Range Queries over DHTs

Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker

IRB-TR-03-009

June, 2003

Intel Research Berkeley

Range Queries over DHTs

Sylvia Ratnasamy, Joseph M. Hellerstein, Scott Shenker

Abstract

Distributed Hash Tables (DHTs) are scalable peer-to-peer systems that support exact match lookups. This paper describes the construction and use of a *Prefix Hash Tree* (PHT) – a distributed data structure that supports range queries over DHTs. PHTs use the hash-table interface of DHTs to construct a search tree that is efficient (insertions/lookups take $O(\log \log |D|)$ DHT lookups, where D is the data domain being indexed) and robust (the failure of any given node in the search tree does not affect the availability of data stored at other nodes in the PHT).

1 Introduction

Distributed Hash Tables [7, 2, 6, 5] are scalable P2P systems that support functionality similar to that of a hash table. There is one basic operation in DHT systems, `lookup(key)`, which returns the identity (*e.g.*, the IP address) of the node storing the object with that key thus enabling nodes to insert and retrieve data items based on their key. DHTs are thus analogous to hash indexes, providing only exact-match lookups. An open question has been whether range queries can be supported in as graceful and elegant a distributed fashion as DHTs support exact match queries. In this paper, we propose implementing range queries *over* a DHT via a trie-based scheme, with the added twist that a bucket in the trie is stored at the DHT node obtained by hashing its corresponding prefix. This hash-based assignment of trie buckets to DHT nodes lends the PHT a number of performance advantages relative to a more straightforward implementation of a distributed trie in which each node in the trie holds an explicit pointer to its children. Specifically, (1) accessing a PHT vertex take O($\log \log |D|$) DHT lookups where D is the data domain being indexed, (2) a PHT is robust in that the failure of a given node does not affect the availability of any other nodes in the trie and finally, (3) a PHT avoids overloading the nodes at the higher levels of the trie by not requiring that all trie traversals start at the root and recurse down. Finally, PHTs build entirely on top of the DHT *lookup()* interface and are hence agnostic to the particular choice of underlying DHT routing algorithm.

2 Prefix Hash Trees

At the abstract level, a PHT is a trie[1] in which every vertex corresponds to a distinct prefix of the data domain being indexed.[2] More specifically, every vertex in the tree has an associated prefix label that is determined as follows: given a vertex with label *l*, its left and right child vertices are labeled *l0* and *l1* respectively. The root of the tree is labeled with the attribute name and all downstream vertices are labeled recursively as above.

A data item is stored at the PHT node with the longest prefix match between the node label and the item to be stored. A node stores upto B data items; when this threshold is exceeded, the node "splits" into its two child vertices and the data items it stores are appropriately partitioned (based on their prefixes) between its children. In other words, the prefixes of the leaf nodes in the PHT form a *universal prefix set*[3] and a data item is inserted at the PHT leaf node whose label is a prefix of the item. Note thus that data items are stored only at the leaf nodes in the PHT and the PHT itself grows dynamically with data insertions. Hence while the worst case depth of the PHT is $\log |D|$ (where D is the data domain being indexed), in practice, the depth and width of the PHT will grow dynamically based on the distribution of inserted values.

For scalability, this logical PHT structure is distributed across the nodes in a DHT. The key idea behind PHTs is to achieve this distribution by *hashing* the prefix labels of PHT vertices over the DHT node identifier space; using the DHT lookup operation, a PHT vertex with label *l* is thus assigned to the node with identifier closest to HASH(*l*). In short, PHT vertices are assigned to DHT nodes by consistent hashing of PHT labels over the node identifier space.

This hash-based assignment of the trie vertices to DHT nodes means that one can directly "jump" to *any* vertex in the trie via a single DHT lookup which offers three key advantages relative to traditional trees where accessing a vertex requires tracing the path from the root of the tree recursively down to the desired vertex:

- Given a data item of $\log |D|$ bits, we can locate its as-

[1]A trie is a multi-way retrieval tree used for storing strings over an alphabet in which there is one node for every common prefix and all nodes that share a common prefix hang off the node corresponding to the common prefix.

