Efficient Range Query Processing in Peer-to-Peer Systems

Dongsheng Li, Jiannong Cao, *Senior Member, IEEE*, Xicheng Lu, and Keith C.C. Chan

Abstract—With the increasing popularity of the peer-to-peer (P2P) computing paradigm, many general range query schemes for distributed hash table (DHT)-based P2P systems have been proposed in recent years. Although those schemes can support range query without modifying the underlying DHTs, they cannot guarantee to return the query results with bounded delay. The query delay in these schemes depends on both the scale of the system and the size of the query space or the specific query. In this paper, we propose Armada, an efficient range query processing scheme to support delay-bounded single-attribute and multiple-attribute range queries. We first describe the order-preserving naming algorithms for assigning adjoining ObjectIDs to objects with close attribute values. Then, we present the design of the forwarding tree to efficiently match the search paths of range queries to the underlying DHT topology. Based on the tree, two query processing algorithms are proposed to, respectively, process single-attribute and multiple-attribute range queries within a bounded delay. Analytical and simulation results show that Armada is an effective general range query scheme on constant-degree DHTs, and can return the query results within $2\log N$ hops in a P2P system with N peers, regardless of the queried range or the size of query space.

Index Terms—Peer-to-peer computing, distributed hash table (DHT), delay bounded, range query.

1 Introduction

Many peer-to-peer (P2P) systems such as Chord [1], CAN [2], Tapestry [3], Pastry [4], and FissionE [5] are based on distributed hash tables (DHTs), using a hash table-like interface to publish and lookup objects. DHT-based P2P systems have proven to be scalable, robust, efficient, and generally applicable. As a result, DHT has become a general infrastructure for building many P2P distributed applications such as archive storage systems, data management systems, application-level multicasts, and discovery services.

As it is not an easy task to implement, deploy, and manage a full-edged DHT for an application, it is valuable to take the DHT infrastructure and related services as an application-independent building block to implement certain key components of many applications. The DHT can act as an outsourced service [6], [7] for easing the implementation of applications, which can inherit the scalability and robustness of the DHT. As the capability of the existing DHT is limited, many services may be built on the same DHT infrastructure to support different applications, and these services should be designed without modifying the underlying DHT.

The basic capability supported by the DHT infrastructure is exact-match query. However, the ever-wider use of DHT infrastructures has found applications that require support for *range query* [8], [9], [10]. Examples of range query include the query "$10{,}000 < salary < 20{,}000$" in P2P data management systems, the query "2.4 GHz $< CPU <$ 4 GHz and 1 Gbyte $\leq Memory \leq 4$ Gbytes" in grid information services, and queries for game information in an area in P2P online games.

A number of range query schemes have been proposed for DHT-based P2P systems. One important category of them is the *general range query scheme* (e.g., [7], [8], [9], and [10]), which is built entirely on top of existing DHT infrastructures and does not need to modify the topology or behavior of the underlying DHTs. This way of using DHTs as a shared general infrastructure allows different applications to be built on the same DHT infrastructure [6], [7], providing the range query capability without the cost of specifically tuning the underlying DHT. However, because such schemes do not adapt the behavior of the underlying DHT to the requirement of range queries, often they are not very efficient. In most existing general range query schemes, the query delay depends on both the total number of peers in the system (N) and the size of the query space or the specific query. As a result, these schemes cannot guarantee to return all query results in a bounded delay that is related only to the scale of the system. When the whole query space or the queried range is large, the query execution can be very slow.

In this paper, we describe Armada, an efficient, delay-bounded general range query scheme. Armada operates on top of FissionE [5], a high-performance constant-degree DHT scheme, and does not need to modify the underlying FissionE infrastructure. Armada provides support for efficient single-attribute and multiple-attribute range queries and can return all query results in a bounded delay, independent of the size of the query space or the

- *D. Li and X. Lu are with the National Lab for Parallel and Distributed Processing, National University of Defense Technology, Changsha, Hunan 410073, P.R. China, and also with the College of Computer, National University of Defense Technology, Changsha, Hunan 410073, P.R. China. E-mail: {dsli, xclu}@nudt.edu.cn.*
- *J. Cao and K.C.C. Chan are with the Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: {csjcao, cskcchan}@comp.polyu.edu.hk.*

Manuscript received 20 Oct. 2006; revised 3 Aug. 2007; accepted 18 Apr. 2008; published online 13 May 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0485-1006.
Digital Object Identifier no. 10.1109/TKDE.2008.99.

1041-4347/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

queried range. The main contributions of this paper include the following three parts:

1. We propose *the partition tree* model to provide order-preserving mappings from the query space to the namespace of FissionE. The single-attribute naming algorithm $Single\_hash$ and the multiple-attribute naming algorithm $Multiple\_hash$ are designed to assign adjoining ObjectIDs to objects with close attribute values, so that they can be published to the same or related peers in the system to support efficient range queries.
2. We design *the forward routing tree* (FRT), which matches the search paths of range queries to the underlying FissionE topology efficiently. Based on the tree, we propose the range query processing algorithms *PrunIng Routing Algorithm* (*PIRA*) and *Multiple-attribute prunIng Routing Algorithm* (*MIRA*) to, respectively, perform single-attribute and multiple-attribute range queries within a bounded delay.
3. We analyze the lower bounds on the delay and message cost for range queries, and evaluate the query delay and message cost of Armada by both theoretical analysis and simulations.

Results of our analysis and simulation studies show that Armada can achieve high efficiency in query processing. For any single-attribute or multiple-attribute range query, Armada can return all query results within $2\log N$ hops (in this paper, $\log N$ represents $\log_2 N$), and its average query delay is less than $\log N$, which reaches the delay lower bound $O(\log N)$ for range queries on constant-degree DHTs. The average message cost of single-attribute range queries in Armada is about $\log N + 2n - 2$ (n is the number of peers that intersect with the query), which is very close to its asymptotic lower bound $O(\log N) + n - 1$. Armada uses FissionE [5] as the underlying DHT to organize the peers in an overlay and to deal with the dynamic join or leave of peers. The average degree of the FissionE overlay is 4 and its average routing delay is less than $\log N$. To our knowledge, Armada is the first delay-bounded range query scheme on top of constant-degree DHTs.

The remainder of this paper is organized as follows: Section 2 introduces the related work. Section 3 gives an overview of Armada. Sections 4 and 5 describe the detail design of single-attribute and multiple-attribute range queries of Armada, respectively. Section 6 evaluates the performance of Armada by analysis and simulations. Section 7 concludes this paper.

2 Related Work

Range query schemes for DHT-based systems can be categorized as either general or customized schemes. General range query schemes (e.g., [7], [8], [9], and [10]) are entirely layered on top of existing DHTs and do not modify the underlying DHT. Customized schemes (e.g., [16], [17], [18], and [19]) either make use of custom-designed P2P overlays or add specific modifications to the behavior of the underlying DHTs. In this paper, we focus on general range query schemes.

2.1 General Range Query Schemes

Chord [1] is a famous DHT of $O(\log N)$ degree. Gupta et al. [8] propose a probability scheme to support single-attribute range queries on Chord. The scheme uses locality-sensitive hashing to locate objects that are relevant to the range query. However, it can only return approximate results. In contrast, Armada can return exact results for range queries.

Schmidt and Parashar propose Squid [9] to provide the multiple-attribute range query capability on Chord. Squid uses a space-filling curve (SFC) [27] to map objects with multiple attributes to peers and processes range queries by searching SFC clusters recursively. Each search step in Squid, however, invokes one DHT routing of Chord, which needs to travel $O(\log N)$ hops in the system. This results in a relatively large delay and message cost. 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), much larger than $\log N$.

Among the existing general range query schemes, only the work reported in [7] and [10] are range query schemes that can run on top of constant-degree DHTs. Andrzejak and Xu [10] propose a single-attribute range query scheme based on CAN [2]. The scheme uses inverse SFC to map a resource with single attribute to peers in CAN's d-dimensional virtual space. For any range query, the scheme first routes the query to the peer in charge of the median value of the query, and then recursively forwards the query to its neighbors until all peers that intersect with the query are visited. The scheme compares three forwarding mechanisms, among which, the directed controlled flooding (DCF) mechanism (hereafter called *DCF-CAN*) can achieve a good overall performance, but it has a query delay of more than $O(N^{1/d})$, with an increasing rate almost proportional to the increase in the size of range queries. *DCF-CAN* can support only single-attribute range query.

PHT [7] is a solution proposed to support both single-attribute and multiple-attribute range queries on any DHT (including constant-degree DHTs). With the assumption that all the keys of objects can be expressed in a binary format, PHT builds a prefix hash tree in which leaf nodes are keys and every internal node corresponds to a distinct key prefix. The prefix tree is distributed among the peers by hashing the labels of tree nodes over the underlying DHT. A range query in PHT is performed by parallel search on the tree, but each search step along the tree must invoke one DHT routing. PHT is a good general scheme that can run on any DHT, but its delay and message cost are related to the query space and overly large. When the underlying DHT is of constant degree, its query delay is about $O(b^* \log N)$, where b is the height of the prefix tree.

Compared with the schemes reported in [7] and [10], Armada is delay-bounded and, given the same degree of the underlying DHTs, the average query delay of Armada is less than $\log N$, much less than that of the two schemes.

2.2 Customized Range Query Schemes

Many customized range query schemes have been proposed in recent years. Skip graph [11] and SkipNet [12] are P2P networks of $O(\log N)$ degree based on the skip-list data structure. They can directly support single-attribute range queries, but have a range query delay of $O(\log N + n)$ (n is the number of peers that intersect with the query), which is

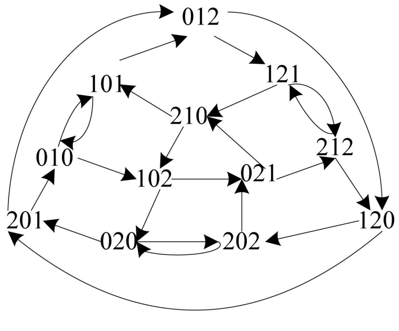

Fig. 1. Kautz graph $K(2,3)$.

