Delay-Bounded Range Queries in DHT-based Peer-to-Peer Systems

Dongsheng Li[1,2], Jiannong Cao[2], Xicheng Lu[1], Keith C. C. Chan[2], Baosheng Wang[1], Jinshu Su[1], Hong-va Leong[2], Alvin T. S. Chan[2]
[1]*School of Computer, National University of Defense Technology, Changsha, China*
[1]*{dsli,xclu,bswang,sjs}@nudt.edu.cn*
[2]*Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong*
[2]*{csdsli,csjcao,cskcchan,cshleong,cstschan}@comp.polyu.edu.hk*

Abstract

Many general range query schemes for DHT-based peer-to-peer (P2P) systems have been proposed, which do not need to modify the underlying DHTs. However, most existing works have the query delay depending on both the scale of the system and the size of the query space or the specific query, and thus cannot guarantee to return the query results in a bounded delay. In this paper, we propose Armada, an efficient general range query scheme to support single-attribute and multiple-attribute range queries. Armada is the first delay-bounded range query scheme over constant-degree DHTs, and can return the results for any range query within 2logN hops in a P2P system with N peers. Results of analysis and simulations show that the average delay of Armada is less than logN, and the average message cost of single-attribute range queries is about logN+2n−2 (n is the number of peers that intersect with the query). These results are very close to the lower bounds on delay and message cost of range queries over constant-degree DHTs.

1. Introduction

Distributed hash table (DHT) [1] based peer-to-peer (P2P) systems such as Chord [2], CAN [3], Tapestry [4], and FISSIONE [5] use a hash table-like interface to publish and lookup objects on distributed peers. Given a query for a specific *key*, DHTs can efficiently locate the peer which owns the object with the keyword key. DHT-based P2P systems have proven to be scalable, robust, efficient and generally applicable. As a result, the DHT has become a general infrastructure for building many P2P applications, such as distributed storage systems, naming services, data management systems, and large-scale online games.

The basic functionality supported by the DHT infrastructure is exact-match query, which is enough for many applications. For example, a P2P storage system can use the exact-match query interface with the filename as the keyword to publish and lookup files. However, the ever wider use of DHT infrastructures has found applications that require support for *range queries* [6-20]. Examples of range query include the query "$70 \leq score \leq 80$" in P2P data management systems and the query "$1GB \leq Memory \leq 4GB$ and $50GB \leq disk \leq 200GB$" in grid information services.

A number of range query schemes [6-20] have been proposed for DHT-based P2P systems. An approach to build the range query support is the *general range query scheme* [9-10], which is built entirely over 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 [10,21], providing the range query functionality 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 systems (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 N. When the query space or the queried range is large, the query execution can be very slow.

In this paper, we present Armada, an efficient, delay-bounded general range query scheme. Armada operates over FISSIONE [5], a high-performance constant-degree DHT scheme previously proposed by the authors [5], and does not need to modify the underlying FISSIONE infrastructure. Armada provides

Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)
0-7695-2540-7/06 $20.00 © 2006 IEEE

support for efficient single-attribute and multiple-attribute range query and can return all query results in a bounded delay, independent of the size of the query space or the queried range. Results of our analysis and simulation studies show that Armada can achieve high efficiency in range query processing. Armada can return all query results within 2log$N$ 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 $\log N+2n-2$ (n is the number of peers that intersect with the query), which is very close to its asymptotic lower bound $O(\log N)+n-1$. To our knowledge, Armada is the first delay-bounded range query scheme over constant-degree DHTs.

The remainder of the paper is organized as follows. Section 2 introduces the related work. Section 3 briefly describes FissionE as the background of this paper. Sections 4 and 5 present the design of the single-attribute and multiple-attribute range query schemes in Armada, respectively. Section 6 concludes the paper with future work.

