A Fault Tolerant Data Structure for Peer-to-Peer Range Query Processing

Zahra Mirikharaji* Bradford G. Nickerson†

Abstract

We present a fault tolerant dynamic data structure based on a constant-degree Distributed Hash Table (DHT) called FissionE that supports orthogonal range search in d-dimensional space. A publication algorithm, which distributes data objects among all nodes in the network is described, along with a search algorithm that processes range queries and reports all objects in range to the query issuer. Routing messages between two nodes is performed by the FissionE routing algorithm. The worst case orthogonal range search cost in our data structure with n nodes is $O(\log n+m)$ messages plus reporting cost, where m is the minimum number of nodes intersecting the query. Storing d complete copies of each data object on d different nodes provides redundancy for our scheme. This redundancy permits completely answering a query in the case of simultaneous failure of $d-1$ nodes.

1 Introduction

In structured peer-to-peer networks, the P2P overlay network topology is tightly controlled to place contents at specified locations that will make data discovery more efficient. Many structured P2P systems like Chord [15], Tapestry [19], Pastry [11], CAN [9] and FissionE [8] use a DHT [10] to distribute data objects deterministically among the peers and retrieve them with the data object's unique key. DHT-based systems employ hashing to assign IDs to the peers; each peer is responsible for a small specific subset of the data. The number of required messages exchanged between nodes to answer a query defines search cost in these networks.

DHT schemes are normally capable of processing exact match searches, but not more complex searches such as range search . A number of recent papers have investigated DHTs to process range queries. DHT-based techniques for range query answering are classified into two groups [18]; layered indexing and customized indexing. In layered indexing techniques, the underlying topology and routing algorithm of DHTs are used to answer range queries. Our work is in the layered category. Customized indexing uses a custom-designed P2P overlay or modifies an existing P2P overlay network to support range search.

In layered indexing, Gupta et al. [5] use a probabilistic scheme that relies on locality sensitive hashing. However, this method can only report approximate answers for one dimensional range queries. Squid [12] and DCF-CAN [1] use space-filling curves (SFC) to map multi-dimensional keys to the peers. Space-filling curves are locality preserving, but they lead to a less efficient search cost, because a single range query may cover several parts of the curve, each of which requires a separate query. In customized indexing, the skip graph [2] and SkipNet [6] are P2P networks having $O(\log n)$ degree that support one dimensional orthogonal indexing. Family trees [17] and the rainbow skip graph [4] are both constant-degree and support one dimensional range queries. Mercury [3], Znet [13] and MIDAS [16] provide indexing schemes for multi-dimensional space. Mercury [3] provides multiple attribute range queries by indexing the data set along each attribute. The latency of the message routing algorithm in Mercury [3] is $\log^2 \frac{n}{k}$ when each node maintains k links to other nodes. MIDAS [16] resolves the request in $O(\log n)$ hops when each peer's degree is $O(\log n)$. In Znet [13], SFCs (Space Filling Curves) are used and skip graphs [2] are extended for query routing, with each node maintaining $O(\log n)$ states.

Most distributed indexing structures supporting range search don't work on a constant-degree graph. Among the existing constant-degree schemes supporting range search, Armada [7] provides a higher efficiency in terms of query delay and number of required messages. It has been proven in [7] that the lower bound on the message cost of general range query schemes on constant-degree distributed hash tables (DHTs) is $\Omega(\log n)+m-1$, where m is the number of nodes intersecting queries. Armada uses the FissionE P2P network topology DHT scheme based on Kautz graphs [8]. Li et al. [7] have proven that the average message cost of one dimensional queries on uniformly distributed data in PIRA (PrunIng Routing Algorithm) is close to the lower bound on message cost of range queries on constant-degree DHTs. For multi-dimensional indexing, Li et al. [7] have not presented any guarantee on the number of messages required to answer a d-dimensional orthogonal range query. They used simulation to show that the average message cost of MIRA (Multiple attribute prunIng Routing Algorithm) is about $\log n + 4m - 1$ messages.