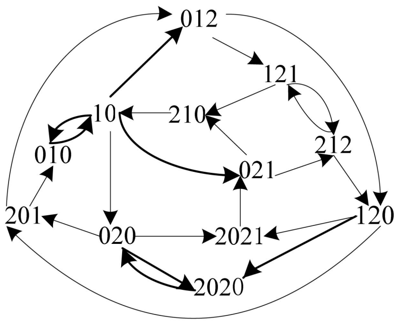

Fig. 2. An example of the FissionE overlay.

dependent on the sizes of specific queries. Family tree [13] and Rainbow skip graph [14] both are constant-degree overlays that can support single-attribute range queries. Family tree [13] combines the techniques of Viceroy and SkipNet and has a range query delay of $O(\log N + n)$ in expectation and $O(\log^2 N + n)$ with high probability. Rainbow skip graph [14] is an adaptation of the Skip graph and has a range query delay of $O(\log N)$ with high probability. SCRAP [15] uses the SFC to support multiple-attribute range queries on Skip graph, but its query delay remains to be $O(\log N + n)$.

Mercury [16] and SWORD [17] provide multiple-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. MURK [15] and P-tree [19] build specific P2P networks to support range queries, respectively, based on KD-tree [28] and B^+-tree. Brushwood [20] and Znet [21] provide the multiple-attribute range query capability on Skip Graph. Aspnes et al. [22] and Ganesan et al. [23] propose mechanisms to improve the load balance of range queries. Sahin et al. [24] present a scheme for caching range queries. VBI-Tree [25] builds a framework based on a binary balanced tree structure to support both point queries and range queries efficiently. Skip-webs [26] improve randomized distributed data structures (e.g., SkipNet and Skip graph) and present a framework for designing efficient distributed data structures for single-attribute or multiple-attribute range queries. However, all the above schemes are customized P2P overlays or need to make specific modifications to the underlying P2P networks.

There are also some work (e.g., [32], [33], and [34]) on P2P queries that are orthogonal to ours. Tang et al. [32] propose pSearch to query relevant documents in CAN-based P2P systems. Awerbuch and Scheideler [33] propose a methodology to support prefix search for user-defined names in P2P systems. Shen et al. [34] present a three-tier framework to support semantic based retrieval of documents in P2P networks.

3 System Overview

3.1 Background: FissionE

We first give an introduction to FissionE on which Armada is built. FissionE [5] is a constant-degree DHT scheme based on Kautz graph $K(2,k)$ [5], which is a static topology with many desirable properties such as optimal diameter and optimal fault tolerance.

A *Kautz string* ξ [5] of length k and base d is defined as a string $a_1a_2\ldots a_k$, where $a_j \in \{0,1,2,\ldots,d\}$ $(1 \le j \le k)$ and $a_i \ne a_{i+1}$ $(1 \le i \le k-1)$, i.e., neighboring symbols in a Kautz string should be different. The *Kautz namespace* $KautzSpace(d,k)$ is the set containing all Kautz strings of length k and base d. The *Kautz graph* $K(d,k)$ is a directed graph in which each node is labeled with a Kautz string in $KautzSpace(d,k)$ and has d outgoing edges: for each $\alpha \in \{0,1,2,\ldots,d\}$ and $\alpha \ne u_k$, node $U = u_1u_2\ldots u_k$ has one out-edge to node $V = u_2u_3\ldots u_k\alpha$ (denoted by $U \to V$). For example, node 012 in $K(2,3)$ has two out-edges: 012 → 120 and 012 → 121. Fig. 1 shows the Kautz graph $K(2,3)$.

In FissionE, the identifiers of peers (i.e., PeerIDs) are Kautz strings of base 2 and their lengths may be different. The maximum length of PeerIDs is less than $2\log N$ and the average length is less than $\log N$. Peers are organized into an approximate Kautz graph according to their PeerIDs. FissionE maintains a topology rule called *neighborhood invariant* which requires that the difference between the lengths of PeerIDs of neighboring peers is always no more than one. Therefore, PeerIDs of out-neighbors of peer $U = u_1u_2\ldots u_k$ are in the style of $u_2u_3\ldots u_kq_1\ldots q_m$ with $0 \le m \le 2$. Fig. 2 shows an example of the topology of the FissionE overlay.

Routing in FissionE is based on the left-shifted routing algorithm in the Kautz graph, which is accomplished by taking the label of source node U and left shifting in symbols of the label of the destination node V. For example, the routing path in the Kautz graph $K(2,3)$ from node 010 to node 120 is $010 \to 101 \to 012 \to 120$, and the routing path from peer 010 to peer 120 in the FissionE overlay with topology shown in Fig. 2 is $010 \to 10 \to 012 \to 120$.

Each object in FissionE is assigned an ObjectID by the naming algorithm *Kautz_hash*. ObjectIDs are Kautz strings distributed in the Kautz namespace $KautzSpace(2,100)$ and are of fixed length 100, which is long enough for a P2P system with 2^{50} peers [5]. Each object is published onto a unique peer whose PeerID is a prefix of its ObjectID. FissionE adopts effective self-stabilization and fault-tolerant mechanisms to deal with the dynamic join or leave of peers.

Analysis and simulations show that FissionE is a high-performance constant-degree DHT. The average degree of FissionE is 4, its diameter is less than $2\log N$, its average routing delay is less than $\log N$, and the maintenance message cost of peer joining or leaving is less than $3\log N$.

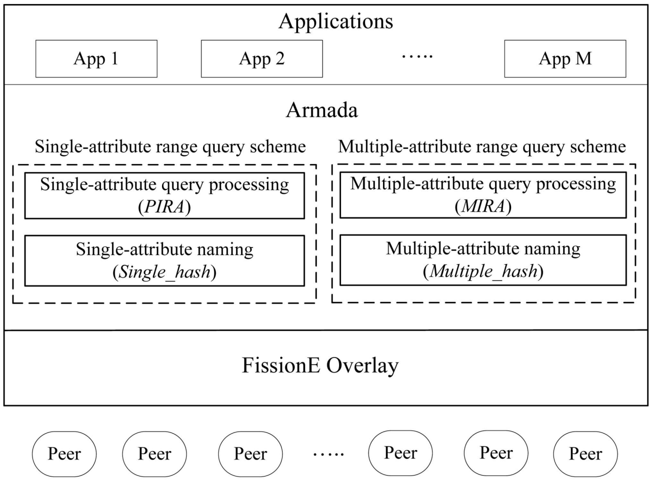
Applications
App 1
App 2
.....
App M
Armada
Single-attribute range query scheme
Multiple-attribute range query scheme
Single-attribute query processing (PIRA)
Multiple-attribute query processing (MIRA)
Single-attribute naming (Single_hash)
Multiple-attribute naming (Multiple_hash)
FissionE Overlay
Peer
Peer
Peer
.....
Peer
Peer
Peer

Fig. 3. Components of Armada and its relation to FissionE.

3.2 Armada Design

Like many other DHTs, FissionE provides support for scalable and efficient exact-match query of distributed objects on peers. However, it cannot support range queries for attribute values. In this paper, we have designed a general range query scheme, called Armada, to support range queries on top of FissionE without modifying the underlying FissionE DHT.

The basic components of Armada include two parts: *order-preserving naming* and *range query processing.* Armada first uses order-preserving naming algorithms to assign to objects with close attribute values the ObjectIDs adjoining in the Kautz namespace so as to publish them on related peers. Then, Armada provides efficient query processing algorithms to forward range queries to appropriate peers and return query results within a bounded delay.

Based on the number of attributes that queried, range queries can be classified as the single-attribute range query and the multiple-attribute range query. Armada adopts order-preserving naming algorithms $Single\_hash$ and $Multiple\_hash$, and query processing algorithms *PIRA* and *MIRA*, to perform single-attribute and multiple-attribute range queries, respectively. The components of Armada and its relation to the underlying FissionE DHT are shown in Fig. 3.

Like other general range query schemes [7], [8], [9], [10], Armada is built on top of the underlying DHT. It relies on the underlying DHT to organize its P2P overlay and provide much of the robustness, availability, and load balancing. Armada uses the naming algorithms to assign to objects order-preserving ObjectIDs and efficiently propagates the range queries in the overlay, while the underlying FissionE DHT organizes the peers in an overlay and handles the dynamic joining or leaving of peers. If a peer fails, the underlying FissionE DHT automatically ensures that other peers in the overlay takes over the responsibility for the failed peer and provides graceful fail-over by using replication or other mechanisms. And the underlying FissionE DHT also deals with the routing and publishing of objects according to the ObjectIDs. In some sense, the underlying DHT shields Armada from the dynamics of peers and the complexity of the P2P overlay, so the design of Armada can be focused on the naming and range query processing algorithms.

In the following sections, we describe the naming and query processing algorithms used in Armada, respectively, for single-attribute and multiple-attribute range queries.

4 Single-Attribute Range Query

In this section, we present the design of the single-attribute range query scheme in Armada.

4.1 Single-Attribute Naming

FissionE uses the *Kautz_hash* naming algorithm, which generates ObjectIDs by hashing keywords (or values) of objects. However, *Kautz_hash* destroys the locality of attribute values, and thus cannot support range queries. In this section, we propose an order-preserving naming algorithm $Single\_hash$ to assign to objects with close single-attribute values the ObjectIDs adjoining in the Kautz namespace. According to the properties of FissionE, objects with adjoining ObjectIDs are published on the same or related peers.

In this paper, we assume that the entire interval of attribute values of objects is a real-number interval $[L, H]$ and use symbol $\prec$ to denote the "no more than" relation between Kautz strings in the lexicographical order. It is fairly straightforward to extend Armada to support the attribute values in some other forms. Below, we give some definitions about *order-preserving* naming.

Definition 1. *The Kautz region* $[\![\alpha, \beta]\!]$ *is defined as a subset of Kautz namespace* $KautzSpace(2, k)$ *which includes all Kautz strings between* α *and* β, *i.e.,* $[\![\alpha, \beta]\!] = \{s | s \in KautzSpace(2, k) \text{ and } \alpha \prec s \text{ and } s \prec \beta\}$.

For example, $[\![010, 021]\!] = \{010, 012, 020, 021\}$. Because the namespace of objects is the $KautzSpace(2{,}100)$ and each object is published onto the peer whose PeerID is a prefix of its ObjectID, each peer in FissionE is in charge of a Kautz region in the Kautz namespace $KautzSpace(2{,}100)$. For example, peer 10 owns all objects whose ObjectIDs has a prefix 10, thus it is in charge of the Kautz region $[\![101^*, 102^*]\!]$.

