RAQ: A Range-Queriable Distributed Data Structure

Hamid Nazerzadeh
Microsoft Research Lab
New England, U.S.A
hamidnz@microsoft.com

Mohammad Ghodsi*
Computer Engineering Department,
Sharif University of Technology, Tehran, Iran
IPM School of Computer Science,Tehran, Iran
ghodsi@sharif.edu

October 5, 2010

Abstract

Different structures are used in peer-to-peer networks to represent their inherently distributed, self-organized, and decentralized memory structure. In this paper, a simple range-queriable distributed data structure, called RAQ, is proposed to efficiently support exact match and range queries over multi-dimensional data. In RAQ, the key space is partitioned among the network with n nodes, in which each element has links to $O(\log n)$ other elements. We will show that the look-up query for a specified key can be done via $O(\log n)$ message passing. Also, RAQ handles range-queries in at most $O(\log n)$ communication steps.

1 Introduction

Distributed and peer-to-peer networks are significant components of recent research on networking. There is a simple idea behind the peer-to-peer networks: each node maintains its own index and searching mechanism compared to the traditional client-server architecture with global information. The significant growth in the scale of such networks, (e.g. Gnutella [4]), reveals the critical emerging need to develop decentralized searching methods. A peer-to-peer storage system can be considered as a large scale distributed decentralized data structure. We use the term *Queriable Distributed Data Structure* (QDS) to denote such a self-organized, decentralized, distributed, internet-scale structure which provides searching and data transferring services. New file sharing systems such as Scour, FreeNet, Ohaha, Kazaa and Jungle Monkey are examples QDS from current internet systems.

In QDS, every node of the network is an element of the whole structure, which provides decentralized searching services over the data scattered among the nodes of the network.

Distributed Hash Table (DHT) [15, 11] can be viewed as a QDS. In DHT systems, keys and data are stored in the nodes of the network using a *hash function*, in which data can be the addressing information (e.g. IP address of the server containing the data), rather than its actual data. Searching mechanism in these systems consists of two main phases: (1) hashing the key, and (2) querying the network to find the node that contains the key. This node handles the query by providing the actual data or its addressing information.

*This author's work has been partially supported by IPM School of CS (contract: CS1388-2-01).

Similarly, some other systems like *SkipNet* [6] are designed based on more theoretical data structures like *skip graphs* [2], allows more flexibility on the location of the data on the nodes of the network.

In this paper, we propose RAQ, a range-queriable distributed data structure to handle exact match and range queries over multi-dimensional data efficiently. In RAQ, the key space is partitioned among the n nodes of the network, in which each element has links to $O(\log n)$ other elements of the network. We will show that the look-up query for a specified key can be done, in our structure, via $O(\log n)$ message passing. The bound on the out-degree of the nodes and the exact-match query cost are both comparable to those in DHT systems like Chord [15], CAN [11], Pastry [13] and Viceroy [9].

The main contribution of RAQ is that it is simple and can handle range-queries in multi-dimensional space. Our data structure supports such queries in at most $O(\log n)$ communication steps. *Split the Space, Duplicate the Query* is a novel approach used by the RAQ to resolve range-queries. This method anticipates the answer space of the query at the source and spreads the query only through the appropriate nodes by duplicating the query meanwhile each of the new queries addresses a reduced subspace.

Most other QDS systems do not support multi-dimensional range-queries, because they mostly use one-dimensional key space. CAN [11] supports multi-dimensional key space, but despite of its similarity to RAQ's basic structure, the out-degree of node and its routing cost depend on the dimension of the key space. For a d dimensional space, the average routing path length in CAN is $(d/4)(n^{1/d})$ hops and individual nodes maintain $2d$ neighbors. This limitation forces the system to use hashing to reduce the dimension. But, since hashing destroys the logical integrity of the data, such systems cannot support range queries over multi-dimensional data efficiently.

In this paper, we first overview the related works briefly, followed by the principal ideas and structures of RAQ. Query handling methods are discussed in sections 5 and 6, followed by the algorithms for joining and leaving a node.

2 Related Works

Supporting range queries in QDS systems has been the subject of several recent works. In SWAM [3], for example, the key space is partitioned according to *Voronoi Tesselation*. By this property, and using links based on *Small-World Phenomenon* [8, 10], SWAM resolves k-nearest-neighbor search and range queries via $O(\log n + R)$ message passing, where R is the size of the answer. But, the number of links of each node grows exponentially with the dimension size [14].