*Faculty of Computer Science, University of New Brunswick, zahra.miri@unb.ca

†Faculty of Computer Science, University of New Brunswick, bgn@unb.ca

Armada uses the failure recovery mechanisms of the underlying DHT structure of FissionE [8] to accommodate routing recovery, but they don't provide data recovery. Our work improves on Armada's MIRA algorithm by efficiently providing support for orthogonal range search even with simultaneous failure of up to $d-1$ nodes in a network of n nodes.

2 Results

Our paper presents a peer-to-peer distributed dynamic data structure employing FissionE [8] as a constant-degree DHT to route the messages. We give a data publication algorithm to assign d copies of each object to d different nodes. An orthogonal rage search algorithm for each node in an n-node peer-to-peer network is given that can answer d-dimensional range queries Q issued from any network node. The worst case cost for a d-dimensional range search on our data structure with n nodes is $O(\log n + m)$ messages, for m the minimum number of nodes intersecting the query. To support dynamic joining and departure of nodes and failure recovery, we use split large and merge small policies [8]. To the best of our knowledge, our data structure is the first distributed dynamic spatial data structure to fully support orthogonal range search with simultaneous failure of $d-1$ nodes.

3 Data Structure

3.1 Introduction to FissionE

FissionE [8] is a constant-degree distributed hash table based on the Kautz graph. A Kautz graph is a directed graph with static topology that uses Kautz strings as node identifiers. In the following, we present related definitions explaining Kautz graph topology on which the FissionE DHT is built.

The string $u_1u_2...u_k$ of length k and base d is a Kautz string where $u_i \in \{0,1,2,...,d\}$ and $u_i \neq u_{i+1}$ $(1 \leq i \leq k-1)$. All Kautz strings of length k and base d create the $KautzSpace(d,k)$ of size d^k+d^{k-1}. To show the size of $KautzSpace(d,k)$, we know that the first symbol in a Kautz string has $d+1$ possibilities. Two consecutive symbols in a Kautz string must be different, so all other symbols have d possibilities.

The Kautz graph $K(d,k)$ is a directed graph of degree d with $d^k + d^{k-1}$ nodes labelled by strings in $KautzSpace(d,k)$. Each node $U = u_1u_2...u_k$ of a Kautz graph has the same out-degree and in-degree d. There is an outgoing edge from U to V if and only if $V = u_2u_3...u_k\alpha$ where $\alpha \in \{0,1,...,d\}$ and $\alpha \neq u_k$. Figure 1 shows Kautz graph $K(2,3)$ with out-degree 2 and 12 nodes.

The Kautz graph has desirable properties like optimal diameter that are important in peer-to-peer networks.

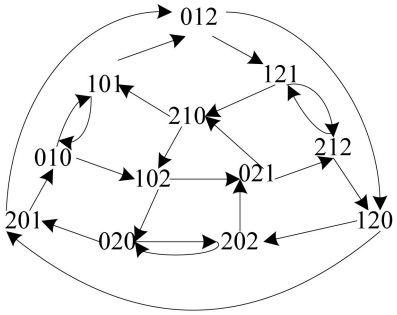

Figure 1: Kautz graph $K(2,3)$ (from [8]).

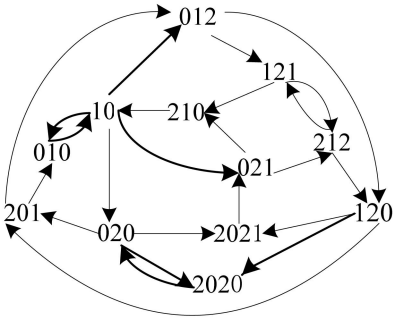

Figure 2: An example of FissionE topology (from [8]).

Diameter is the longest shortest path between any two vertices of a graph and is always in trade-off with the degree of a graph. For a graph with $n = d^k + d^{k-1}$ nodes and degree d, the Kautz graph has the smallest diameter of any possible directed graph. In addition, in Kautz graph $K(d,k)$, there are d disjoint paths between any two nodes and failure of $d-1$ nodes is tolerable.