Definition 2. *Assume* F *is a surjection function from the real-number interval* D *to the Kautz namespace* V. F *is an order-preserving function if and only if, for any* a_1 *and* a_2 *in* D, *if* $a_1 \leq a_2$, *then* $F(a_1) \prec F(a_2)$.

Definition 3. *Assume* F *is an order-preserving function from the real-number interval* D *to the Kautz namespace* V. F *is an interval-preserving function if and only if, for any subinterval* $[a, b]$ *of* D, *the corresponding range of function* F *is the Kautz region* $[\![F(a), F(b)]\!]$.

If the $Single\_hash$ naming algorithm is designed to be an interval-preserving function from attribute-value interval $[L, H]$ to $KautzSpace(2, k)$ (k is 100 in FissionE or Armada), any attribute-value subinterval can be mapped to a Kautz region in the charge of some related peers. Then, range queries can be performed by forwarding queries to the appropriate peers.

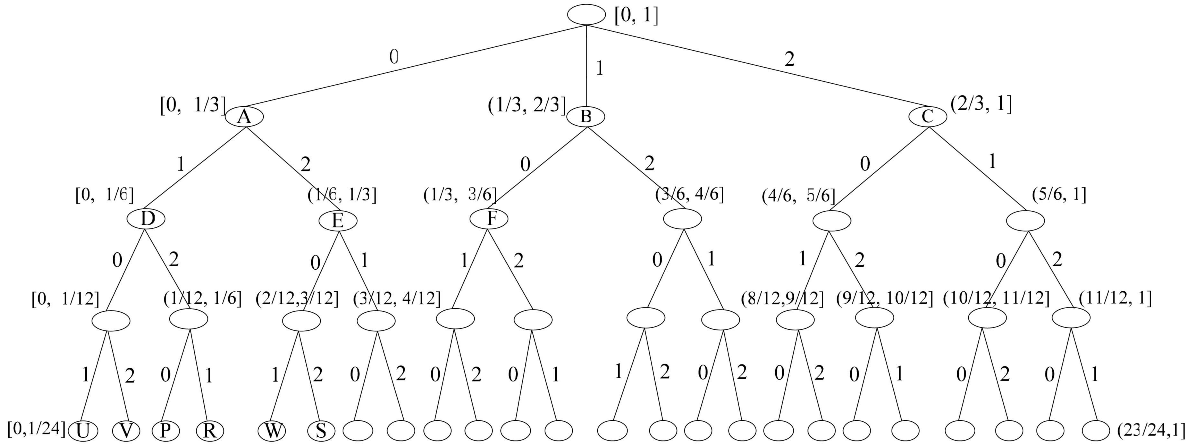

Fig. 4. Partition tree $P(2, 4)$ for attribute-value interval [0, 1].

We propose a *partition tree* $P(2, k)$ model to help design the *Single_hash* algorithm. The structure of the partition tree $P(2, k)$ is similar to that of a complete binary tree, but different in labels of edges and branches of the root. The partition tree $P(2, k)$ has $k + 1$ levels with the root node at the 0th level. The root node has three child nodes, while other intermediate nodes have only two children. Labels of edges from a father node to its children can be 0 or 1 or 2, increasing from left to right, but they should be different from in-edge's label of the father node. The label of the root node is *null*, and the label of any other node is the concatenation of the labels of the edges on the path from the root to it. It is easy to see that the labels of the leaf nodes in $P(2, k)$ contain all Kautz strings in $KautzSpace(2, k)$ and they increase from left to right in the order of $\prec$. In Armada, k is set to be 100 because the length of ObjectIDs is 100. Fig. 4 shows an example of the partition tree $P(2, 4)$.

We partition the entire interval of attribute values $[L, H]$ onto the partition tree $P(2, k)$. The root node represents the entire interval $[L, H]$, and other nodes represent subintervals of $[L, H]$. Each child node evenly partitions the subinterval represented by its father node. In the example shown in Fig. 4, the entire interval of attribute values is [0, 1]. Nodes A, B, and C, with labels 0, 1, and 2, respectively, are children of the root and evenly partition the interval [0, 1]. Thus, nodes A, B, and C represent subintervals [0, 1/3], (1/3, 2/3], and (2/3, 1], respectively.

Since leaf nodes in $P(2, k)$ and Kautz strings in $KautzSpace(2, k)$ are biunique, the interval $[L, H]$ can be partitioned into some subintervals, each of which is represented by a unique Kautz string. Thus, we can design the naming algorithm *Single_hash* to map the attribute values in $[L, H]$ to Kautz strings in $KautzSpace(2, k)$ based on the partition tree. It works as follows: Suppose the attribute value of object O is c ($c \in [L, H]$), c surely lies in a subinterval represented by a leaf node in the partition tree. Suppose the label of the leaf node is S, then the Kautz string S is assigned as O's ObjectID, i.e., $Single\_hash(c, L, H, k) = S$. In the example shown in Fig. 4, the attribute value 0.1 is in the subinterval represented by the leaf node P with label 0120; thus, the Kautz string 0120 is assigned as the ObjectID of the object with attribute value 0.1 by the *Single_hash* algorithm. The pseudocode of the *Single_hash* algorithm is shown in Fig. 5.

Notice that the partition tree is only a visual assistant model to help design the *Single_hash* algorithm, and Armada does not need to maintain any structure for or information about it. We can use the *Single_hash* algorithm as shown in Fig. 5 directly. Meanwhile, as each object is published onto the peer whose PeerID is a prefix of the object's ObjectID, the subinterval that a real physical peer P is in charge of can be determined by the partition tree (or the *Single_hash* algorithm). In the example shown in Figs. 2 and 4, the ObjectID of any object with an attribute value in [0, 1/24] is 0101 and the ObjectID of any object with an

```
Procedure Single_hash ( AttributeValue c, Value L, Value H, Length k)
// The entire interval of attribute values is [L, H]
// The attribute value of the object is c
// The generated ObjectID is a Kautz string S of length k
left ← 0; right ← 1;
  // initialize the vector nextID
nextID[0][left] ←1;  nextID[0][right] ←2;
nextID[1][left] ←0;  nextID[1][right] ←2;
nextID[2][left] ←0;  nextID[2][right] ←1;
if c>H or c<L   then return error;
S ←null;
  // value c lies in the subinterval represented by the
       first child of the root node in the partition tree
if c ∈ [L, L+1/3*(H–L)]
  then S[0] ←0; a←L; b← L+1/3*(H–L) ;
  // value c lies in the second child of the root node
if c ∈ (L+1/3*(H–L), L+2/3*(H–L)]
  then S[0] ←1; a← L+1/3*(H–L); b← L+2/3*(H–L);
  // value c lies in the third child of the root node
if c ∈ (L+2/3*(H–L), H]
  then S[0] ←2; a← L+2/3*(H–L); b←H;
  // determine whether value c lies in the subinterval
       represented by the left or right child
for i=1 to k–1
  do if c>(a+b)/2
       then direction←right; a← (a+b)/2;
       else direction←left;  b← (a+b)/2;
     S[i] ←nextID[S[i–1]][direction];
return S;
```

Fig. 5. The pseudocode of the *Single_hash* algorithm.

attribute value in (1/24, 1/12] is 0102. Thus, the subinterval that peer 010 is in charge of is the real-number interval [0, 1/12].

Theorem 1. *The* $Single\_hash$ *algorithm is an interval-preserving function from the attribute-value interval* $[L, H]$ *to the Kautz namespace* $KautzSpace(2, k)$.

The proof of Theorem 1 is straightforward and shown in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99. In the example shown in Fig. 4, the range of [0.1, 0.24] by the $Single\_hash$ algorithm is the Kautz region $[\![0120, 0202]\!]$, which contains four adjoining Kautz strings 0120, 0121, 0201, and 0202.

4.2 Object Publishing

With the naming algorithm $Single\_hash$, the procedure of object publishing in Armada works as follows: if peer P wants to publish an object O with attribute value c, it first obtains O's ObjectID S by using the $Single\_hash$ algorithm (i.e., $S = Single\_hash(c, L, H, k)$). It then invokes a FissionE DHT routing to reach Kautz string S. By the characteristics of FissionE [5], the routing will stop at a unique peer W whose PeerID is a prefix of S. Then, object O is published onto peer W and the average delay of publishing is less than $\log N$ hops [5]. Armada also adopts the self-stabilization and fault-tolerant mechanisms of the underlying FissionE DHT to deal with the dynamic join and leave of peers.

4.3 Single-Attribute Range Query Processing

When a peer P invokes a range query $[LowV, HighV]$, it first acquires Kautz strings $LowT$ and $HighT : LowT = Single\_hash(LowV, L, H, k)$ and

$$HighT = Single\_hash(HighV, L, H, k).$$

Because the $Single\_hash$ algorithm is interval-preserving, objects with attribute values in the interval $[LowV, HighV]$ are published exactly onto peers that are in charge of the Kautz region $[\![LowT, HighT]\!]$. Now, we discuss the search algorithm for these destination peers.

In FissionE, PeerIDs of out-neighbors of peer $P = u_1u_2 \ldots u_b$ are $u_2 \ldots u_b v_1 \ldots v_q$ $(0 \le q \le 2)$. Based on the topology properties of FissionE, we can form a *forward routing tree* (FRT) for any peer P. The FRT of peer $P = u_1u_2 \ldots u_b$ is formed by using the following four rules:

1. The root is peer P.
2. Each node in the FRT is a peer in FissionE.
3. For each node in the FRT, its child nodes at the next level are its out-neighbors in FissionE, and they are sorted from left to right in the increasing order of $\prec$ defined over PeerIDs.
4. The FRT has $(b + 1)$ levels with the root node at the 0th level.

According to these rules, the ith level $(0 \le i \le b - 1)$ of the FRT contains all the peers whose PeerIDs have a prefix $u_{i+1} \ldots u_b$ and the last level (bth level) contains all the peers whose PeerIDs do not have u_b as the first symbol.

Fig. 6 shows the FRT of peer 212 with the FissionE topology shown in Fig. 2. The FRT of peer 212 has four levels, and nodes at the first and second levels, respectively, have a common prefix 12 and 2, which are both suffixes of 212.