2. Related work

Range query schemes for DHT-based systems can be categorized as either general or customized schemes. General range query schemes [7-13] are entirely layered over existing DHTs and do not modify the underlying DHT. Customized schemes [14-20] either make use of custom-designed DHTs or add specific modifications to the behavior of the underlying DHTs. In this paper, we focus on general range query schemes.

Gupta *et al.* [7] proposes a probability scheme that uses locality sensitive hashing to support single-attribute range queries on Chord. However, it can only return approximate results.

Schmidt *et al.* proposes Squid [8] to provide multiple-attribute range query functionality on Chord. Squid uses a space-filling curve (SFC) to map objects with multiple attributes to peers and performs 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$N$.

Skip Graph [11] and SkipNet [12] are DHT schemes that can directly support single-attribute range queries, but have query delays of $O(\log N+n)$, which depends on the sizes of specific queries. SCRAP [13] uses the space-filling curve to support multiple-attribute range queries on Skip Graph, but its query delay remains to be $O(\log N+n)$.

All the schemes described above are based on DHTs with $O(\log N)$ degree. Among the existing general range query schemes, only Armada and the works reported in [9, 10] are range query schemes that can run over constant-degree DHTs. Among the three schemes, only 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 other two.

Andrzejak and Xu [9] propose a single-attribute range query scheme based on CAN. For any range query, the scheme first routes the query to the peer in charge of the median value of the query, and then floods the query to its neighbors until all related peers are visited. The scheme compares three flooding 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.

Chawathe *et al.* [10] designs PHT to support both single-attribute and multiple-attribute range queries on any DHT (including constant-degree DHTs). PHT builds a prefix hash tree in which leaf nodes are keys and every internal node corresponds to a distinct key prefix. Range query in PHT is performed by parallel search in the prefix hash 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.

Table 1 shows the comparisons of the general range query schemes described above.

Many customized range query schemes have also been proposed in recent years. Mercury [16] and SWORD [17] provide multiple-attribute range queries by indexing the data set along each individual attribute. Liu *et al.* [18] proposes *NR-tree,* which support range queries and k-nearest neighbor queries in super-peer P2P systems. MURK [13] and P-tree [19] build specific P2P networks to support range queries, respectively based on KD-tree and B^+-tree. Brushwood [14] provides the multiple-attribute range query functionality on Skip Graph. Aspnes *et al.* [15] and Ganesan *et al.* [20] respectively propose some mechanisms to improve the load balance of Skip Graph.

However, these schemes design specific P2P networks or need to make specific modifications to the underlying DHTs.

Table 1. Comparisons of some general range query schemes

Schemes	Underlying DHT	Degree of underlying DHT	Query functionality		Average delay	Delay bounded?
			Singe attribute	Multiple attribute		
Squid [8]	Chord	$O(\log N)$		√	$O(h*\log N)$ +	No
Skip Graph, SkipNet [11,12]	---	$O(\log N)$	√		$O(\log N+n)$	No
SCRAP [13]	Skip Graph	$O(\log N)$		√	$O(\log N+n)$	No
DCF-CAN [9]	CAN	d	√		$> O(N^{1/d})$	No
PHT [10] &	Any DHT	d	√	√	$O(b*\log N)$ +	No
Armada [this paper]	FissionE	4	√	√	$< \log N$	Yes

& This row shows the performance of PHT with only constant-degree DHT

+ h and b are two parameters related to the size of the query space and the specific query

3. Overview of FISSIONE

In this section, we give an introduction to FISSIONE on which Armada is built. FISSIONE [5] is a constant 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 of length k and base d is defined as a string $a_1a_2...a_k$ where $a_j \in \{0,1,2,...,d\}$ $(1 \le j \le k)$ and $a_i \ne a_{i+1}$ $(1 \le i \le k\text{-}1)$, 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 $\alpha \in \{0,1,2,...,d\}$ and $\alpha \ne u_k$, node $U=u_1u_2...u_k$ has one out-edge to node $V=u_2u_3...u_k\alpha$ (denoted by $U \rightarrow V$). Figure 1 shows a Kautz graph $K(2,3)$.