FissionE uses a $K(2,k)$ Kautz graph. A Kautz graph is a static topology, so it needs some adjustment to be used for dynamic peer-to-peer networks. Li et al. in [8] propose a new topology called approximate Kautz graph. To achieve an approximate Kautz graph, the network topology is first initiated with a Kautz graph, and then in dynamic operations (addition and deletion of nodes) a topological rule called the *neighbourhood invariant* rule is adopted. Based on this rule, the length of identifiers may be different for different peers and the difference in length of node identifiers of any two neighbours must be one or zero. Figure 2 shows an example of neighbourhoods in FissionE topology. This topology is first initiated with Kautz graph $K(2,3)$. Node 202 is split to permit node 2021 to join the network with existing node 202 becoming node 2020. Data in nodes 101 and 102 are merged to provide one less node which results in node 101 being relabelled to node 10, and node 102 departing from the network.

To distribute objects among nodes in the FissionE scheme, the *Kautz_hash* algorithm is proposed in [8]. The *Kautz_hash* algorithm maps an object's unique key

(of any length) to the destination Kautz string of length m consisting of digits $v_i \in 0, 1, 2$, where consecutive digits must be different. Li et al. [8] show that when $m = 100$, the *Kautz_hash* algorithm uniformly distributes the Kautz strings it generates in Kautz namespace $KautzSpace(2, 100)$. As mentioned in Algorithm 1.2.2 of [8], this namespace has size $2^{100}+2^{99} \simeq 1.9\times10^{30}$. To publish an object O in a FissionE topology from node p, the Kautz string V of the object is first computed. Next, node p (the original node) routes the generated Kautz string V to place O in the node whose identifier is a prefix of V. To locate an object in the network, the same process is performed, with node p being the query issuer.

The long path routing algorithm in a Kautz graph is chosen as the Routing algorithm in FissionE. In this algorithm, routing from node U to the node where destination Kautz string V resides is performed by left shifting the symbols of U and adding the symbols of V from left to right at the end of U. For example, if $U = 021$ and $V = 12010$, the longest common prefix of V and suffix of U is equal to 1. So the length of path from node U to the node whose identifier is a prefix of V is 2, and the routing path is $021 \rightarrow 212 \rightarrow 120$.

It is proven in [8] that the average degree of vertices in a FissionE network is 4 and its Kautz graph diameter is less than $2 \log n$. These desirable characteristics motivate us to use FissionE as our overlay network to route messages between nodes and provide dynamic operations of node arrival and departure.

3.2 Data Distribution

DHT-based peer-to-peer networks usually use consistent hashing functions to map data objects and peers to a namespace. In the namespace, each node takes the responsibility of storing values with IDs close to its own ID. In the case of range queries, a peer-to-peer network requires data ordering, so the hash function used to map values into the namespace is replaced by a locality preserving mapping function. Although the FissionE scheme is a high performance distributed peer-to-peer network and achieves optimal diameter on a constant degree graph, it supports only processing of exact match queries (point queries). In this work, we present a general range query scheme that uses FissionE for routing messages. Two main components of our work are the data distribution and the range search algorithms. Our data distribution algorithm publishes d copies of each object on d different nodes in a way that preserves data locality. Our range search algorithm efficiently forwards queries to the appropriate nodes in range. We first give a formal definition of a total order relation, and then explain our data distribution algorithm. To efficiently answer a range query over a peer-to-peer network, it is required to define a total order relation on the dataset to keep the order of data in each dimension. A total order relation is a binary relation on set X denoted by $\leq$ which has the following properties for all a, b and $c \in X$:

1. Antisymmetry: If $a \leq b$ and $b \leq a$ then $a = b$.
2. Transitivity: if $a \leq b$ and $b \leq c$ then $a \leq c$.
3. Totality: $a \leq b$ or $b \leq a$.

A total order relation on data provides propagation of objects on FissionE nodes in such a way that objects with close values are placed on the same or neighbouring nodes. In our data structure, we define a total order relation for each dimension i as follows:
For two points $P(p_1, p_2, ..., p_d)$ and $Q(q_1, q_2, ..., q_d)$, $P \leq Q$ in dimension i if $p_i \leq q_i$.

As explained in section 3.1, in FissionE the identifier of nodes are Kautz strings and network nodes are initiated to a Kautz graph. All Kautz graphs have a Hamiltonian path. A Hamiltonian path in a graph is a path that visits each node of a graph exactly one time. In our work, we use the Hamiltonian path in the underlying Kautz graph of FissionE, and assign the index of each node in the Hamiltonian path as the key to each node.