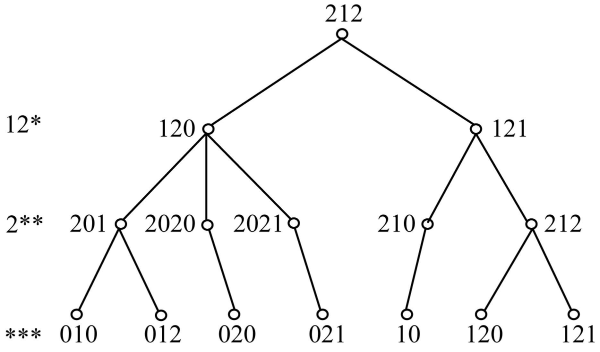

Fig. 6. An example of the FRT.

Based on the FRT, Armada uses *PIRA* to perform a pruning search in the FRT for all the destination peers that are in charge of the Kautz region $[\![LowT, HighT]\!]$. Suppose the Kautz strings $LowT$ and $HighT$ have a common prefix (if they have no common prefix, we can divide $[\![LowT, High]\!]$ into several (at most three) subregions with common prefixes and deal with each subregion, respectively), then all the destination peers are at the same level of the FRT. Let $ComT$ denote the longest common prefix of $LowT$ and $HighT$, and $ComS$ the longest Kautz string which is both the prefix of $ComT$ and the suffix of the root peer P's PeerID. Suppose the length of $ComS$ is f, then all the destination peers are adjoining nodes at the $(b - f)\text{th}$ level of the FRT. For example, in the FRT shown in Fig. 6, if peer 212 performs a pruning search for all the destination peers that are in charge of the Kautz region $[\![2010, 2021]\!]$, we have $ComT = 20$ and $ComS = 2$. Thus, $b = 3$ and $f = 1$, and all destination peers are at the second level of the FRT.

When a peer B at the $i\text{th}$ $(0 \le i \le b - 1)$ level of the FRT receives the search message, the PeerID of B is $u_{i+1} \ldots u_{b-f}X$. Consider any out-neighbor $C = u_{i+2} \ldots u_{b-f}XY$ of peer B, peer C is at the $(i + 1)\text{th}$ level of the FRT. By the properties of the FRT, PeerIDs of C's descendants at the $(b - f)\text{th}$ level of the FRT have a prefix XY. If the Kautz region $[\![LowT, HighT]\!]$ includes a Kautz string that has a prefix XY, descendants of C in the FRT will contain part of the destination peers, and peer B should forward the search message to peer C. The pseudocode of *PIRA* is shown in Fig. 7.

From the above discussion, it is easy to see that *PIRA* can forward any single-attribute range query exactly to all the destination peers that intersect with the query.

Fig. 8 shows an example of using *PIRA* for search in the FRT shown in Fig. 6. In the example, peer 212 issues a range query [0.1, 0.24], the entire attribute-value interval is [0, 1] and $k = 4$. By the $Single\_hash$ algorithm, we can get $LowT = 0120$ and $HighT = 0202$. Thus, the destination peers are all at the third level of the FRT. The dashed lines with arrows in Fig. 8 show search paths of *PIRA*.

4.4 Dynamic Maintenance and Load Balancing

Design of range query support in P2P systems should take into account the dynamics and load balancing of peers. As

```
Procedure P.PIRA (Value LowV, Value HighV)
// Peer P invokes a range query for objects with
      attribute values in the range [LowV, HighV]
// The entire attribute-value interval is [L, H]
// The length of ObjectIDs is k
if LowV < HighV  then return error;
LowT←Single_hash(LowV, L, H, k);
HighT←Single_hash(HighV, L, H, k);
ComT ←CommonPrefix(LowT, HighT);
if ComT=null
    then  Rangset←DivideRange(LowT, HighT) ;
          //divide ⟦ LowT, HighT ⟧ into several sub-regions
          //parallel search for each sub-region Range_i
          for each Range_i ∈ Rangset
              do P.PruningSearch(range_i.LowT, range_i.HighT);
    else  P.PruningSearch(LowT, HighT) ;

Procedure P.PruningSearch (String LowT, String HighT)
// Peer P invokes a pruning search for all destination
      peers in charge of Kautz region ⟦ LowT, HighT ⟧
ComT ←CommonPrefix(LowT, HighT);
   // if peer P's PeerID is a prefix of ComT, the
         destination peer is P and the search is finished
if isprefix(P, ComT)   then  query(P); return;
ComS ← SuffixPrefix(PeerID(P), ComT);
MaxLevel←Lengthof (PeerID(P)) – Lengthof (ComS);
P. IDown (MaxLevel, LowT, HighT);

Procedure U. IDown (Depth h, String LowT, String HighT)
// Peer U=a_1...a_hX deals with the pruning search message
      for Kautz region ⟦ LowT, HighT ⟧
// The level of peer U is h higher than that of destination
      peers in the FRT
if  h =0    //reach a destination peer
   then query(U);  return ;
   else  for each R=a_2...a_hXY ∈ outneighbors (U)
             do  W← XY;
                 len← LengthofString(W);
                 if  (prefix(LowT, len) ≺ W ) and
                                    ( W ≺ prefix(HighT, len) )
                     then  R. IDown (h–1, LowT, HighT);
```

Fig. 7. The pseudocode of *PIRA*.

Armada is a general range query scheme built on top of the underlying DHT, it can make use of the dynamic maintenance and failure recovery mechanisms of the underlying DHT to deal with the join/leave/fail of peers. When some peers in the query path fail, Armada can make use of the fault-tolerant routing mechanism of the underlying DHT [5] to forward the query messages. Armada can also take advantage of the replication techniques employed by the underlying DHT to deal with the loss of objects due to the failure of peers.

Load balancing is another important and widely studied problem in DHT-based systems. Many mechanisms, such as ID selection [37], virtual servers [38], and item balancing [39], have been proposed to improve load balancing of DHTs. In Armada, when the distribution of objects' attribute values is skewed, the distribution of ObjectIDs generated by the $Single\_hash$ algorithm may not be uniform, and thus the load across the peers may become unbalanced. Armada uses two mechanisms to balance load across peers: 1) ObjectID balancing: a *Probability-based lOad Balancing Mechanism* (*POBM*) is used to generate uniform ObjectIDs when the distribution of attribute values is known in advance and 2) runtime balancing: each peer in Armada houses multiple virtual servers and the load is balanced by transferring virtual servers from heavily loaded peers to lightly loaded peers. Armada can build on the underlying DHT the mechanisms to achieve the runtime balancing [38], [39]. In this paper, we give a brief introduction to *POBM* and leave the detail of the runtime balancing mechanisms in our future work.

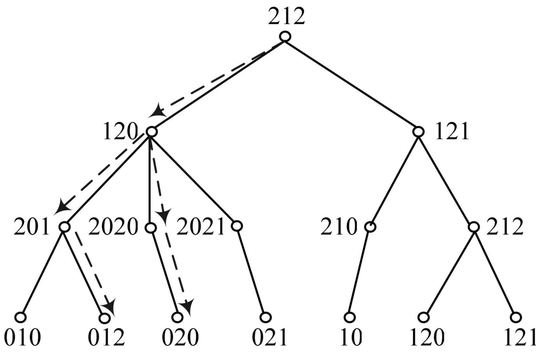

Fig. 8. An example of *PIRA*.

The basic idea of *POBM* is to partition the entire attribute-value interval $[L, H]$ onto the partition tree according to the probability distribution of the attribute values and to ensure that the probability distribution of the attribute values in the subinterval represented by each node at the same level of the partition tree is equal. Suppose the probability density function (*PDF*) of the attribute values in interval $[L, H]$ is $\rho(x)$ (obvious $\int_L^H \rho(x) = 1$). If node A representing a subinterval $[a, b]$ in the partition tree has f child nodes, then the subinterval $[a_i, b_i]$ represented by any child node satisfies $\int_{a_i}^{b_i} \rho(x) = \int_a^b \rho(x)/f$. Based on $POBM$, we can propose a new naming algorithm $Balance\_hash$ to replace the $Single\_hash$ algorithm to assign ObjectIDs to objects. For any object with attribute value v, we have $Balance\_hash(v) = Single\_hash(\int_L^v \rho(x), 0, 1, k)$. *POBM* also makes a slight revision to *PIRA* by replacing $Single\_hash$ with $Balance\_hash$ in the pseudocode of *PIRA*.

In some applications, the exact *PDF* of attribute values is unknown in advance, but it is possible to obtain the probability distribution in certain subintervals by historical statistic. For example, it is known that 20 percent attribute values are distributed in a subinterval $[c_1, c_2]$. Generally, if we have known p points $(v_1, F_1), \ldots, (v_p, F_p)$ of the probability distribution function $F(x)$, for any object O with attribute value v, we can calculate the function value $F(v)$ by using Newton interpolation method [31]. Then, $POBM$ can use $Single\_hash(F(v), 0, 1, k)$ to generate the ObjectID of object O.

5 Multiple-Attribute Range Query

Many applications require the support for multiple-attribute range query on DHTs, e.g., the query "$15 \leq age \leq 18$ and

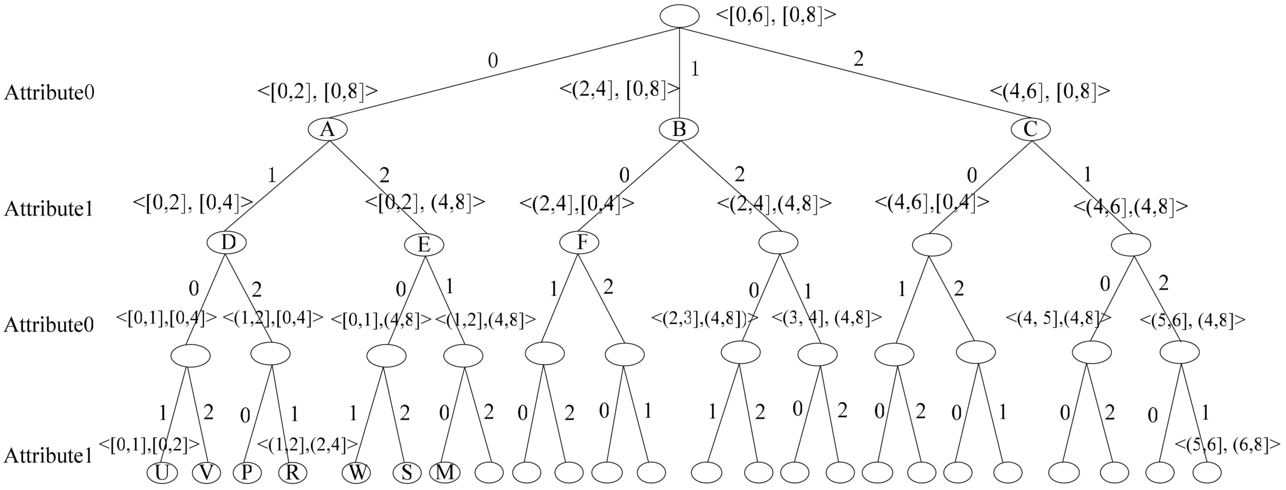

Fig. 9. Partition tree $P(2,4)$ for multiple-attribute space $< [0,6], [0,8] >$.

