Performance Analysis of BSTs in Local DNS Caches

Ruojun Hong

November 10, 2015

Abstract

This paper does performance comparison and analysis on three binary search tree (BST) based data structures: unbalanced BSTs, red-black trees, and splay trees, by imitating the behavior of DNS queries made to local DNS caches using a real world dataset. The dataset, being DNS queries collected from a Wireshark experiment, is maintained as original order, shuffled, and sorted, and tested with three different BSTs respectively. For every tree structure tested by each dataset, we measure the lookup time and insertion time it takes to respond to the entire set of queries. Our result indicates that splay trees are the most appropriate for DNS caches.

1 Introduction

The Domain Name System(DNS) resolves domain names to IP addresses and is implemented as a hierarchical and distributed database. DNS caching allows DNS servers or individual clients to locally store the DNS records and reuse them in the future, thereby decreasing the need of recursion steps and new queries to name servers. In this paper, we focus on the efficiency of different tree data structures that might be used by local DNS caches: unbalanced BSTs, red-black trees and splay trees. Approximately 36,000 packets under DNS or MDNS protocol are captured on the Ithaca College wireless network at the campus center after an estimated experiment time of 4 hours, during which 9,127 packets are FQDN[1]queries made from individual end systems and hence our dataset. We process the same data but order them by three different ways: original, shuffled and sorted, then send them into three different tree structures respectively to test their performances. The insertion time and lookup time are measured whereas the delete time is omitted for the reason of cache flushing. The analysis of results leads to the conclusion that splay trees are the most appropriate for local DNS caches.

The remainder of this paper is organized as follows. Section 2 describes the hypotheses and the methods we are using, including the details of the experiment; section 3 demonstrates and interprets the results of the experiment. Section 4 discusses and analyzes the experiment, and comes up with concerns and future possibilities.

2 Methods

To test the efficiency of different tree data structures used in local DNS cached, we imitate the behavior of DNS queries made to the cache. The dataset is obtained by running Wireshark in monitor mode on the Ithaca College wireless LAN. Around 36,000 packets under DNS protocol from a current randomly-selected channel are captured during an arbitrary 4-hour period. By applying the filters, we get 9,127 FQDN queries made from different individual clients. We maintain the original order, sort them alphabetically and shuffle them completely to get three datasets with the same entries but different orders. The pattern of original queries is not predictable.

[1]Fully qualified domain name(FQDN):uniquely identifies the host's position within the DNS hierarchical tree by specifying a list of names separated by dots in the path from the referenced host to the root.

Among three tree structures, BSTs don't have any balancing rules; red-black trees have stricter balancing rules than splay trees do. We assume red-black trees and splay trees have a trade-off between updating(insertion) time and lookup time. Splay trees will respond much faster to sorted data, because every time a node is accessed in splay trees, it gets "splayed" to the root of the tree, and therefore clustered items are likely to be stored near the root. The performance of BSTs is assumed to be acceptable for original and shuffled data because BSTs have a high probability of remaining balanced with natural, random insertions.

We use some well-known Java implementations[2] to build the trees. When a query that includes an FQDN is passed in, we first search in the tree to find the key, if the key doesn't exist, we then insert the key-value pair into the tree. After all 9,127 FQDN queries are processed, we sum up the lookup time and insertion time in milliseconds. The numbers of lookup and insertion operations are also recorded, but they remain constant after all operations are done since the queries are exactly the same. We don't take deletion into account because in reality the DNS cache flushes periodically on its own.

3 Results

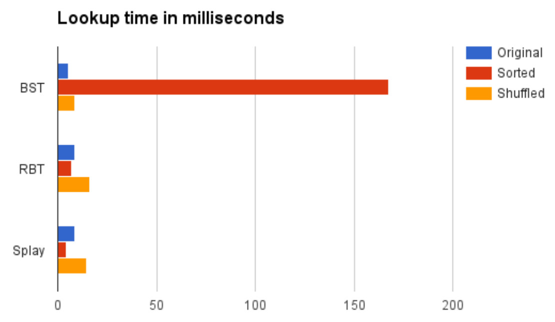
The tables below demonstrate lookup results and insertion time in milliseconds(rounded to 2 decimal places):

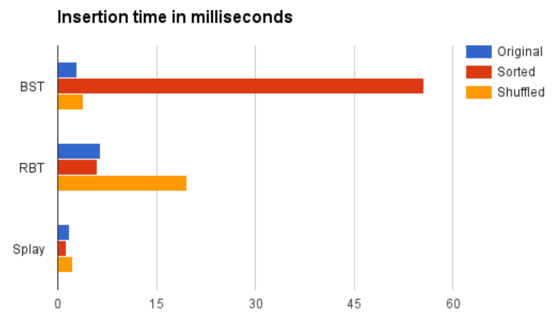
Lookup time(ms)	Original	Sorted	Shuffled
BST	5.64	167.59	8.45
RBT	8.53	6.84	16.30
Splay	8.82	4.21	14.56

Insertion time(ms)	Original	Sorted	Shuffled
BST	2.91	55.62	3.83
RBT	6.42	5.93	19.68
Splay	1.75	1.35	2.21

[2]The implementations are open-source, in Algorithms, 4th Edition, by Robert Sedgewick and Kevin Wayne.

The results of tables above are illustrated in the figures below:





Most of our hypotheses are proved by the results above. The most significant disadvantage of BSTs compared with the other two trees is their behavior responding to sorted queries. This is apparently caused by their lack of self-balancing rules. BSTs exceed the other two trees slightly in lookup time with both original and shuffled data. Red-black trees take the most time to insert while splay trees take the least time, which is also obvious because red-black trees have strict balancing rules while splay trees insert the node to the root. Splay trees perform better than red-black trees according to both lookup time and insertion time.

4 Discussion

From the experiment, we observe that specific balancing rules of each tree structure do influence its performance in a significant way. From the results, we can deduce that the pattern of original DNS queries appears to be clustered. This is reasonable because well-known domain names, like www.google.com, www.amazon.com, etc., are likely to be queried frequently. We assume splay trees respond to those queries faster because its balancing rule takes advantage of locality in the keys used in incoming lookup requests. Although BSTs do show slightly better performance than splay trees, we attribute the fact to the sampling method that we use to get the data. The number of packets captured by Wireshark is not enough for splay trees to demonstrate their advantages. Initially, we tried to reach the DNS administrator of Ithaca College Information Technology Services to get access to the campus DNS server or some centralized routers to get DNS queries from the campus. Unfortunately due to time constraints and privacy concerns, we are unable to access those data. In the future, we are looking forward to more scientific and appropriate sampling methods to conduct research on.

References

[1] B. Pfaff, "*Performance Analysis of BSTs in System Software*".

[2] R. Sedgewick and K. Wayne, *"Algorithms*", 4th Ed.