To preserve data locality along all dimensions, we distribute data objects among nodes by partitioning the space based on point coordinates. In d-dimensional space, we assign d sets of points to each node i on the network, each set corresponding to one dimension. Set S_{ji} is the data stored on node i based on the total order relation in dimension j. For example, in 2-dimensional space, if we denote dimension 0 with x coordinates, and dimension 1 with y coordinates, we assign two sets of points S_{xi} and S_{yi} to every FissionE node i.

The distribution of points based on each dimension over n nodes is a noncrossing partition $NC(\mathcal{S}) = \{S_{j1}, S_{j2}, ..., S_{jn}\}$ [14]. A partition over set $\mathcal{S}$ on dimension j has the following properties:

- The union of the sets of $NC(\mathcal{S}) = \{S_{j1}, S_{j2}, ..., S_{jn}\}$ is equal to $\mathcal{S}$. The elements of $NC(\mathcal{S})$ are said to cover $\mathcal{S}$; i.e. for any j, $0 \leq j \leq d-1$, $\cup_{i=1}^{n} S_{ji} = \mathcal{S}$ where d is the number of dimensions in the data structure.
- The intersection of any two distinct sets of $NC(\mathcal{S})$ is empty; i.e. the elements of $NC(\mathcal{S})$ are pairwise disjoint. Thus $S_{ji} \cap S_{jk} = \emptyset$ if $S_{ji} \in NC(\mathcal{S})$, $S_{jk} \in NC(\mathcal{S})$, $i \neq k$.

In 2-dimensional space, our data structure provides one backup copy of data published on all nodes to achieve search cost near the lower bound, in addition to providing data recovery. A copy of data stored in

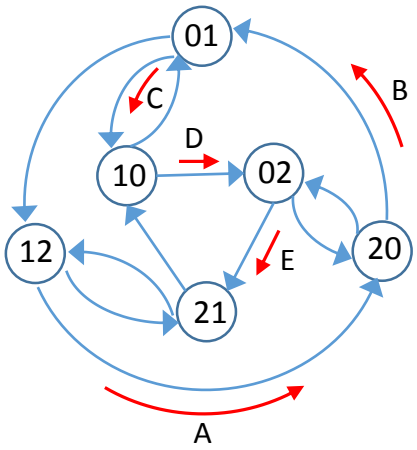

Figure 3: A Hamiltonian path on Kautz graph $K(2,2)$.

node i is stored in all $n-1$ other nodes except node i. Figure 4 shows an overview of the data distribution in a 2-dimensional space over the Kautz graph $K(2,2)$ shown in Figure 3. If the data is distributed in a uniform random fashion in space, a balanced load for each node results. The horizontal colour bar in each cell indicates the place of the first copy of data in that cell based on dimension 0 (X), and the vertical colour bar indicates the place of the second copy of data in that cell based on dimension 1 (Y). For example, assume $L = [0,0]$ and $U = [12,12]$ are the lower and upper bound of the entire 2-dimensional space, and $P = [0.8, 1.2]$ is a point. By uniformly partitioning the space among the six nodes in Figure 3, point P is placed in the lower left cell with red and orange bars. The red and orange bars show that the first and second copies of point P are stored on nodes 12 and 20, respectively.

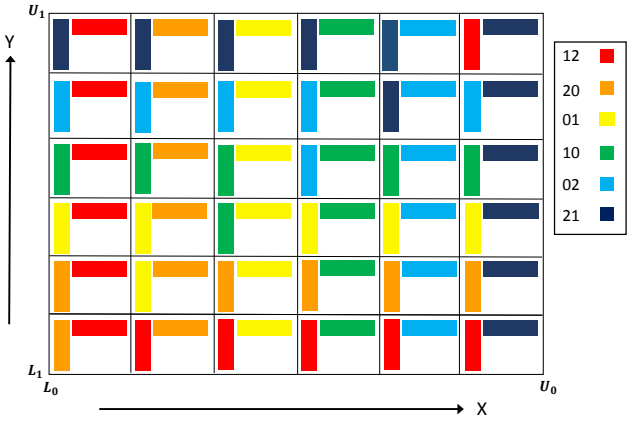

Figure 4: An overview of a 2-dimensional data distribution in our data structure.