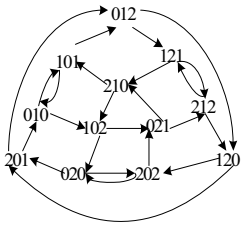


Figure 1. Kautz graph *K*(2,3)

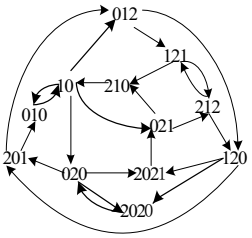


Figure 2. An example of FISSIONE topology

In FISSIONE, the identifiers (i.e., PeerIDs) of peers are Kautz strings with base 2. The lengths of PeerIDs 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 $U=u_1u_2...u_k$ are in the style of $u_2u_3...u_kq_1...q_m$ with $0 \le m \le 2$. Figure 2 shows an example of the P2P network topology in FISSIONE.

Each object in FISSIONE is assigned an ObjectID by a naming algorithm *Kautz_hash*, which are Kautz strings with fixed length k (generally k=100). Each object is published on a unique peer whose PeerID is a prefix of its ObjectID. FISSIONE adopts effective self-stabilization and fault-tolerant mechanisms to deal with the joining or departing of peers. Analysis and simulations show that FISSIONE is a constant-degree and high-efficiency DHT scheme. The average degree of FISSIONE is 4, its diameter is less than $2\log N$, and its average routing delay is less than $\log N$.

4. Single-attribute range queries

Like many other DHTs, FISSIONE provides support for scalable and efficient exact-match query of distributed objects on peers. However, it can not support range queries for numeric attribute values. Therefore, we have designed Armada to support single-attribute and multiple-attribute range queries over FISSIONE.

The basic components of Armada include two parts: *object naming* and *range query processing*. Armada first uses an order-preserving naming algorithm 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 returns query results within a bounded delay.

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

4.1 Single-attribute naming

In this subsection, we propose an order-preserving naming algorithm *Single_hash* to assign to objects with close 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 the paper, we assume that the entire interval of attribute values of objects is a real-number interval [L, H] and use symbol $\preceq$ to denote the relation "no more than" between Kautz strings in the lexicographical order. Below we give some definitions of order-preserving naming.

Definition 1. The *Kautz region* $[\![\alpha, \beta]\!]$ is defined as: $[\![\alpha, \beta]\!] = \{ s \mid s \in KautzSpace(2,k)$ and $\alpha \preceq s$ and $s \preceq \beta \}$. For example, Kautz region $[\![010, 021]\!] = \{010, 012, 020, 021\}$.

Definition 2. Assume F is an surjection function from a real-number interval D to Kautz namespace V. F is an *interval-preserving* function, if and only if for any subinterval [a, b] of D, the corresponding range of [a, b] by function F is Kautz region $[\![F(a), F(b)]\!]$.

We proposes a *partition tree* $P(2,k)$ model to help design of 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 it to the root. Figure 3 shows an example of the partition tree $P(2,4)$. 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 $\preceq$.

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 Figure 3, the entire interval of attribute values is [0, 1]. Nodes A, B and C are children of the root and evenly partition the interval [0, 1]. And node U with label 0101 represents the subinterval [0, 1/24].