[2]For simplicity, we describe PHTs as being binary tries however our discussion extends naturally to higher bases.

[3]A set of prefixes is a universal prefix set if and only if for any infinite binary sequence b there is exactly one element in the set which is a prefix of b.

sociated PHT leaf node through a binary search over the $\log|D|$ possible prefix labels corresponding to the item. Thus, a given data item can be inserted into (retrieved from) the PHT through a sequence of $O(\log\log|D|)$ DHT lookups.[4]

- In contrast to typical trees, the failure of a PHT node does not affect the availability of nodes in the subtree rooted at the failed node.
- Because accesses to individual nodes need not traverse through the root, PHTs avoid creating a bottleneck at the root of the PHT.

Range Queries using PHTs: A number of techniques can be used to perform a range query over the items stored in a PHT. Some example techniques include:

- locating the PHT node corresponding to the longest common prefix in the range and then performing a parallel traversal of its subtree to retrieve all the desired items.
- parallelize the query by dividing the range into a number of sub-ranges that can then be retrieved as above.

In addition, a number of simple performance optimization could be used to improve search performance. For example:

- Require every PHT leaf node to hold a pointer to the leaf node to its right (with wrap-around) effectively forming a link list of PHT leaf nodes. Now a range query could proceed by locating the PHT node corresponding to the lowest value in the range and traversing the link list to retrieve the required values.
- Record the "shape" of the entire PHT tree at a well defined DHT location (for example, at the DHT node corresponding to HASH(*attribute-name/shape/*)) by having leaf nodes register at this location. A range query might then proceed by first retrieving the shape of the PHT which would reveal the exact set of PHT leaf nodes to be accessed for the range query. Note, that this performance optimization could also speed up data insertions.
- Have every PHT node propogate hints, such as the depth of its subtree and/or the number of data items stored in its subtree, up to its parent. This allows any internal node in the PHT to immediately direct queries to the appropriate leaf nodes in its subtree.

3 Open Issues and Related Work

This paper offered only a high-level description of the PHT data structure. Much work remains – for example, the benefit and cost of the different range query techniques and performance optimizations should be evaluated and issues with concurrency control should be addressed. An apparent issue with PHTs is that the search complexity is in the size of the domain, not the number of data items being indexed. More work is probably required to refine this technique for skewed datasets over large domains. Finally, while it seems clear that a PHT could quite easily support a number of operations such as sorting, finding the max/min/average over a data set *etc.*, other operations such as multi-dimensional range queries appear more involved. Understanding the nature and scope of operations that one could efficiently support using a PHT is a topic of ongoing research.

Our PHT proposal is reminiscent of Litwin's Trie Hashing [4], but has an added advantage that the "memory addresses" where buckets of the trie are stored are in fact the DHT keys obtained by hashing the corresponding prefixes. Alternative schemes have been proposed as well, including a DHT-based range-caching scheme [3], and a technique specifically for the CAN DHT based on space-filling curves [1].

[4] Note that the search time could be further reduced by parallelizing these lookups in various ways although at the cost of some redundant lookups.

4 Acknowledgments

The authors would like to thank Richard Karp, Sriram Ramabhadran, Mehul Shah and members of the PIER research group for taking the time to discuss the ideas presented here. In particular, Mehul and Sriram proposed some of the performance optimizations we describe.

5 REFERENCES

[1] ANDRZEJAK, A., AND XU, Z. Scalable, efficient range queries for grid information services. In *Proceedings of the second IEEE Internation Conference on Peer-to-peer Computing (P2P2002)* (Sweden, Sept. 2002).

[2] DRUSCHEL, P., AND ROWSTRON, A. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)W* (Nov 2001).

[3] GUPTA, A., AGRAWAL, D., AND ABBADI, A. E. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the first Biennial Conference on Innovative Data Systems Research* (2002).

[4] LITWIN, W. Trie hashing. In *Proceedings of SIGMOD* (May 1981), pp. 19–29.

[5] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proc. ACM SIGCOMM* (San Diego, CA, August 2001), pp. 161–172.

[6] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference* (San Diego, California, August 2001).

[7] ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.