Algorithm 1 shows how d-dimensional data objects are distributed on a network of n nodes. This algorithm publishes d copies of object O on d different nodes. The place of the ith copy of object O depends on O_i and the place of the other $i-1$ copies of O that are already specified. $[L_i, U_i]$ in this algorithm is the entire interval of object values in dimension i. When a network is initiated with a Kautz graph, a Hamiltonian path of the graph is found and stored on all nodes. In the case of join and departure of a node, this Hamiltonian path P is updated on all nodes.

Algorithm 1 Publish data object O on a network with n nodes.

1: **procedure** DATADISTRIBUTION(ObjectVal O, NumofNodes n, NumofDims d, LowerBound L, UpperBound U, HamiltonianPath P)
// $O = [O_0, O_1, ..., O_{d-1}]$ is the coordinate of a point that should be published on d nodes.
// $L = [L_0, L_1, ..., L_{d-1}]$ and $U = [U_0, U_1, ..., U_{d-1}]$ are the lower and upper bounds, respectively, of the entire space.
2: **if** $(O < L \| O > H)$ **then**
3: **return** (O is not in range.)
4: **end if**
5: $NodeIndex \leftarrow null$ // $NodeIndex$ is an array of size d showing the indices of nodes that O will be published on.
6: **for** $i \leftarrow 0, d-1$ **do**
7: $node = \lceil \frac{O_i - L_i}{U_i - L_i}(n) \rceil - 1$
8: **for** $j \leftarrow 0, i$ **do**
9: **if** $NodeIndex[j] == node$ **then**
10: $node = (node + 1) \mod n$
11: $j \leftarrow 0$
12: **end if**
13: **end for**
14: $NodeIndex[i] \leftarrow node$
15: **end for**
16: $NodeID \leftarrow null$ // $NodeID$ is an array of size d showing the Kautz string of nodes where O is stored.
17: **for** $i \leftarrow 0, d-1$ **do**
18: $NodeID[i] \leftarrow P[NodeIndex[i]]$
19: **end for**
20: **return**($NodeID$)
21: **end procedure**

3.3 Range Search

We initially assume that the query Q is searching for one specific point $P(x,y)$ in our data structure. The query can be issued at any one of the nodes. Routing starts at the query issuer node. The query issuer uses Algorithm 1 to find the $NodeID$ list corresponding to point P which contains d distinct addresses of P. The query issuer then determines which of the point's addresses to use. To do that, the longest Kautz string S_i which is a suffix of the query issuer ID and a prefix of $NodeID[i]$ is calculated for each i, $0 \leq i \leq d-1$. Then the query issuer uses the FissionE routing algorithm [8] to pass the query to the $NodeID[j]$, where S_j has the

maximum length among all S_is.

A Hamiltonian path A, B, C, D, E for the Kautz graph $K(2,2)$ is shown in Figure 3. Algorithm 2 answers a d-dimensional range query Q in a network of n nodes. Our range search algorithm has two main parts. First, it determines which dimension of Q intersects the minimum number of nodes. Second, the first node i in range is found and the first portion of data in range is reported. Node i checks if the query upper bound is greater than the node i upper bound; if so then the rest of the search result might be in the next node and an updated query is sent to the next node in range. The new query rectangle is the result of subtracting the query range of node i from the old query rectangle. The same process continues until the last node intersecting the query reports the last part of the result to the query issuer node.

For example, assume that j is the dimension intersecting the fewest nodes. Node i is the first node that sends the result back to the query issuer if $ThisNode_{jL} < Q_{jL} < ThisNode_{jU}$ where $ThisNode_{jL}$ and $ThisNode_{jU}$ are node i's lower bound and upper bound respectively, in dimension j and Q_{jL} is the lower bound of the query in dimension j. If the upper bound Q_{jU} of the query in dimension j is greater than $ThisNode_{jU}$, the query is passed to the next node in range by the call at line 17. The next node on the Hamiltonian path now receives the query, which was updated as shown on line 16.

