribute range query functionality based on SkipNet with a high query delay. March *et al*. [11] focuses on multi-attribute range query processing over read-only DHTs.

Among the well-known layered range query schemes, only Armada [12], PlaceLab [13] and DCF-CAN [14] are built on top of *constant-degree* DHTs.

In our previous work [12] we proposed Armada, a delay-bounded range query scheme based on the FissionE DHT [20]. 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. In contrast, ERQ directly inherits the load balancing property from the PHT technique. MR-FissionE [22] is a variation of Armada and supports multi-attribute range queries on top of FissionE [20] with a fixed base $d = 2$. However, it suffers from a long query delay similar to Armada.

DCF-CAN [14] 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 directed controlled flooding (DCF) mechanism can achieve a good tradeoff between query delay and overhead. DCF-CAN can only support single-attribute range queries.

Chawathe *et al*. designed the PHT structure to support range queries in PlaceLab [13]. 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 similar to the BK 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 *et al*. proposes LHT [15] that redesigns the indexing scheme of PHT to reduce the maintenance cost. LHT and PHT have similar range query delay.

### 6.2. Customized schemes

Among the customized range query schemes, Mercury [16] and SWORD [17] provide multi-attribute range queries by indexing the data set along each individual attribute; Liu et al. [18] propose NR-tree, which extends R*-tree index to support range queries and k-nearest neighbor queries in super-peer P2P systems; P-tree [19] builds specific P2P networks to support range queries based on $B^+$-tree. Since customized range query schemes requires a clean-slate design of underlying DHTs which are complicated and error-prone, our ERQ has an obvious methodological advantages over these schemes.

## 7. Conclusion

This paper presents ERQ, a DHT-based, delay-bounded range query scheme for efficient Web service discovery. ERQ designs a balanced Kautz tree to uniformly map the *m*-dimensional data space onto DHT nodes, and presents a parallel query processing algorithm with an on-the-fly space pruning mechanism.

ERQ utilizes the *d*-Search algorithm to locate the representing node for the common prefix. In the future we plan 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. We also plan to extend ERQ to support other complex queries on DK, such as top-*k* queries, skyline queries, and nearest neighbor queries.

## Acknowledgement

This work is sponsored in part by the National Basic Research Program of China (973) under Grant No. 2005CB321801, the National Natural Science Foundation of China under Grant No. 60673167 and 60703072. This work is also partially sponsored by grants from NSF CISE CSR program, CyberTrust Program, an IBM SUR Grant, and an IBM Faculty Award.

## References

[1] I. Stoica, R. Morris, D. R. Karger, et al. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Trans. Networking*, 2003, 11(1): 17–32

[2] S. Ratnasamy, P. Francis, M. Handley, et al. A Scalable Content Addressable Network. *Proc. SIGCOMM 2001*.

[3] Y. Zhang, L. Liu, D. Li, et al. Distributed Line Graphs: A Universal Framework for Building DHTs Based on Arbitrary Constant-Degree Graphs. *Proc. ICDCS 2008*.

[4] Y. Yang, R. Dunlap, et al. Performance of Full Text Search in Structured and Unstructured Peer-to-Peer Systems. *Proc. INFOCOM* 2006.

[5] R. Cox, A. Muthitacharoen, R. T. Morris. Serving DNS Using a Peer-to-Peer Lookup Service. *Proc. IPTPS 2002.*

[6] Y. Zhang, D. Li, et al. PIBUS: A Network Memory-based Peer-to-Peer IO buffering service. *Proc. Networking 2007*.

[7] C. Schmidt, and M. Parashar. Enabling Flexible Queries with Guarantees in P2P systems. *IEEE Internet Computing*, Vol. 8, No. 3, pp. 19-26, May/June 2004.

[8] A. Gupta, D. Agrawal, et al. Approximate Range Selection Queries in Peer-to-Peer systems. *Proc. CIDR 2003*.

[9] N. Harvey, M. Jone, et al. Skipnet: A Scalable Overlay Network with Practical Locality Properties. *Proc. USITS 2003*.

[10] C. Zhang, A. Krishnamurthy, et al. Brushwood: Distributed Trees in Peer-to-Peer Systems. *Proc. IPTPS 2005*.

[11] V. March, Y. M. Teo. Multi-Attribute Range Queries on Read-only DHT. *Proc. ICCCN 2006*.

[12] D. Li, J. Cao, X. Lu, et al. Delay-Bounded Range Query in DHT-based Peer-to-Peer Systems. P*roc. ICDCS 2006*.

[13] Y. Chawathe, et al. A Case Study in Building Layered DHT Applications*. Proc. SIGCOMM 2005*.

[14] A. Andrzejak and Z. Xu. Scalable Efficient Range Queries for Grid Information Services*. Proc. IEEE P2P Computing 2002*.

[15] Y. Tang, S. Zhou. LHT: A Low-Maintenance Indexing Scheme over DHTs*. Proc. IEEE ICDCS* 2008.

[16] A. R. Bharambe, M. Agrawal, S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries*. Proc. SIGCOMM 2004*.

[17] D. Oppenheimer, J. Albrecht, et al. Distributed Resource Discovery on Planetlab with SWORD*. Proc. WORLDS 2004*.

[18] B. Liu, W. Lee, D. L. Lee. Supporting Complex Multi-Dimensional Queries in P2P Systems*. Proc. ICDCS 2005*.

[19] A. Crainiceanu, P. Linga, et al. PTree: A P2P Index for Resource Discovery Applications*. Proc. WWW 2004*.

[20] D. Li, X Lu, J Wu. FISSIONE: A Scalable Constant-Degree and Low Congestion DHT Scheme Based on Kautz Graphs*. Proc. INFOCOM 2005*.

[21] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. *Proc. SIGMOD 1990*.

[22] Y. Zhang, et al. Scalable Distributed Resource Information Service for Internet-Based Virtual Computing Environment. *Journal of Software (in Chinese with English abstract)*. Vol.18, No.8, August 2007, pp.1933–1942.

[23] K. Lee, B. Zheng, H. Li, and W. Lee. Approaching the skyline in Z order. *Proc. VLDB 2007*.