Prefix Hash Tree is a solution proposed by Rantasamy *et. al* [12] to face the problem of hashing used in DHTs that destroys the integrity of the data. Their approach is based on distributed *trie*. Given a query, this system attempts to recognize the longest prefix of the query that appears as a trie-node. The complexity of this operation is $O(\log \log d \times \log n)$, where d is the size of the discrete domain.

Recently, some authors have addressed the problem from the load balancing viewpoint. Karger and Ruhl [7], offer an approach to overcome the known limitation of hash functions, by designing an algorithm to maintain load balance in not-uniform distributed key space. Aspens *et al.* [1] have proposed a similar solution based on *skip graphs*.

Gao and Steenkiste [5], present a QDS which relies on a logical tree data structure, the Range Search Tree (RST), to support range queries in one dimensional space. In this system, nodes in RST are registered in groups. To handle the range queries, queries are decomposed into a small number of sub-queries where the cost depends on the load factor of the data and query capacity of the nodes in the network.

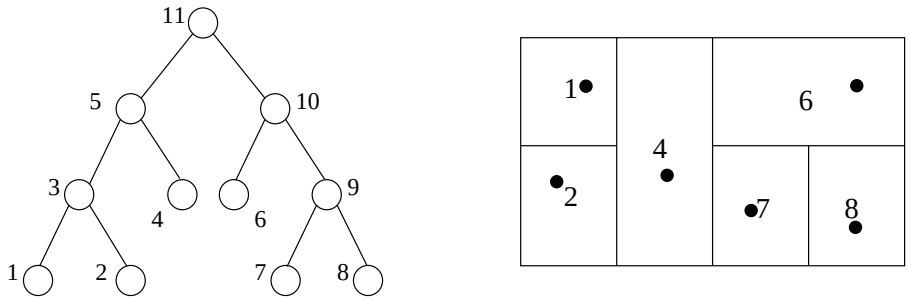

Figure 1: The partition tree, $\mathcal{P}_2$, on the left corresponds to the points on the right.

3 Partition Tree

We have n points in our d-dimensional space. *Partition Tree*, $\mathcal{P}_d$, is the main data structure used in RAQ. Similar to the data structure used in [11], $\mathcal{P}_d$ partitions the d-dimensional space, so that in the final level, each region has only one point. Each internal vertex of the tree corresponds to a region in space, and the root represents the whole space. Each pair of the sibling vertices divide their parent region into two parts, and each leaf represents an undivided region called a *cell*, each corresponds to one single point in that region. Figures 1 portrays the partitioning of $\mathcal{P}_2$.

Each vertex x is assigned a *label* to specify the region space of x. We define $x_{label} = ((p_1, d_1), (p_2, d_2), \cdots, (p_{r(x)}, d_{r(x)}))$ where:

$r(x)$**:** The distance of x from the root of the tree.

p_i**:** The plane equation that partitions the current region into two parts.

d_i**:** Determines one side of the plane p_i.

$parent(x)_{label} = ((p_1, d_1), \cdots, (p_{r(x)-1}, d_{r(x)-1}))$

$sibling(x)_{label} = ((p_1, d_1), \cdots, (p_{r(x)}, \bar{d}_{r(x)}))$, where $\bar{d}_i$ is the opposite side of d_i.

$root_{label} = \lambda$, i.e. the empty string.

We treat the labels as strings. The expression $l_1 \sqsubset l_2$ means that l_1 is a prefix of l_2, $|l|$ represents the size of l (i.e. the number of (p_i, d_i) pairs) and $+$ is the concatenation operator. Also, $[l] \stackrel{\text{def}}{=} \{x \in V | l \sqsubset x_{label}\}$ where l is a label and V is the vertex set of the partition tree. Obviously, for a vertex x, $|x_{label}| = r(x)$.

4 Design Principles of RAQ

RAQ is a structure on the nodes of a network. Each node maps to one point in the d dimensional search space. A partition tree $\mathcal{P}tree$ is constructed over the points and thus each node corresponds to a single cell. We say that a node owns its cell and is responsible for providing data to the queries targeting any point in that cell. Since there is a one-to-one map between nodes and the leaf points of the partition tree, we use them interchangeably. So, for example, we assume having labels for each node.

Moreover, each node has several *links* to other nodes of the network. Each link is basically the addressing information of the target node which can be its IP address. The links are established based on the partition tree information and the following rule.

