Brief Announcement: Prefix Hash Tree

Sriram Ramabhadran *

Sylvia Ratnasamy

Joseph M. Hellerstein

Scott Shenker

ABSTRACT

This paper describes the Prefix Hash Tree, a distributed data structure that enables range queries over Distributed Hash Tables.

Categories and Subject Descriptors

C.2.4 [**Comp. Communication Networks**]: Distributed Systems — *distributed applications*; E.1 [**Data Structures**]: *distributed data structures*

General Terms

Algorithms, Design, Performance

Keywords

distributed hash tables, data structures, range queries

1. PREFIX HASH TREE

The explosive growth but primitive design of peer-to-peer file-sharing applications such as Gnutella inspired the research community to invent Distributed Hash Tables [1]. While the DHT has enjoyed some success as a building block for Internet-scale applications, it is seriously deficient in one regard — it only directly supports *exact match* queries. However, *range queries*, asking for all objects with values in a certain range, are more difficult to implement over a DHT. This is because most DHT designs use hashing to distribute keys uniformly, and therefore cannot rely on any structural properties of the keyspace, such as an ordering among keys. This paper proposes the Prefix Hash Tree (henceforth abbreviated as PHT), a trie-like data structure that addresses this limitation. A key advantage of the PHT is that unlike some other recent proposals [2] [3], it is particularly suitable for implementation over a DHT, relying only on the canonical `put(key,value)` and `get(key)` hashtable operations.

In essence, the PHT data structure is a binary trie built over the data set being indexed [1]. Each node of the trie is labeled with a prefix that is defined recusively: given a node with label l, its left and right nodes are labeled with $l0$ and $l1$ respectively, with the root labeled with the domain being indexed. Leaf nodes store data; a data item K is stored at the unique leaf node whose label is a prefix of K. A node stores upto B items; when this threshold is exceeded, the node is split into its two children, and its data items are partitioned between its children. Conversely, when the number of data items at two sibling nodes falls below B, they are merged, and their data items are transferred to the parent. Finally, leaf nodes are threaded; each leaf node maintains pointers to the leaf nodes on its left and right.

As described thus far, the PHT is a fairly routine binary trie. The novelty of the PHT lies in how this logical trie is *distributed* among the peers in the network. This is achieved by *hashing* the prefix labels of PHT nodes over the DHT identifier space, i.e., a node with label l is assigned to the peer responsible for `HASH`(l) in the DHT. This hash-based assignment implies that given a prefix label, it is possible to locate the corresponding PHT node via a single DHT lookup. This "direct access" property results in the PHT having several desirable features.

The main operation on the PHT is *lookup*, i.e., given a data item K , locating the leaf node whose label is a prefix of K. Since there are $D + 1$ distinct prefixes of K, the naive method is to perform a linear scan of these $D + 1$ nodes until the required leaf node is located. However, the direct access property of the PHT allows this to be improved upon by performing *binary search* on prefix lengths, which reduces the number of DHT lookups required from $D + 1$ to about $\log D$. Insertion or delection of data items involve a single PHT lookup, followed by splitting or merging of leaf nodes, if necessary. Range queries involve a single PHT lookup, followed by a sideways traversal of all leaf nodes whose prefixes overlap with the query.

To conclude, some of the important advantages of the PHT are summarized below. The PHT is *efficient*; lookups are doubly logarithmic in the size of the domain. It is *load-balanced*; nodes store only upto B data items, and binary search removes the need for lookups to go through the upper levels of the trie. It is *fault-resilient*; a leaf node can always be located in $D + 1$ DHT lookups, independent of the failure of other nodes. Finally, the PHT is built entirely on top of the DHT lookup interface, and does not assume knowledge of nor require changes to the DHT topology or routing behavior, thus allowing it to run over any DHT.

*email `sriram@cs.ucsd.edu`

[1] For simplicity, it is assumed that the domain being indexed is $\{0, 1\}^D$, i.e., binary strings of length D.

PODC'04, July 25-28, St.Johns, Newfoundland, Canada.
ACM 1-58113-802-4/04/0007.

2. REFERENCES

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, A Scalable Content Addressable Network, In ACM SIGCOMM 2001.

[2] M. Ruhl and D. Karger, Simple and Efficient Load Balancing Algorithms for Peer-to-Peer Systems, In ACM SPAA 2004.

[3] J. Aspnes and G. Shah, Skip Graphs, In ACM-SIAM SODA 2003.

[4] P. Yalagandula and J. Browne, UT CS Tech.Report TR-04-18.