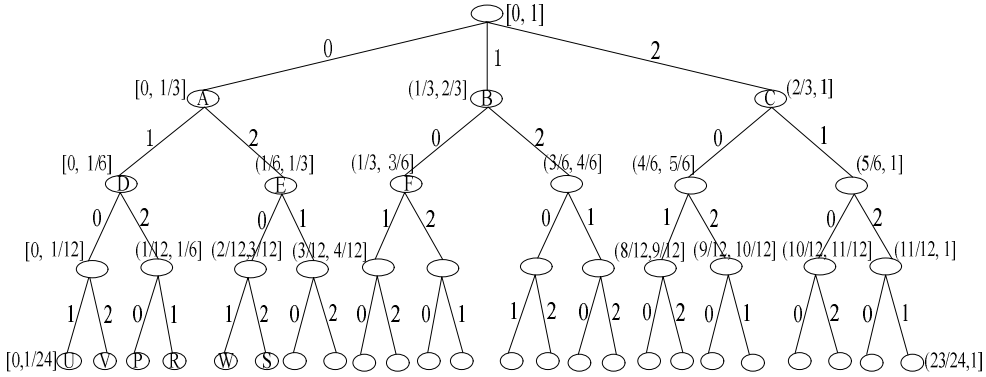


Figure 3. Partition tree $P(2,4)$ for attribute value interval [0, 1]

In the partition tree $P(2,k)$, nodes at the same level represent subintervals with the same size, whose union is the entire interval [L, H]. 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* 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 Kautz string S. Then S is assigned as the ObjectID of object O, i.e., $Single\_hash(c,L,H,k) = S$. In the example shown in Figure 3, 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 whose attribute value is 0.1. Due to the limit in space, the pseudocode of *Single_hash* [23] is omitted here.

It can be proved that [23] the *Single_hash* algorithm is an interval-preserving function from [L, H] to

KautzSpace(2,*k*). In the example shown in Figure 3, the range of [0.1, 0.24] by the *Single_hash* algorithm is the Kautz region ⟦ 0120, 0202 ⟧, which contains the four adjoining leaf nodes *P*, *R*, *W* and *S*.

4.2 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*), *HighT* = *Single_hash*(*HighV*,*L*,*H*,*k*). Because the *Single_hash* algorithm is interval-preserving, objects with attribute values in the range [*LowV*, *HighV*] are published exactly on 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 $P=u_1u_2...u_b$ are $u_2...u_bv_1...v_q$ ($0 \le q \le 2$). Based on the topology properties of FISSIONE, we can design a *forward routing tree* (FRT) for any peer *P*. The forward routing tree of peer $P=u_1u_2...u_b$ 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 tree, 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 $\preceq$ defined over PeerIDs; (4) The FRT has (*b*+1) levels with the root node at the 0th level. Therefore, the *i*th level ($0 \le i \le b-1$) of the FRT contains all the peers whose PeerIDs have a prefix $u_{i+1}...u_b$ and the last level (*b*th level) contains all the peers whose PeerIDs do not have u_b as the first symbol. Figure 4 shows the FRT of peer 212 for the FISSIONE topology shown in Figure 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 suffixes of 212.

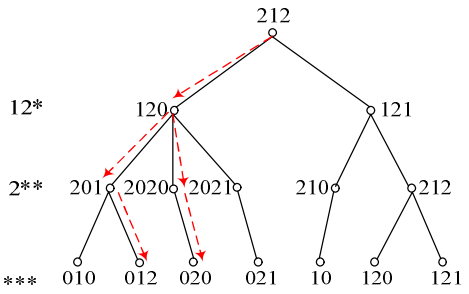


Figure 4. An example of the FRT tree

Based on the FRT, Armada uses *PrunIng Routing Algorithm* (*PIRA*) to perform a pruning search in the FRT for all the destination peers that are in charge of 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*, *HighT* ⟧ into several (at most three) sub-regions with common prefixes and deal with each sub-region 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*)th level of the FRT.

When a peer *B* at the *i*th ($0 \le i \le b-1$) level of the FRT receives the search message, the PeerID of *B* is $u_{i+1}...u_{b-f}X$. Consider any out-neighbor $C=u_{i+2}...u_{b-f}XY$ of peer *B*, peer *C* is at the (*i*+1)th level of the FRT. By the properties of the FRT, PeerIDs of *C's* descendants at the (*b*−*f*)th level of the FRT have a prefix *XY*. If the Kautz region ⟦ *LowT*, *HighT* ⟧ includes a Kautz string *S* that has a prefix *XY*, descendants of *C* in the FRT contains part of the destination peers and peer *B* should forward the search message to peer *C*. The pseudocode of the *PIRA* algorithm is omitted here and can be found in [23]. From the above discussion, it is easy to see that the *PIRA* Algorithm can forward any single-attribute range query exactly to all the destination peers that intersect with the query.