$80.5 \leq score \leq 95$" in P2P data management systems. Next, we describe Armada's naming and query processing algorithms for multiple-attribute range queries.

5.1 Multiple-Attribute Naming

Assume that there are m attributes $A_0, A_1, \ldots,$ and A_{m-1} and the entire attribute-value interval of A_i is $[L_i, H_i]$. The multiple-attribute value of an object is denoted by a tuple $\delta = <v_0, v_1, \ldots, v_{m-1}>$, where v_i $(0 \leq i \leq m-1)$ is the value of attribute A_i. The multiple-attribute value space, called *multiple-attribute space*, is an m-dimensional subspace and denoted by a tuple $\omega = <r_0, r_1, \ldots, r_i, \ldots, r_{m-1}>$, where r_i is a subinterval of the value of attribute A_i (e.g., $r_i = (x_i, y_i]$). We then give the definition of the partial-order $\lhd$ between multiple-attribute values.

Definition 4. *For two multiple-attribute values* $\delta_1 = <u_0, u_1, \ldots, u_{m-1}>$ *and* $\delta_2 = <v_0, v_1, \ldots, v_{m-1}>$ *in multiple-attribute space,* $\delta_1 \lhd \delta_2$ *if and only if, for each* $0 \leq i < m-1$, $u_i \leq v_i$.

Definition 5. *Assume that* F *is a surjection function from the multiple-dimensional space* D *to the Kautz namespace* V. F *is a multiple-attribute partial-order preserving function if and only if, for any* δ_1 *and* δ_2 *in* D, *if* $\delta_1 \lhd \delta_2$, *then* $F(\delta_1) \prec F(\delta_2)$.

Some algorithms have been proposed, including SFC [27], KD-tree [28], and Z-curve [29], for mapping from a multiple-dimensional space to a line. But these algorithms cannot generate the Kautz strings needed in FissionE. We again use the partition tree to help design the multiple-attribute naming algorithm, $Multiple\_hash$, to assign to objects *partial-order preserving* ObjectIDs. We partition the entire multiple-attribute space $<[L_0, H_0], \ldots, [L_i, H_i], \ldots, [L_{m-1}, H_{m-1}]>$ onto the partition tree along attributes $A_0, A_1, \ldots,$ and A_{m-1} in a round-robin style. Each node in the partition tree represents a multiple-attribute subspace and the root node represents the entire space $<[L_0, H_0], \ldots, [L_i, H_i], \ldots, [L_{m-1}, H_{m-1}]>$. For any node P at the jth level with f child nodes, let i denote the value of $j \bmod m$. Then, the subspace ω represented by node P is evenly divided into f pieces along the ith attribute (i.e., attribute A_i), and each of the f child nodes represents one piece. Thus, each node at the same level of the tree represents a multiple-attribute subspace of the same size and the union of all such subspaces is the entire multiple-attribute space. Fig. 9 shows an example of the partition tree $P(2,4)$ that represents the 2D $(m = 2)$ multiple-attribute space $<[0,6], [0,8]>$.

There can be several ways to partition the attributes over the partition tree levels, but the round-robin style is simple and scalable for the number of attributes or the scale of the Armada system. The round-robin style is helpful to produce ObjectIDs that reflect the objects' attribute values of each dimension. It can also be effective to utilize the query condition of each attribute to reduce the number of peers searched during the range query processing.

Based on the partition tree for multiple-attribute space, the naming algorithm, $Multiple\_hash$, can be designed. It works as follows: For any object O with the multiple-attribute value $V = < v_0, v_1, \ldots, v_{m-1} >$, V is surely in a subspace represented by a leaf node in the partition tree. Then, the label of the leaf node is assigned as O's ObjectID. The idea of the $Multiple\_hash$ algorithm is similar to the $Single\_hash$ algorithm, and its pseudocode can be found in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99.

Theorem 2. *The* $Multiple\_hash$ *algorithm is a multiple-attribute partial-order preserving function from the multiple-attribute space to the Kautz namespace* $KautzSpace(2,k)$.

The proof of Theorem 2 is straightforward. Once an object is assigned ObjectID by the $Multiple\_hash$ algorithm, publishing the object is the same as that in Section 4.2 and omitted here.

5.2 Multiple-Attribute Range Query Processing

Suppose a peer $P = u_1 u_2 \ldots u_b$ issues a multiple-attribute range query $\omega = <[x_0, y_0], \ldots, [x_i, y_i], \ldots, [x_{m-1}, y_{m-1}]>$. Let δ_1 denote $<x_0, x_1, \ldots, x_{m-1}>$ and δ_2 denote $<y_0, y_1, \ldots, y_{m-1}>$, and let $LowT = Multiple\_hash(\delta_1, L, H, k)$ and $HighT = Multiple\_hash(\delta_2, L, H, k)$. Because the $Multiple\_hash$ algorithm is a multiple-attribute partial-order preserving function, the codomain of ω by $Multiple\_hash$ is the Kautz region

$[\![LowT, HighT]\!]$. However, $Multiple\_hash$ does not have the interval-preserving property that $Single\_hash$ has, thus the range of ω may be only a proper subset of $[\![LowT, HighT]\!]$. In the example shown in Fig. 9, when $\omega = <[1.2, 1.8], [1, 5]>$, $\delta_1 = <1.2, 1>$, and $\delta_2 = <1.8, 5>$. Thus, $[\![LowT, HighT]\!] = [\![0120, 0210]\!]$, which contains five leaf nodes P, R, W, S, and M, while nodes W and S do not intersect with the query. It is easy to see that any mapping from a multiple-dimensional space to a 1D space cannot be interval-preserving, thus we cannot improve $Multiple\_hash$ to achieve that. If we use *PIRA* to forward multiple-attribute range queries to all the peers that are in charge of $[\![LowT, HighT]\!]$, *PIRA* may search many peers that do not intersect with the query.

Therefore, we propose a new algorithm, called *MIRA*, to process multiple-attribute range queries. *MIRA* follows the basic idea of *PIRA* to perform pruning search on the FRT of peer $P = u_1u_2 \dots u_b$ that issues the range query ω, as *PIRA* introduced in Section 4.3. However, when forwarding the query to a node in the FRT, *MIRA* needs to determine whether some descendants of the node in the FRT intersect with the query ω, while *PIRA* only needs to determine the relation between the PeerID and the Kautz region $[\![LowT, HighT]\!]$. The pseudocode of *MIRA* can be found in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99, and omitted here due to the limit on paper length. Below, we only discuss *MIRA* about the case in which the Kautz strings $LowT$ and $HighT$ have a common prefix.

In this case, peers that are in charge of Kautz region $[\![LowT, HighT]\!]$ are at the same level of the FRT, but they may be not adjoining. Suppose $ComS$ of length f is the longest Kautz string which is both the prefix of $LowT$ and $HighT$ and the suffix of the root P's PeerID. Then, all the peers that intersect with query ω are at the $(b - f)\text{th}$ level of the FRT. *MIRA* can perform a pruning search on the FRT to reach exactly all the destination peers as follows: When peer $B = u_{i+1} \dots u_{b-f}X$ at the ith level of the FRT receives the query, any out-neighbor $C = u_{i+2} \dots u_{b-f}XY$ of peer B is at the $(i + 1)\text{th}$ level of the FRT. By the properties of the FRT, PeerIDs of C's descendants that are at the $(b - f)\text{th}$ level of the FRT have a prefix XY. If the subspace represented by the node with label XY in the partition tree intersects with ω, descendants of node C in the FRT contain a part of the destination peers, and peer B should forward the query to peer C.

Fig. 10 shows an example of using *MIRA* for multiple-attribute range query in the FRT shown in Fig. 6. In the example in Fig. 10, we set $m = 2$, $k = 4$ and the multiple-attribute space is $< [0, 6], [0, 8] >$ (these parameters are set to be the same as that in Fig. 9), and peer 212 issues a multiple-attribute range query $< [1.2, 1.8], [1, 5] >$. By the $Multiple\_hash$ algorithm, we can get $LowT = 0120$ and $HighT = 0210$. Therefore, the destination peers are all at the third level of the FRT. The dashed lines with arrows in Fig. 10 show the search paths of *MIRA*. The search message is not forwarded to peer 2020 because there is no intersection between its descendants and the destination peers.

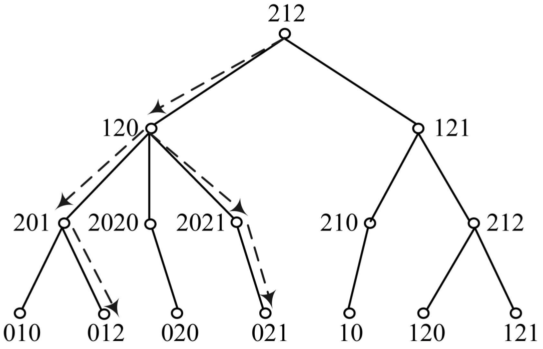

Fig. 10. An example of *MIRA*.

5.3 Discussions

Same as the single-attribute range query scheme, the multiple-attribute range query scheme of Armada can make use of the underlying DHT functionalities to deal with the join/leave/fail of peers. Also, *POBM* and other runtime balancing mechanism introduced in Section 4.4 can be easily extended to the multiple-attribute range query scheme.

