Thesis Proposal: Supersuper Fast and Efficient Hashing

Chenyao Lou

October 5, 2019

1 Introduction

The Hash table is a very crucial important component for systems. For example, the load balancer uses the hash table to decide the associated backend server for specified connections. Or it can store the Forwarding Information Base (FIB) for network applications [Dong, CuckooSwitch] like SDN. The lookup of FIB is the biggest bottleneck either for software packets forwarding or hardware packets forwarding. Thus, developers desire a high-performance hash table for such tasks.

Besides, the space efficiency is also an important metric for hash tables. It has two benefits. First, the small query structure can fit into the CPU cache, which is ten times faster than the main memory (DRAM). Thus the lookup performance an be accelerated further with the same hardware. Second, a space efficient hash table storing more items benefits memory-constrained tasks.

Therefore, we propose a new space efficient hash table SSFE (Supersuper Fast Hashing) design with the high lookup performance. The basic functionality of a hash table is that, given a set of tuples $\{(k_0, v_0), (k_1, v_1), ..., (k_{n-1}, v_{n-1})\}$, the hash table can give v_i for the question $q(k_i)$. There are variations of this definition: For a $k \notin \{k_0, ..., k_{n-1}\}$, (1) the hash table returns $\perp$ (key is not existing), or (2) returns a random $v \in \mathcal{V}$. SSFE is designed for the second definition, i.e. it returns a random value when the query key has never been seen before.

SSFE consists of two structures, the *Query Structure* and the *Maintenance Structure*. The query structure has less memory footprint than the maintenance structure, and only serves for the lookup. It does not store any keys of tuples. That is why quering a non-existing key results in a random value. The maintenance structure is used to store all tuples so that it supports update operations. This seprating structures design fully utilize the CPU cache since the query structure stores more information by discarding keys. The system can update the query structure on a dedicated core without sacrificing the lookup performance.

In section 2 we will disscuss the full design of SSFE, and engineering techniques. We compare SSFE with existing works in section 3.

2 Design

3 Related Work

4 Conclusion

References