The dashed lines with arrows in Figure 4 show an example of search paths of the *PIRA* algorithm. In the example, peer 212 issues a range query [0.1, 0.24]. Since *k*=4, we have *LowT*=0120 and *HighT*=0202. All the destination peers are at the 3rd level of the FRT.

4.3 Performance evaluation

4.3.1 Lower bounds analysis. We derive the lower bounds on query delay and message cost for range queries over constant-degree DHTs without requiring modifications of the underlying DHTs. It has been shown [22] that the lower bound on delay for routing in constant-degree DHTs is $O(\log N)$. Because a range query should reach no less than one destination peer, its query delay is no less than the delay lower bound of one DHT routing. Thus the delay lower bound of range queries over constant-degree DHTs is $O(\log N)$.

Let *n* be the number of destination peers that intersect with the specific range query. The range query should reach *n* peers, and thus its message cost is no less than the sum of the message cost of one DHT routing to reach the first destination peer and the message cost to reach the other *n*−1 destination peers. Therefore, the lower bound on message cost for range queries over constant-degree DHTs is $O(\log N)+n-1$.

4.3.2 Analysis on *PIRA*. First, we analyze the query delay of the *PIRA* algorithm. The query delay of Armada is no more than the height of the forward routing tree (FRT), which is equal to the length of the root's PeerID. By the properties of FISSIONE, the maximum length of PeerIDs is less than $2\log N$ and the average length is less than $\log N$. Thus the maximum query delay of *PIRA* is less than $2\log N$ hops and the average delay is less than $\log N$ hops, i.e., *PIRA* can return all query results within $2\log N$ hops, regardless of the size of the query space or the specific query. Therefore, *PIRA* is *delay-bounded* and its delay reaches the delay lower bound $O(\log N)$.

Next, we analyze the message cost of the *PIRA* algorithm. The average message cost of *PIRA* is about $\log N+2n-2$, which is close to the message cost lower bound $O(\log N)+n-1$ for range queries on constant-degree DHTs. The detail of the analysis is omitted here due to the limit in space and can be found in [23].

4.3.3 Simulations. We have implemented the single-attribute range query scheme of Armada in the FISSIONE simulator [5]. Among the well-known general range query schemes, only *DCF-CAN* [9], PHT [10] and Armada can support single-attribute range queries over constant-degree DHTs. Since the delay and message cost of PHT is much larger than that of Armada (see Section 2), we only compared the *PIRA* algorithm in Armada with *DCF-CAN* [9] (the average degree of the underlying DHT is 4) in this subsection.

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

(a) *Destpeers*: the number of destination peers that intersect with the query. These peers need to query their local information to return query results.

(b) *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.

(c) *IncreRatio*: defined as (*Messages*−log*N*) / (*Destpeers*−1). Similar to *MesgRatio*, *IncreRatio* is used to evaluate the increasing rate of message cost 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 analysis results in Section 4.3.2.

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. We varied these parameters, one at a time, and measured the query delay and message cost. In the simulations, the entire interval of each attribute value is set to [0, 1000]. For each measurement, the result is averaged over 1000 range queries that are randomly selected from the interval [0, 1000] and invoked by a random peer.

Figures 5 and Figure 6 show the evaluation results about the impact of range size on range queries. In the simulations, the number of peers is set to 2000 and the size of range query varies from 2 to 300. From Figure 5, it can be observed that the query delay of *DCF-CAN* is much larger than that of *PIRA*. When the size of range query increases, the average delay of *DCF-CAN* increases remarkably, while *PIRA* is delay-bounded: its average delay is almost unchanged and always less than log*N* no matter the size of range query.