The multiple-attribute and single-attribute range query schemes are much similar since they both use the partition tree to help design the naming algorithms and propagate the queries along the FRT. However, the multiple-attribute naming need to reflect the object's attribute value of each dimension, and it generate symbols of ObjectIDs along attributes in a round-robin style. The single-attribute naming is order preserving and the destination peers that intersect a specific single-attribute range query are adjoining peers at certain level of the FRT. While the multiple-attribute naming is not order preserving but partial-order preserving, and the destination peers that intersect a specific multiple-attribute range query may be not adjoining at certain level of the FRT. Those factors lead to the different designs of *PIRA* and *MIRA*.

6 Performance Evaluation

In this section, we evaluate the performance of the algorithms in Armada by both analysis and simulations. As Amada is built on the underlying FissionE DHT [5], the cost of constructing and maintaining the overlay network, as well as publishing the object in Armada, is determined by the underlying FissionE DHT and discussed thoroughly in [5].

6.1 Theoretical Analysis

We first present the lower bounds on query delay and message cost for range queries performed on top of constant-degree DHTs without requiring the modification of the underlying DHTs.

Theorem 3. *In an N-peer P2P system, the lower bounds on the maximum query delay and message cost of general range query schemes on constant-degree DHTs are* $O(\log N)$ *and* $O(\log N) + n - 1$, *respectively (n is the number of destination peers that intersect with the query).*

Proof. It has been shown [30] that the lower bound on the diameter of constant-degree DHT overlays is $O(\log N)$. For any range query, there are some destination peers. The delay of the range query is no less than the number of hops that the query needs to propagate from the initial peer invoking the query to the farthest destination peer. Thus, the maximum delay of range queries is no less than the diameter of the dynamic DHT overlays, and the lower bound on the maximum delay of range queries on top of constant-degree DHTs is $O(\log N)$.

The range query should reach n peers, so its message cost is no less than the sum of the message cost of the query reaching one destination peer and the message cost for reaching the other $n-1$ destination peers. Therefore, the lower bound on maximum message cost for range queries on top of constant-degree DHTs is $O(\log N)+n-1$. □

Next, we analyze the query delay and message cost of Armada.

Theorem 4. *In an N-peer P2P system, the maximum query delay of PIRA or MIRA is less than* $2\log N$ *hops and the average delay of them is less than* $\log N$ *hops.*

Proof. The query delay of *PIRA* or *MIRA* is no more than the height of the FRT, which is equal to the length of the PeerID of the root peer. By the properties of FissionE [5], the maximum length of PeerIDs is always less than $2\log N$ and the average length is less than $\log N$ in the dynamic FissionE system. Thus, Theorem 4 holds. □

From Theorem 4, it can be inferred that both *PIRA* and *MIRA* are *delay-bounded* because they can return all query results within $2\log N$ hops, regardless of the size of the query space or the specific range queried, and their delays both reach the lower bound $O(\log N)$.

Theorem 5 below analyzes the message cost of *PIRA*. In FissionE, the lengths of most PeerIDs are equal. Considering the case that all PeerIDs are of the same length, then the FRT of any peer would be a complete binary tree and its average height is less than $\log N$. In this case, if the destination peers of a range query are n adjoining nodes at mth $(1 \le m < \log N)$ level of the FRT, the operation that *PIRA* performs will be a pruning search for n adjoining leaf nodes in an $(m+1)$-level complete binary tree. Theorem 5 gives the message cost of *PIRA* under this case.

Theorem 5. *Suppose PIRA searches for* n *adjoining leaf nodes in an* $(m+1)$-level *complete binary tree (with the root at the 0th level). When* n *is a power of 2, the average message cost of PIRA is less than* $m+2n-2$.

The proof of Theorem 5 is long, and it is put in Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99. From Theorem 5 and the fact that $1 \le m < \log N$, we can know that the upper bound of *PIRA*'s average message cost is less than $\log N + 2n - 2$ under the case that all PeerIDs are of the same length and n is a power of 2. For the general case, simulations in next section also show that the average message cost of *PIRA* is about $\log N + 2n - 2$, which is close to the lower bound $O(\log N) + n - 1$ on message cost of range queries on constant-degree DHTs.

For *MIRA* in Armada, the destination peers of a multiple-attribute range query may be disjoining nodes at the same level of the FRT and the distribution of their places is difficult to get. We therefore evaluate the message cost of *MIRA* by simulations.

6.2 Simulations

We have implemented Armada on the FissionE simulator [5]. In the simulations, the Armada P2P system is formed by the continuous joining and leaving of peers. Initially, there are only three peers in the system and, as the system size increases, the joining/leaving ratio (i.e., the ratio of the number of joining peers to the number of leaving peers) is set to be 4. Once the system size reaches the predefined value, the joining/leaving ratio is set to be 1. There are two parameters involved in range queries: the number of peers in the system (i.e., network size) and the size of queried range (i.e., range size). We varied these parameters, one at a time, and measured the query delay and message cost. The entire interval of each attribute value is set to [0, 1,000]. For each measurement, the result is averaged over 1,000 range queries that are randomly selected from the interval [0, 1,000] and invoked by a random peer.

Besides the delay and message cost, we also evaluated the following metrics:

1. *Destpeers.* The number of destination peers that intersect with the query. These peers need to query their local information to return query results.
2. *MesgRatio.* Defined as *Messages/Destpeers*, where *Messages* is the total number of messages produced by the query. MesgRatio is used to evaluate the average message cost per destination peer.
3. *IncreRatio.* Defined as

$$(Messages - \log N)/(Destpeers - 1).$$

Similar to MesgRatio, *IncreRatio* is used to evaluate the increasing rate of messages when the number of destination peers increases, excluding the impact of the first destination peer (whose message cost is about $\log N$). *IncreRatio* can also be used to evaluate the analytical results in Section 6.1.

6.2.1 Single-Attribute Range Query

Among the well-known general range query schemes, only PHT [7], *DCF-CAN* [10], and Armada can support single-attribute range queries on constant-degree DHTs. Since the delay and message cost of PHT [7] is much larger than that of Armada (see Section 2.1), we only compared the single-attribute range query scheme of Armada (i.e., *PIRA*) with *DCF-CAN* when the degree of the underlying DHT is equal (i.e., the parameter d is set to be 2 in *DCF-CAN*). The *DCF-CAN* scheme uses CAN [2] as the underlying DHT. When a peer P invokes a range query $[l, u]$ in *DCF-CAN*, it first routes the query to the peer in charge of the median value (i.e., $(l+u)/2$) and then starts two "waves" of propagation. In the first wave, the current peer propagates the query only to the neighbors that intersect the query and have a "higher" interval than the current peer. Then, the current peer propagates the query to the neighbors with a "lower" interval.

Figs. 11 and 12 show the evaluation results about the impact of range size on range queries. In the simulations, the number of peers is set to 2,000 and the range size varies from 2 to 300. From Fig. 11, it can be observed that the query delay of *DCF-CAN* is much larger than that of *PIRA*. When

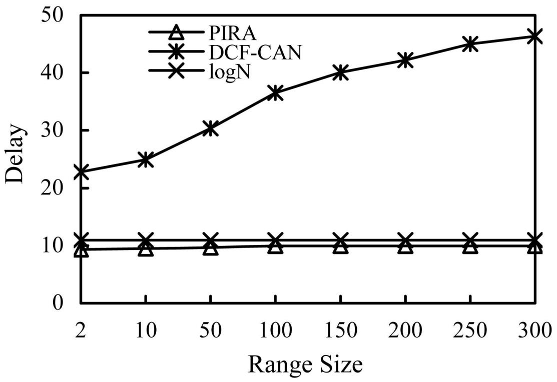

Fig. 11. The impact of range size on query delay.

the size of queried range increases, the average delay of *DCF-CAN* increases remarkably, while *PIRA* is delay-bounded: its average delay is always less than $\log N$, regardless of the range size.

Fig. 12 shows the message cost and related parameters of *PIRA* when the range size varies. From Fig. 12a, it can be observed that the message costs of *PIRA* and *DCF-CAN* are close, and *PIRA* is slightly better. Therefore, *PIRA* can achieve delay-bounded property without imposing overly large message cost. Fig. 12a also shows *Destpeers* of *PIRA*, which is about one half of the number of messages. From Fig. 12b, it can be inferred that *MesgRatio* and *IncreRatio* are close to 2, and *IncreRatio* is almost always no more than 2. From the definitions of these two parameters, we can get that the increase ratio of messages is about twice that of destination peers. Thus, it validates our analytical result about the message cost of *PIRA* in Section 6.1.

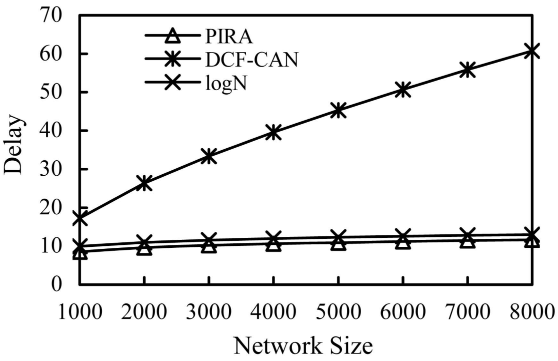

Fig. 13. The impact of network size on query delay.

Figs. 13 and 14 show the evaluation results of the impact of network size on range query. In the simulations, the network size varies from 1,000 to 8,000, and the range size is always 20. From Fig. 13, it can be inferred that the delay of *PIRA* is less than that of *DCF-CAN*, and the advantage of *PIRA* over *DCF-CAN* becomes more remarkable as the network size increases. Fig. 13 also shows that the average delay of *PIRA* is always less than $\log N$ with different values of the network size N. Fig. 14 shows the message cost and related parameters of *PIRA* when the network size varies. From Fig. 14a, it can be observed that the message costs of *PIRA* and *DCF-CAN* are close, and *PIRA* is slightly better than *DCF-CAN*. From Fig. 14b, we can observe that *MesgRatio* and *IncreRatio* are close to 2. And it again validates our analytical result about *PIRA*.