Connection Rule: Consider node x and its label $x_{label} = ((p_1, d_1), (p_2, d_2), \cdots, (p_k, d_k))$. The connection rule of node x implies that x is connected to exactly one node in each of these $|x_{label}|$ sets:

$$[((p_1, \bar{d}_1))] \ , \ [((p_1, d_1), (p_2, \bar{d}_2))] \ , \ \cdots \ , \ [((p_1, d_1), (p_2, d_2), \cdots, (p_k, \bar{d}_k))]$$

For example, in figure 1, node 1 is connected to one node in each of these sets: $\{2\}, \{4\}, \{6, 7, 8\}$. We will show that the *join and leave* mechanisms guarantees the maintenance of connection rule over the network.

Lemma 1 *An arbitrary chosen vertex has link to $O(\log n)$ other nodes in the network.*

It is important to note that the partition tree is not directly maintained by the elements of RAQ; given the coordinates and the labels of the leaves, all information of the partition tree can be uniquely obtained, and these are the only data which are maintained by the nodes of the network. In fact, the partition tree is the *virtual data structure* of RAQ.

It is obvious that $\mathcal{P}tree$ is a balanced tree with the height of $O(\log n)$ when it is first constructed. We will argue that this property holds even in the dynamic environment where the nodes join and leaves the network.

5 Exact Match Query

In RAQ, exact-match queries are of the form $Exact - Query(target, metadata)$. The value of *target* is the coordinate of the point that is searched for and *meta-data* contains the data to be used after the query reaches the target. Note that the queries aim to reach the target and the responses vary in the different cases. As mentioned, the target of a query is a node whose region contains the query target point.

We say, a point p matches a label l at level k, if k is the greatest value of i such that the subspace induced by a vertex x with $x_{label} = ((p_1, d_1), (p_2, d_2), \cdots, (p_i, d_i))$ contains p and $x_{label} \sqsubset l$. In other words, say l represents a leaf y in $\mathcal{P}tree$. Then, x is the lowest common ancestor of y and the node containing p.

Lemma 2 *Suppose node x receives a query targeting point p and p matches x_{label} at level k. If $k = |x_{label}|$ then the cell of x contains p. Otherwise, x has a link to a node y so that y_{label} matches p at a level greater than k.*

Proof: Let $x_{label} = ((p_1, d_1), \cdots, (p_{r(x)}, d_{r(x)}))$. If $k = |x_{label}|$, then, obviously, x contains p. If not, from the connection rule, we know that x is linked to a node y with $((p_1, d_1), \cdots, (p_{k+1}, \bar{d}_{k+1})) \sqsubset y_{label}$. Therefore, according to the definition, p matches y_{label} at a level not less than $k + 1$. □

Now, the algorithm for exact match routing becomes clear. Once the query Q is received by a node x, if x contains the target point, then we have done. Otherwise, x sends the query via a link to a node y with a label that matches the target point at a higher level. This will continue until the query reaches the target.

Theorem 1 *The exact match query resolves via $O(\log n)$ message passing.*

Proof: Suppose the target of query Q is the node x. From lemma 2, Q will reach to x in at most $|x_{label}|$ steps and $|x_{label}| = O(\log n)$. So, the routing operation is performed by $O(\log n)$ message passing. □

6 Range Query

We assume that a range query Q is of the form *Range-Query(label, pivot, func, d_1, d_2, metadata)* where *label* implies that Q must be sent only to the nodes x so that $label \sqsubset x_{label}$, we denote the label of Q by Q_{label}. The initial value of Q_{label} is set to empty string. The value of *pivot* is the coordinate of the point that the distances are measured from, and *func* is the distance function.

The above range query means that Q should be sent to every node in the network with the distance of $d_1 \leq d \leq d_2$ from *pivot*. *func* can be any distance function $\mathcal{F}$ with the following characteristic: Given a point p, a hyper-cubic subspace S and a distance d, let $A = \{x | x \in S \text{ and } \mathcal{F}(x, p) < d\}$. The problem of whether or not A is empty must be computable. For example, $\mathcal{F}$ can be L_p-*norm* function, in which case the answer space of Q is $\{x | d_1 \leq (\sum (p_i - x_i)^p)^{1/p} \leq d_2\}$.

To handle the range queries, we use *split the space, duplicate the query* method, or split-duplicate for short. Suppose that node x receives a range query Q and $x_{label} = ((p_1, d_1), \cdots, (p_{r(x)}, d_{r(x)}))$. Obviously, $Q_{label} \sqsubset x_{label}$. If $Q_{label} = x_{label}$, then x itself will give the appropriate response. Otherwise, we iterate the following sequence of operations:

1. Duplicate Q and name the results as Q_1 and Q_2. Set $Q_{1_{label}} = Q_{label} + (p_{|l|+1}, d_{|l|+1})$ and $Q_{2_{label}} = Q_{label} + (p_{|l|+1}, \bar{d}_{|l|+1})$.