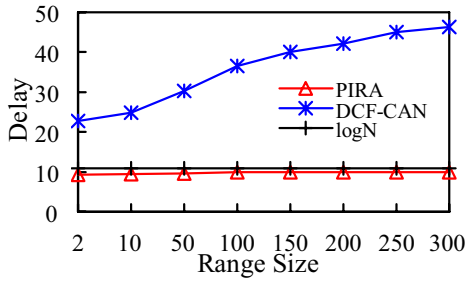
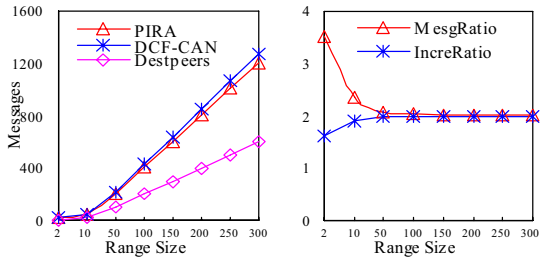


Figure 5. Query delay at different range size



(a) Number of messages (b) Related parameters

Figure 6. Messages at different range size

Figure 6 shows the message cost and related parameters of *PIRA* when the range size varies. From Figure 6(a), it can be observed that the message cost 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. Figure 6(a) also shows *Destpeers* of *PIRA*, which is about one half of the number of messages. From Figure 6(b), it can be inferred that *MesgRatio* and *IncreRatio* are close to 2 and *IncreRatio* is almost always no more than 2. By the definition of these two parameters, we can see that the increase ratio of messages is about twice that of destination peers. Thus it validates our analytical result about the message cost in Section 4.3.2.

Figure 7 and Figure 8 show the evaluation results about the impact of network size on range query. In the

simulations, the network size varies from 1000 to 8000 and the range query size is always set to 20. From Figure 7, 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. Figure 7 also shows that the average delay of *PIRA* is always less than log$N$ with different values of the network size N.

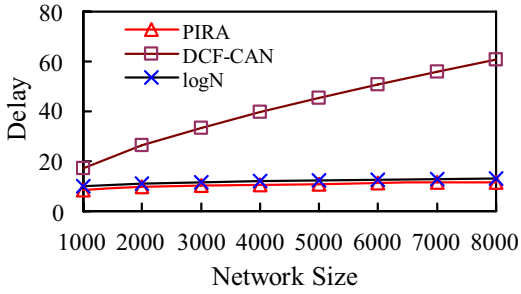
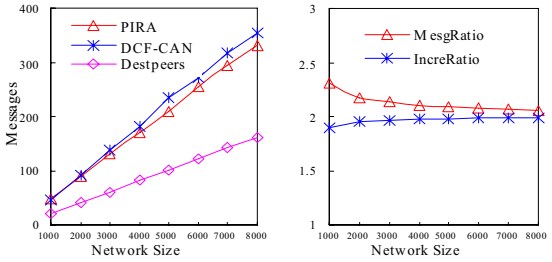


Figure 7. Query delay at different network size

Figure 8 shows the message cost and related parameters of *PIRA* when the network size varies. From Figure 8(a), it can be observed that the message cost of *PIRA* and *DCF-CAN* are close and *PIRA* is slightly better than *DCF-CAN*. From Figure 8(b), we can observe that *MesgRatio* and *IncreRatio* are close to 2. Thus it again validates our analytical result about *PIRA* described in Section 4.3.2.



(a) Number of messages (b) Related parameters

Figure 8. Messages at different network size

5. Multiple-attribute range queries

Due to the limit in space, here we only sketch the multiple-attribute range query scheme in Armada. Interested reader can find the details in [23]. Assume that there are m attributes A_0, A_1, ..., and A_{m-1} and the entire value interval of attribute A_i is $[L_i, H_i]$. We define partial-order relation $\lhd$ between multiple-attribute values as follows.