Fig. 15 shows the simulation results of the load balancing property of Armada. In the simulations, the number of peers is 2,000, and 600,000 random objects are generated with attribute values following a Zipf distribution with *PDF*

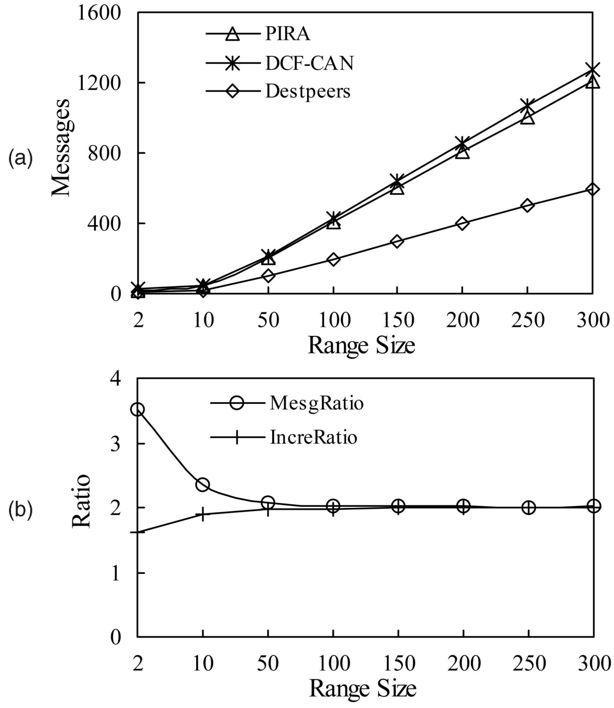

Fig. 12. The impact of range size on message cost. (a) Number of messages. (b) Parameters about message cost of *PIRA*.

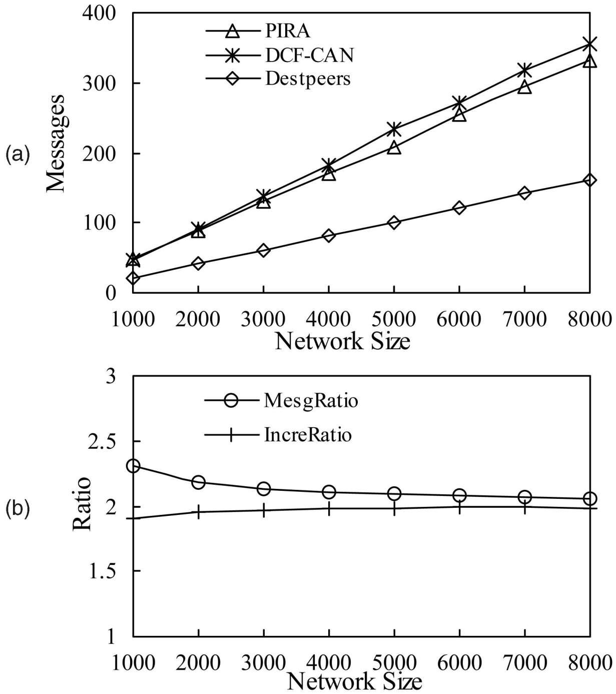

Fig. 14. The impact of network size on message cost. (a) Number of messages. (b) Parameters about message cost of *PIRA*.

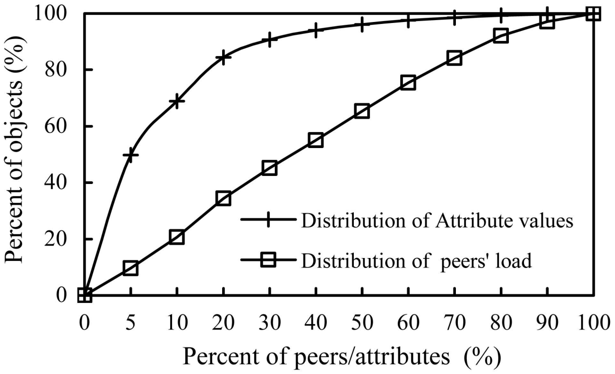

Fig. 15. The load balancing property of Armada.

$\rho(x) = c^* x^{-a}$, $a = 2.5$, $x \in [1, 11]$, where c is a normalized constant. The objects are assigned ObjectIDs by *POBM* and are published onto peers. Then, we calculate the load (the number of objects) of each peer. We sort the peers in decrease order by their load, and, respectively, calculate the total load of first 5 percent, 10 percent, 20 percent, $\ldots$, 90 percent peers. And we deal with the attribute values in the same way. From Fig. 15, it can be inferred that the distribution of peers' load can be almost uniform while the distribution of attribute values is much skewed (50 percent objects have about 5 percent attribute values).

6.2.2 Multiple-Attribute Range Query

Among the well-known general range query schemes, only PHT [7] and Armada can support multiple-attribute range queries on constant-degree DHTs. Since the delay and message cost of PHT is obviously larger than that of Armada (see Section 2.1), we did not compare PHT in the simulations. Armada uses *MIRA* to perform multiple-attribute range queries, and we present the evaluation of *MIRA* when either the range size or the network size varies.

Figs. 16 and 17 show the results of evaluating the impact of the range size on the performance of *MIRA*. In the simulations, the number of peers is 6,000, the number of attributes is 6 (i.e., $m = 6$), and the range size of each attribute varies from 50 to 400. From Fig. 16, it can be observed that the query delay of *MIRA* is always less than $\log N$, regardless of the range size. This demonstrates the delay-bounded property of *MIRA*. Fig. 17 shows the message cost and related parameters of *MIRA* when the range size varies. From Fig. 17b, it can be observed that

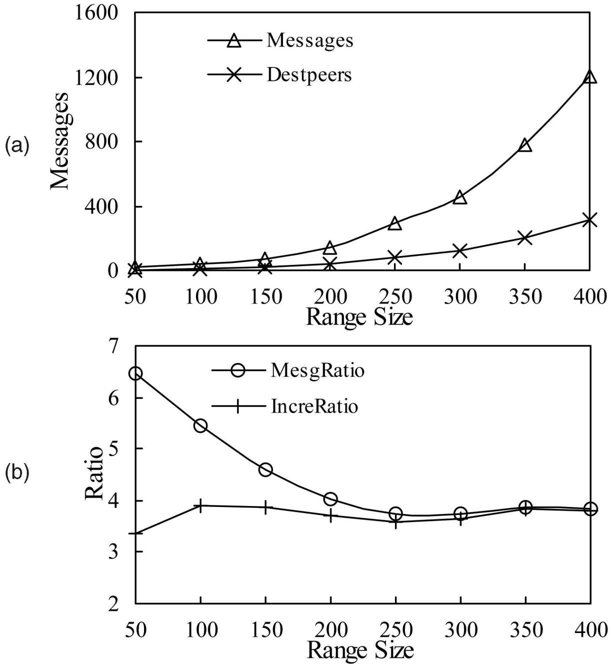

Fig. 17. The impact of range size on message cost of *MIRA*. (a) Number of messages. (b) Parameters about message cost of *MIRA*.

MesgRatio and *IncreRatio* are close to 4. By the definitions of these two parameters, we can see that the increasing rate of message cost is about four times of that of destination peers, and thus, the average message cost of *MIRA* is about $\log N + 4(n-1)$, where n is the number of peers that intersect with the query.

Figs. 18 and 19 illustrate the results of evaluating the impact of network size on the performance of *MIRA*. In the simulations, the network size varies from 1,000 to 8,000, the number of attributes is 6, and the queried range of each attribute is 200. From Fig. 18, it can be inferred that the average delay of Armada is always less than $\log N$ when the network size N varies. From Fig. 19b, it can be observed again that *MesgRatio* and *IncreRatio* are close to 4.

7 Conclusion

In this paper, we have proposed a delay-bounded general range query scheme, called Armada. Built on top of FissionE which is a high-performance constant-degree

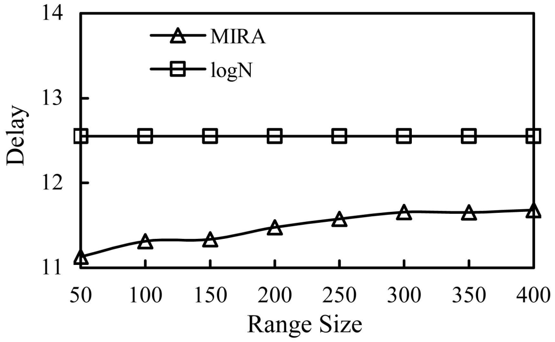

Fig. 16. The impact of range size on query delay.

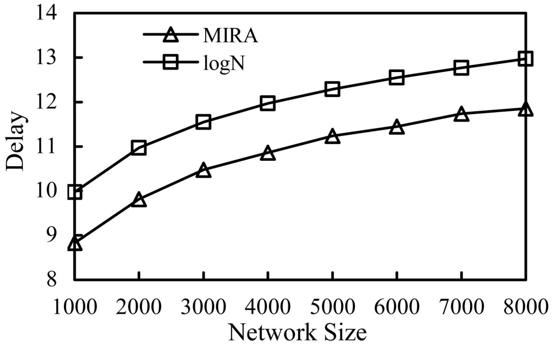

Fig. 18. The impact of network size on query delay.

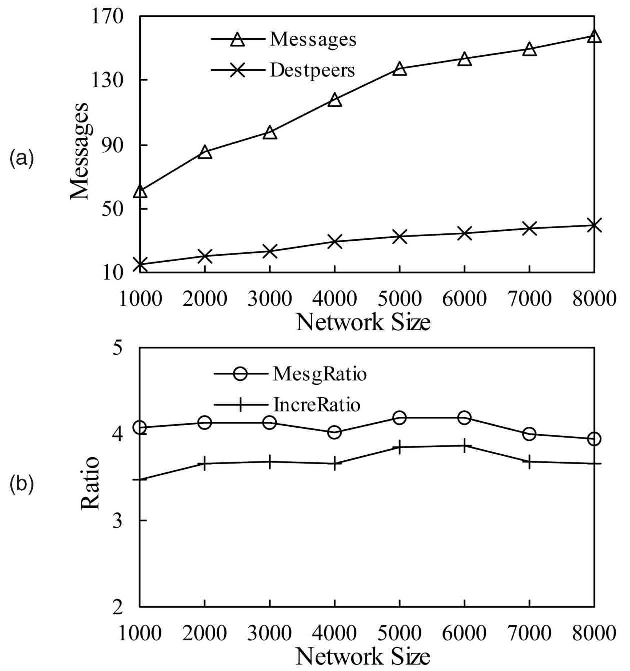

Fig. 19. The impact of network size on message cost of *MIRA*. (a) Number of messages. (b) Parameters about message cost of *MIRA*.