2. If the answer space of Q has intersection with the subspace induced by $Q_{2_{label}}$, then send Q_2 via the link to node y where $Q_{2_{label}} \sqsubset y_{label}$. Note that by the connection rule, y exits.

3. Iterate split-duplicate operation subsequently on Q_1, while the split subspace has intersection with the answer space.

Lemma 3 *If node x receives range query Q, then Q will be routed to all nodes y where $Q_{label} \sqsubset y_{label}$, and the intersection of the cell of y and answer space of Q is not empty.*

Proof: We prove this by backward induction on $|Q_{label}|$. If $|Q_{label}| > |x_{label}|$, then obviously $Q_{label} \not\sqsubset x_{label}$ thus the induction basis holds.

Suppose that x receives the query and the induction hypothesis holds for $k > |x_{label}|$. If $Q_{label} = x_{label}$ then x is the only target of the query and we are done. Otherwise, two new queries are generated by the algorithm, while the second query Q_2 is sent to its adjacent node via an appropriate link, if the subspace induced by $Q_{2_{label}}$ has nonempty intersection with the answer space. The size of the labels of these new queries is increased by one. Thus, by the induction hypothesis, Q will be routed to all nodes z with $Q_{label} \sqsubset z_{label}$ where the cell of z has a nonempty intersection with the answer space. The union of the induced spaces of these labels covers the whole space of Q_{label}. The claim is therefore correct. □

Theorem 2 *RAQ resolves range queries in at most $O(\log n)$ communication steps. In other words, a query will be received by a target node by crossing $O(\log n)$ intermediate hops.*

Proof: As we mentioned, the basic-queries enters the system by initializing its label to an empty string. By lemma 3, the range-query will be received by all nodes whose cells have nonempty intersections with the query answer space. In each communication step, the size of the query label is increased at least by 1. Thus, when a node receives a query, the distance to the source must be $O(\log n)$, or equivalently the size of the label. □

7 Joining and Leaving

In this section, we describe the joining and leaving mechanism and demonstrate the validity of our claim in section 4 that the partition tree remains balanced all the time.

7.1 Joining Mechanism

Suppose that node x wants to join to the network. x chooses a fairly random point p, in the space and finds y one of the active nodes in the network. Several mechanism can be adopted for the arriving node to find an active node; we assume that RAQ uses the same mechanism as in CAN [11].

Sending an exact match query by y to find the node z whose cell contains p. z divides its cell into two parts, with one containing the corresponding point of z and the other includes p. We assume that x possess the cell containing p. This is just a simple insertion into the partition tree. This is done by updating the labels of x and z. Since we are not directly maintaining this tree, this update is sufficient.

The connections are now updated to follow the given connection rule: x chooses one random point in each of the subspaces induced by the labels specified by the connection rule. For each of these points, say r, z routes an exact match query to find the node that owns r. Consequently, x establishes a connection link to this node.

Theorem 3 *Join operation is done via $O(\log^2 n)$ message passing.*

Proof: The arriving node finds its region by an exact match query. By lemma 1, the arriving node has to create $O(\log n)$ connections. Establishing each connection is done by a exact-match lookup, via $O(\log n)$ message passing. Therefore, the whole operation is completed by $O(\log^2 n)$ message passing. □

7.2 Leaving Mechanism

Let x be the node that wants to leave the network. After x leaves the network, its sibling in $\mathcal{P}tree$ will maintain the region once belonged to x. From $\mathcal{P}tree$ viewpoint, leaving is just a simple deletion of a leaf in a binary tree, so z_{label} and thus $\mathcal{P}tree$ will be updated easily. The difficult part is updating the connection links of the nodes that have links to x. To handle this issue *Departure links* or for short *dlinks*, are defined below.

In RAQ, node b maintains *addressing information* of a, or a *dlink* to a, when node a establishes a connection link to node b. When b decides to leave the network, it sends a message to each of the nodes referred to by its dlinks. In the following, we denote *d-degree* of b as the number its dlinks.