Definition 3. For two multiple-attribute values $\delta_1 = < u_0, u_1, ..., u_{m-1} >$ and $\delta_2 = < v_0, v_1, ..., v_{m-1} >$ in multiple-attribute space, $\delta_1 \lhd \delta_2$ if and only if, for each $0 \leq i < m-1$, $u \leq v_i$.

Definition 4. Assume that F is a surjection function from multiple-dimensional space D to 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 $F(\delta_1) \preceq F(\delta_1)$.

We again use the partition tree to design a multiple-attribute partial-order preserving algorithm, called *Multiple_hash*. We partition the entire multiple-attribute space $< [L_0, H_0], ..., [L_i, H_i], ..., [L_{m-1}, H_{m-1}] >$ onto the partition tree along attributes A_0, A_1, ... , 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 multiple-attribute space $< [L_0, H_0], ..., [L_i, H_i], ..., [L_{m-1}, H_{m-1}]$. For any node B at the jth level of the partition tree that has f child nodes, let i denote the value of j mod m. Then, the subspace ω represented by node B is evenly divided into f pieces along the ith attribute (i.e., attribute A_i), and each of its f child nodes represents one such a piece.

Based on the partition tree for multiple-attribute space, the *Multiple_hash* algorithm works as follows. For any object O with the multiple-attribute value $V = <v_0, v_1, ..., v_{m-1}>$, V is surely in a subspace 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. It is easy to see [23] that the *Multiple_hash* algorithm is a multiple-attribute partial-order preserving function from $< [L_0, H_0], ..., [L_i, H_i], ..., [L_{m-1}, H_{m-1}] >$ to the *KautzSpace*$(2,k)$.

Suppose a peer $P=u_1u_2...u_b$ issues a multiple-attribute range query $\omega = < [x_0, y_0], ..., [x_i, y_i], ..., [x_{m-1}, y_{m-1}] >$. Let δ_1 denote $<x_0, x_1, ..., x_{m-1}>$ and δ_2 denote $<y_0, y_1, ..., y_{m-1}>$, and let $LowT$ = *Multiple_hash*(δ_1) and $HighT$ = *Multiple_hash*(δ_2). *Multiple_hash* does not have the interval-preserving property that *Single_hash* has, thus the range of a range query ω by *Multiple_hash* may be only a proper subset of $[\![LowT, HighT]\!]$. 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 forward routing tree (FRT) of peer $P=u_1u_2...u_b$. However, when forwarding the query to a node in the FRT, *PIRA* only needs to determine the relation between its PeerID and Kautz region $[\![LowT, HighT]\!]$, while *MIRA* needs to determine whether some descendants of the node in the FRT intersect with the real query ω.

Similar to *PIRA*, the delay of *MIRA* is no less than the height of the FRT, which is equal to the length of the root's PeerID. Therefore, *MIRA* is also delay-bounded because its average delay is less than log$N$

and the maximum delay is less than 2log$N$, regardless of the size of the query space or the specific query.

6. Conclusions and future work

In this paper, we have proposed a delay-bounded general range query scheme, called Armada. Built over FISSIONE, a high-performance constant-degree DHT scheme, Armada supports efficient single-attribute and multiple-attribute range queries. Analysis and simulations are carried out to evaluate the performance of Armada and the results show that Armada can achieve high efficiency.

For future work, we plan to extend Armada to support other complex queries, such as top-k query.

Acknowledgement

This work is supported in part by the National Basic Research Program of China (973) under Grant 2005CB321801, the National Natural Science Foundation of China under Grants 90412011 and 90104001, the Hong Kong Polytechnic Universities under the research grant 4-4961, and the University Grant Council of Hong Kong under the CERG grant PolyU 5183/04E.

References

[1] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, "Looking Up Data in P2P Systems," *Communications of the ACM*, Vol. 46, No. 2, pp. 43-48, 2003.

[2] Ion Stoica, Robert Morris, David Liben-Nowell, et al., "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17-32, February 2003.

[3] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM 2001*, New York, USA, pp. 149–160, 2001.