DHT, Armada supports single-attribute and multiple-attribute range queries. Both analytical and simulation results demonstrated that Armada is delay-bounded and highly efficient. The average query delay is less than $\log N$ and the maximum delay is less than $2\log N$, independent of the size of query space and specific queries. The average message cost of single-attribute queries is about $\log N + 2n - 2$ (n is the number of peers that intersect with the query), which is very close to the lower bound on message cost of range queries on constant-degree DHTs. Armada has been used for resource discovery in the iVCE [35] system.

Our ongoing work includes improving the runtime load balancing of Armada and relieving query hotspots. Furthermore, we are extending Armada to support attribute values in various forms and provide other complex query capabilities such as the top-k query and fuzzy query.

Appendix

Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99, shows the proof of Theorem 1. Appendices B and C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99, present the pseudocode of the $Multiple\_hash$ algorithm and *MIRA*, respectively. Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.99, shows the proof of Theorem 5.

Acknowledgments

The authors would like to thank the anonymous reviewers for their critical and constructive comments on this paper. This work was supported in part by the National Natural Science Foundation of China under Grants 60703072 and 60673167, the National Basic Research Program of China (973) under Grant 2005CB321800, the Hunan Provincial Natural Science Foundation of China under Grant 08JJ3125, and the Hong Kong University Grant Council under the CERG Grant PolyU 5103/06E. Some preliminary results of this work were presented at the ICDCS '06 [36].

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking,* vol. 11, no. 1, pp. 17-32, Feb. 2003.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM '01,* pp. 149-160, 2001.

[3] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. Selected Areas in Comm.,* vol. 22, no. 1, pp. 41-53, 2004.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware '01),* pp. 329-350, Nov. 2001.

[5] D.S. Li, X.C. Lu, and J. Wu, "FissionE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs," *Proc. IEEE INFOCOM '05,* pp. 1677-1688, 2005.

[6] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses," *Proc. ACM SIGCOMM '05,* Aug. 2005.

[7] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarcay, S. Shenker, and J. Hellersteinz, "A Case Study in Building Layered DHT Applications," *Proc. ACM SIGCOMM '05,* Aug. 2005.

[8] A. Gupta, D. Agrawal, and A.E. Abbadi, "Approximate Range Selection Queries in Peer-to-Peer Systems," *Proc. First Biennial Conf. Innovative Data Systems Research (CIDR '03),* Jan. 2003.

[9] C. Schmidt and M. Parashar, "Enabling Flexible Queries with Guarantees in P2P Systems," *IEEE Internet Computing,* vol. 8, no. 3, pp. 19-26, 2004.

[10] A. Andrzejak and Z.C. Xu, "Scalable Efficient Range Queries for Grid Information Services," *Proc. Second IEEE Int'l Conf. Peer-to-Peer Computing (P2P '02),* Sept. 2002.

[11] J. Aspnes and G. Shah, "Skip Graphs," *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '03),* pp. 384-393, 2003.

[12] N.J.A. Harvey, M.B. Jone, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties," *Proc. Fourth USENIX Symp. Internet Technologies and Systems (USITS '03),* Mar. 2003.

[13] K.C. Zatloukal and N.J.A. Harvey, "Family Trees: An Ordered Dictionary with Optimal Congestion, Locality, Degree, and Search Time," *Proc. 15th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '04),* pp. 301-310, 2004.

[14] M.T. Goodrich, M.J. Nelson, and J.Z. Sun, "The Rainbow Skip Graph: A Fault-Tolerant Constant-Degree Distributed Data Structure," *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '06),* pp. 384-393, 2006.

[15] P. Ganesan, B. Yang, and H. Garcia-Molina, "One Torus to Rule Them All: Multidimensional Queries in P2P Systems," *Proc. Seventh Int'l Workshop Web and Databases (WebDB '04),* June 2004.

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

[17] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed Resource Discovery on Planetlab with SWORD," *Proc. First Workshop Real, Large Distributed Systems (WORLDS '04),* Dec. 2004.

[18] B. Liu, W.C. Lee, and D.L. Lee, "Supporting Complex Multi-Dimensional Queries in P2P Systems," *Proc. 25th Int'l Conf. Distributed Computing Systems (ICDCS),* 2005.

[19] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram, "P-Tree: A P2P Index for Resource Discovery Applications," *Proc. 13th Int'l World Wide Web Conf. (WWW '04),* May 2004.

[20] C. Zhang, A. Krishnamurthy, and R.Y. Wang, "Brushwood: Distributed Trees in Peer-to-Peer Systems," *Proc. Fourth Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2005.

[21] Y. Shu, B.C. Ooi, K.L. Tan, and A. Zhou, "Supporting Multi-Dimensional Range Queries in Peer-to-Peer Systems," *Proc. Fifth IEEE Int'l Conf. Peer-to-Peer Computing (P2P),* 2005.

[22] J. Aspnes, J. Kirsch, and A. Krishnamurthy, "Load Balancing and Locality in Range-Queriable Data Structures," *Proc. 23rd ACM Symp. Principles of Distributed Computing (PODC '04),* July 2004.

[23] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB),* 2004.

[24] O.D. Sahin, A. Gupta, D. Agrawal, and A.E. Abbadi, "A Peer-to-Peer Framework for Caching Range Queries," *Proc. 20th IEEE Int'l Conf. Data Eng. (ICDE '04),* Apr. 2004.

[25] H.V. Jagadish, B.C. Ooi, Q.H. Vu, R. Zhang, and A. Zhou, "VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes," *Proc. 22nd IEEE Int'l Conf. Data Eng. (ICDE),* 2006.

[26] L. Arge, D. Eppstein, and M.T. Goodrich, "Skip-webs: Efficient Distributed Data Structures for Multi-Dimensional Data Sets," *Proc. 24th ACM Symp. Principles of Distributed Computing (PODC),* 2005.

[27] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, "Space Filling Curves and Their Use in Geometric Data Structures," *Theoretical Computer Science,* vol. 181, pp. 3-15, 1997.

[28] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM,* vol. 18, no. 9, pp. 509-517, Sept. 1975.

[29] J.A. Orenstein and T.H. Merrett, "A Class of Data Structures for Associative Searching," *Proc. Third ACM SIGACT-SIGMOD Symp. Principles of Database Systems (PODS),* 1984.

[30] J. Xu, A. Kumar, and X. Yu, "On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks," *IEEE J. Selected Areas in Comm.,* vol. 22, no. 1, Jan. 2004.

[31] M. Schatzman, *Numerical Analysis: A Mathematical Introduction.* Clarendon Press, 2002.

[32] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks," *Proc. ACM SIGCOMM '03,* Aug. 2003.

[33] B. Awerbuch and C. Scheideler, "Peer-to-Peer Systems for Prefix Search," *Proc. 22nd ACM Symp. Principles of Distributed Computing (PODC),* 2003.

[34] H.T. Shen, Y.F. Shu, and B. Yu, "Efficient Semantic-Based Content Search in P2P Network," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 7, pp. 813-826, July 2004.

[35] X.C. Lu, H.M. Wang, and J. Wang, "Internet-Based Virtual Computing Environment (iVCE): Concepts and Architecture," *Science in China, Series F: Information Sciences,* vol. 49, no. 6, pp. 681-701, Dec. 2006.

[36] D.S. Li, J.N. Cao, X.C. Lu, K.C.C. Chan, B.S. Wang, J.S. Su, H.V. Leong, and A.T.S. Chan, "Delay-Bounded Range Queries in DHT-Based Peer-to-Peer Systems," *Proc. 26th Int'l Conf. Distributed Computing Systems (ICDCS),* 2006.

[37] G.S. Manku, "Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables," *Proc. 23rd ACM Symp. Principles of Distributed Computing (PODC '04),* pp. 197-205, June 2004.

[38] P.B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," *Proc. IEEE INFOCOM '05,* Mar. 2005.

[39] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *Theory of Computing Systems,* vol. 39, pp. 787-804, Oct. 2006.

Dongsheng Li received the BSc and PhD degrees (both with honors) in computer science from National University of Defense Technology (NUDT), Changsha, Hunan, China, in 1999 and 2005, respectively. He was a visiting student at the Hong Kong Polytechnic University in 2005. He is currently an associate professor in the National Lab for Parallel and Distributed Processing, NUDT. His research interests include peer-to-peer computing, grid computing, computer network, and pervasive computing.

Jiannong Cao received the BSc degree in computer science from Nanjing University, Nanjing, China, in 1982 and the MSc and PhD degrees in computer science from Washington State University, Pullman, in 1986 and 1990, respectively. He is currently a professor in the Department of Computing, Hong Kong Polytechnic University (PolyU), Hung Hom, Kowloon, Hong Kong. Before joining PolyU in 1997, he has been on the faculty of computer science in James Cook University, the University of Adelaide in Australia, and the City University of Hong Kong. His research interests include mobile and pervasive computing, wireless networking, parallel and distributed computing, fault tolerance, and distributed software architecture and programming. He has served as an associate editor and a member of editorial boards of several international journals, and also as a chair and a member of organizing program committees for many international conferences. He is a senior member of the IEEE.

Xicheng Lu received the BSc degree in computer science from Harbin Military Engineering Institute, Harbin, China, in 1970. He was a visiting scholar at the University of Massachusetts between 1982 and 1984. He is currently a professor in the College of Computer, National University of Defense Technology, Changsha, Hunan, China. His research interests include distributed computing, computer networks, and parallel computing. He has served as a member of editorial boards of several journals and has cochaired many professional conferences. He is a joint recipient of more than a dozen academic awards, including four First Class National Scientific and Technological Progress Prize of China. He is an academician of the Chinese Academy of Engineering.

Keith C.C. Chan received the BMath degree (with honors) in computer science and statistics and the MASc and PhD degrees in systems design engineering from the University of Waterloo, Ontario, Canada. Before joining the Hong Kong Polytechnic University (PolyU), Hung Hom, Kowloon, Hong Kong, he was with the IBM Canada Laboratory, Toronto, where he was involved in the development of Image and Multimedia software as well as software development tools. He was an associate professor in the Department of Electrical and Computer Engineering, Ryerson Polytechnic University, Ontario in 1993. He is currently a professor and the head of the Department of Computing, PolyU. His research interests include software engineering, data mining, and computational intelligence.