Theorem 4 *The expected value of d-degree is $O(\log n)$*

Proof: Here we argue the validity of our claim. From the mechanism described above to establish a connection link, and from the dynamic structure of the network where the nodes frequently join to and leave the network, we can fairly conclude that the probability that a node v has a link to a node u is equal to the probability that u has a link to v. We avoid discussing the uncomplicated details of this claim, due to the lack of space. Accordingly, $E[v_{degree}] = E[v_{d-degree}]$ for an arbitrary node v in the network. □

Corollary 1 *By lemma 1 and theorem 4, each nodes of the network maintains the addressing information of $O(\log n)$ nodes of the network.*

Consider the time when x is to leave the network. x sends a departure message to all of its nodes on its dlink. As mentioned, every connection in RAQ is a link to a subspace, each of the nodes that receives this departure message, chooses a new random point, say p, in the corresponding subspace and sends an exact match query via x to establish a new connection link to the node that possesses p. After these operations, x will peacefully leaves the network and the connection rule of RAQ is maintained.

Theorem 5 *The leave operation is done via $O(\log^2 n)$ message passing.*

Proof: According to theorem 4, $O(\log n)$ links must be updated. Each update is performed by $O(\log n)$ message passing, thus the total number of message passing is $O(\log^2 n)$. □

According to the discussion, arriving nodes are distributed all over the space. Thus, the partition grows uniformly and remains balanced. Uniform distribution of the nodes also implies that the nodes leave the network randomly in the entire space. We can thus conclude that the claimed proposition about the balancing of the partition tree is valid.

8 Conclusion

In this paper we presented RAQ, a range-queriable distributed data structure for peer-to-peer networks to organize the multidimensional data it holds, and to efficiently support exact and range queries on the data. Our structure is easy to implement and use $O(\log n)$ memory space for each of its n nodes. The exact match query can be performed, as in other works, by $O(\log n)$ message passings. The main contribution of this paper is that the structures broadcast the range query to the target nodes within at most $O(\log n)$ link traversing steps. We showed that all properties of RAQ can be maintained when nodes join the network or leave it.

We are currently working on other extensions of the RAQ model, including its probabilistic model to reduce the complexity of the degree of the nodes in the network. We also intend to validate our results through experimental evaluation with real data. Other ideas can be to design a fault tolerant model to handle different faults such as the situation the nodes abruptly leave the network or abstain to handle the queries temporarily. Load balancing is another important property of the RAQ to look at. In this case, we are going to study the situations that the data points are not uniformly distributed in the search space; also, the computing power of the nodes of network are different. Further works on these topics are underway.

References

[1] J. Aspnes, y. Kirschz, A. Krishnamurthy. Load balancing and locality in range-queriable data structures. in Prod, PODC the twenty-third annual ACM symposium on Principles of distributed computing

[2] J. Aspnes and G. Shah. Skip Graphs. In Proceedings of Symposium on Discrete Algorithms, 2003.

[3] F. Banaei-Kashani and C. Shahabi, SWAM: Small-World Access Model. In Proc. CIKM 2004, Thirteenth Conference on Information and Knowledge Management CIKM 2004, Nov 2004.

[4] Gnutella, http://gnutella.wego.com

[5] J. Gao and P. Steenkiste, An Adaptive Protocol for Efficient Support of Range Queries in DHT-based Systems. Proc. 12th IEEE International Conference on Network Protocols (ICNP'04), Berlin, Germany, Oct. 2004.

[6] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In Proc. of Fourth USENIX Symposium on Internet Technologies and Systems, 2003.

[7] D. R. Karger and M. Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In ACM Symposium on Parallelism in Algorithms and Architectures, June 2004.

[8] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In Proc. 32nd ACM Symposium on Theory of Computing (STOC 2000), pages 163-170, 2000.

[9] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butter y. In Proc 21st ACM Symposium on Principles of Distributed Computing (PODC 2002), pages 183-192, 2002.

[10] H. Nazerzade. Making Querical Data Networks Navigable. In Proc. ICI 2004 International Conference on Informatic, Sep 2004.

[11] S. Ratnasamy, P. Francis, M. Handley, and R. M. Karp. A scalable content-addressable network. In Proc. ACM SIGCOMM 2001, pages 161-172, 2001.

[12] S. Ratnasamy, J. Hellerstein, and S. Shenker. Range Queries over DHTs. Technical Report IRB-TR-03-009, Intel Research, 2003.

[13] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), pages 329-350, 2001.

[14] R. Seidel. Exact upper bounds for the number of faces in d-dimensional Voronoi diagrams, DIMACS Series, volume 4. American Mathematical Society, 1991.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM 2001, pages 149-160, 2001.