Theorem 1 *The worst case orthogonal range search cost in our fault tolerant data structure for any data distribution in a d-dimensional space with n nodes is* $O(\log n + m)$ *messages plus reporting cost, where m is the minimum number of nodes intersecting the query on d dimensions.*

4 Fault Tolerance

When failure of one node occurs, problems arise due to an outdated routing table and the fact that the data set assigned to the failed node will be unavailable. To enhance fault tolerance, most distributed data indexing schemes use replication based mechanisms. Data redundancy is part of our distributed spatial data structure as explained in section 3.2. In d-dimensional space, our data structure stores d copies of data in such a way that one copy of each object resides on d different nodes. If one network node fails, we use the involuntary departure of nodes methods in FissionE (Figure 7 in [8]). Each node periodically checks whether its neighbours are alive. When the failure of node k is detected by its neighbour, two neighbour nodes y_1 and y_2 in the network are found which have no neighbour node with less data. They are merged into a new node ℓ and the neighbour lists of ℓ and related nodes are updated. We now have one extra node (y_1 or y_2) to get the $NodeID$ of the failed node k and be responsible for its data. If a query requests data from a failed node, all queries can be processed completely. Our data structure can retrieve data of failed nodes whenever failure of one or more (up to $d-1$) nodes occurs. Thus, our data structure can answer orthogonal range search queries after simultaneous failure of $d-1$ nodes.

Theorem 2 *In our fault tolerant data structure with n nodes storing N points, the cost of recovering network topology and data after failure of one node in d-*

Algorithm 2 Report data objects in a range query to its issuer node.

1: **procedure** IssueQuery(rangeQuery Q)
// Find the dimension j with minimum range $Q_{jU} - Q_{jL}$ in query Q
// L_j and U_j are the lower bound and upper bound, respectively, of all possible values for dimension j.
2: $j \leftarrow$ MinRangeDim(Q) // Proper dimension for query processing
3: $dstIndex \leftarrow \lceil \frac{Q_{jL} - L_j}{U_j - L_j}(n) \rceil - 1$
4: $dstID \leftarrow HamiltonianPath[dstIndex]$
5: FissionERouting($thisNode, dstID, Q, j$)
6: **end procedure**
7: **procedure** FissionERouting(srcNode src, dstNode dst, Rangequery Q, properD j)
// Assume that $src = src_1 src_2 ... src_k$ and $dst = dst_1 dst_2 ... dst_m$.
8: $SP \leftarrow$SuffixPrefix(src, dst)
// $SP = SP_1 SP_2 ... SP_t$ the longest Kautz string that is a prefix of dst and a suffix of src.
9: src.Routing($dst, k-t, SP, Q, j$)
10: **end procedure**
11: **procedure** U.Routing(dstNode dst, pathLen L, sufPre SP, rangeQuery Q, properD j)
12: **if** $(L = 0)$ **then**
13: **if** $(Q_{jL} > ThisNode_{jL})$ and $(Q_{jL} < ThisNode_{jU})$ **then**
// Report all objects in range where $O_j < ThisNode_{jU}$.
14: ReportAnswer(LocalSearch(Q), Q.issuer)
// If not the last node in range
15: **if** $(Q_{jU} > ThisNode_{jU})$ **then**
16: $Q_{jL} \leftarrow ThisNode_{jU}$
// Route updated query to the next node in Hamiltonian path
17: FissionERouting($thisNode, thisNode.next, Q, j$)
18: **end if**
19: **end if**
20: **else if** $\exists Q \in Outneighbors(U)$ & $Q = U_2 ... U_k X$ & $IsPrefix(SX, dst)$ **then**
// Routing method has been called i times.
21: $S \leftarrow SX$
22: Q.Routing($dst, L-1, S, Q, j$)
23: **end if**
24: **end procedure**

dimensional space is $O(\frac{dN}{nB}\log n)$ *messages, where* B *is the number of points that fit in one message.*

5 Conclusion

We have designed a dynamic peer-to-peer data structure for d-dimensional data that is capable of processing orthogonal range search on a set of N points. The constant degree FissionE topology was used to coordinate message passing among nodes. The worst case range search cost is $O(\log n + m)$ messages plus reporting cost, where n is the number of nodes in the peer-to-peer network, and m is the minimum number of peers intersecting a query. A failure recovery method was introduced that permits our data structure to support d-dimensional orthogonal range search when up to $d-1$ nodes fail simultaneously. It remains an open question how to provide load balancing of nodes in our data structure if the distribution of data is changed due to dynamic updates.