[4] Ben Y. Zhao, Ling Huang, Jeremy Stribling et al., "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE JSAC*, Vol. 22, No. 1, pp. 41-53, 2004.

[5] Dongsheng Li, Xicheng Lu, Jie Wu, "FISSIONE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs," *Proc. IEEE INFOCOM 2005*, Miami, Florida, USA: 1677~1688.

[6] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, Ion Stoica, "Complex Queries in DHT-based Peer-to-Peer Networks," *Proc. IPTPS'02*, Cambridge, USA, 2002.

[7] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi, "Approximate Range Selection Queries in Peer-to-Peer Systems," *Proc. CIDR'03,* Asilomar, California, USA, January 2003.

[8] Cristina Schmidt, and Manish Parashar, "Enabling Flexible Queries with Guarantees in P2P Systems," *IEEE Internet Computing*, Vol. 8, No. 3, pp. 19-26, May/June 2004.

[9] Artur Andrzejak and Zhichen Xu, "Scalable Efficient Range Queries for Grid Information Services," *Proc. IEEE P2P'2002*, Linköping, Sweden, September 2002.

[10] Yatin Chawathe, Sriram Ramabhadran, et al., "A Case Study in Building Layered DHT Applications," *Proc. SIGCOMM'05*, Philadelphia, Pennsylvania, USA, August 2005.

[11] J. Aspnes and G. Shah, "Skip graphs," *Proc. SODA'03*, Philadelphia, PA, USA, pp. 384-393, 2003.

[12] N. J. A. Harvey, M. B. Jone, S. Saroiu, et al., "Skipnet: A Scalable Overlay Network with Practical Locality Properties," *Proc. USITS'03*, Seattle, WA, USA, March 2003.

[13] Prasanna Ganesan, Beverly Yang, Hector GarciaMolina, "One Torus to Rule them All: Multidimensional Queries in P2P Systems," *Proc. WebDB'04*, June 1718, 2004, Paris, France,2004.

[14] Chi Zhang, Arvind Krishnamurthy, Randolph Y. Wang, "Brushwood: Distributed Trees in Peer-to-Peer Systems," *Proc. IPTPS'05*, New York, USA, 2005.

[15] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy, "Load balancing and locality in range-queriable data structures," *Proc. PODC'04*, Newfoundland, Canada, July 2004.

[16] Ashwin R. Bharambe, Mukesh Agrawal, Srinivasan Seshan, "Mercury: Supporting Scalable Multi-attribute Range Queries," *Proc. SIGCOMM'04*, Portland, Oregon, USA, 2004.

[17] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed Resource Discovery on Planetlab with SWORD," *Proc. WORLDS'04*, Santa Fe, New Mexico, USA, December 2004.

[18] Bin Liu, Wang-Chien Lee, Dik Lun Lee, "Supporting Complex Multi-dimensional Queries in P2P Systems," *Proc. ICDCS'05*, Ohio, USA, 2005.

[19] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, Jayavel Shanmugasundaram, "PTree: A P2P Index for Resource Discovery Applications," *Proc. WWW'04*, New York, USA, May 2004.

[20] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online balancing of range-partitioned data with applications to peer-to-peer systems," *Proc. VLDB'2004*, Toronto, Canada, 2004.

[21] Sean Rhea, Brighten Godfrey, Brad Karp, et al., "OpenDHT: A Public DHT Service and Its Uses," *Proc. SIGCOMM'05*, Philadelphia, Pennsylvania, USA, August, 2005.

[22] Jun Xu, Abhishek Kumar, Xingxing Yu, "On the Fundamental Tradeoffs between Routing Table Size and Network Diameter In peer-to-peer Networks," *IEEE JSAC*, Vol. 22, No. 1, January 2004.

[23] Dongsheng Li, Jiannong Cao, Xicheng Lu, et al., "Delay-bounded range queries over FISSIONE," Technical Report PolyuTech-200503, the Hong Kong Polytechnic University, March, 2005.