6 Acknowledgements

The authors would like to acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada and the UNB Faculty of Computer Science.

References

[1] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing, 2002.(P2P 2002). Proceedings. Second International Conference on*, pages 33–40. IEEE, 2002.

[2] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4):37, 2007.

[3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[4] M. T. Goodrich, M. J. Nelson, and J. Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393. ACM, 2006.

[5] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *CIDR*, volume 3, pages 141–151, 2003.

[6] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, volume 274. Seattle, WA, USA, 2003.

[7] D. Li, J. Cao, X. Lu, and K. Chan. Efficient range query processing in peer-to-peer systems. *Knowledge and Data Engineering, IEEE Transactions on*, 21(1):78–91, 2009.

[8] D. Li, X. Lu, and J. Wu. Fissione: A scalable constant degree and low congestion dht scheme based on kautz graphs. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1677–1688. IEEE, 2005.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[10] S. Ratnasamy, I. Stoica, and S. Shenker. Routing algorithms for dhts: Some open questions. In *Peer-to-peer systems*, pages 45–52. Springer, 2002.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[12] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[13] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 173–180. IEEE, 2005.

[14] R. Simion. Noncrossing partitions. *Discrete Mathematics*, 217(1):367–409, 2000.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[16] G. Tsatsanifos, D. Sacharidis, and T. Sellis. Midas: multi-attribute indexing for distributed architecture systems. In *Advances in Spatial and Temporal Databases*, pages 168–185. Springer, 2011.

[17] K. C. Zatloukal and N. J. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 308–317. Society for Industrial and Applied Mathematics, 2004.

[18] Y. Zhang, X. Lu, and D. Li. Survey of dht topology construction techniques in virtual computing environments. *Science China Information sciences*, 54(11):2221–2235, 2011.

[19] 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. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

Appendix

Proof of Theorem 1

Proof. Since the most efficient dimension j for the issued query is selected at the beginning of the range search algorithm, the worst case search cost occurs when the query is an equal-sided box. Li et al. have proven in [8] that the diameter of FissionE is $O(\log n)$. So, in the worst case, the cost of finding the node containing the lower bound of the orthogonal range query Q_{jL} is $O(\log n)$. After that we need $O(m)$ messages to pass the updated query to the following nodes in range using the current Hamiltonian path to find data objects intersecting the query. Adding the two costs gives the worst case orthogonal range search cost for d-dimensional points distributed on a peer-to-peer network of n nodes as $O(\log n + m)$ messages plus reporting cost. □

Corollary 3 *Assuming B points fit in one message, the cost of reporting K points found in range back to the query issuer node is $O((\frac{K}{B} + m)\log n)$ messages.*

Proof. The cost to report the points in range is $\sum_{i=1}^{m} \lceil \frac{K_i}{B} \rceil O(\log n)$ messages, where K_i is the number of points in range on node i and $\sum_{i=1}^{m} K_i = K$. As $\sum_{i=1}^{m} \lceil \frac{K_i}{B} \rceil O(\log n) \in O((\frac{K}{B} + m)\log n)$, we have the claimed reporting cost. □

Proof of Theorem 2

Proof. It has been proven in [8] that when one node fails, depart messages are propagated less than $\log n$ hops. So, the cost of merging two nodes and maintenance of the overlay network is $O(\log n)$. After that, each node finds which parts of its own data were stored on the failed node, and sends this data to the replacement node. If we assume that B points can fit in one message, the data recovery process requires $O(\frac{dN}{nB}\log n)$ messages since when one node fails, $\frac{dN}{n}$ points residing on the failed node are lost. So, $O(\frac{dN}{nB})$ messages are forwarded at most $O(\log n)$ hops to send back the lost data to the replacement node. The overall cost is thus $O(\log n + \frac{dN}{nB}\log n) = O(\frac{dN}{nB}\log n)$ messages. □