Building Skew Resistant Hash Tables

Güneş Aluç

SAP Labs, Canada
gunes.aluc@sap.com

1 Introduction

Conventionally, hash tables rely on hashing techniques that are static. This means that the hash value using which a key gets inserted into the hash table is the same as the one it gets looked up with. Consequently, when the hash table is full (or a pre-determined load factor is reached) or when the number of buckets in the hash table need to be changed, all or a subset of the keys that are stored in the hash table are redistributed using a new hash function with the intention of distributing the keys uniformly across the newly adjusted space. Consistent hashing [2] is an improvement that aims to minimize the number of keys that are redistributed every time the hash table grows or shrinks, by relying on a hash function that is consistent with the previously used hash function. These techniques work well when the data are uniformly distributed. Unfortunately, data are naturally skewed, and hash functions are inherently imperfect. Therefore, when static hashing is used in a hash table, hot spots – which are inevitable – cause keys to unnecessarily and frequently get redistributed, resulting in performance stalls.

2 Contributions

In this presentation, we introduce our early work on skew resistant hash tables where a continuously adaptive hash function is used to determine key placement. This means that, in contrast to conventional hash tables with static hashing, the hash value using which a key gets inserted into the hash table is not necessarily the same as the one it is looked up with[1]. Because of this relaxed constraint, the underlying hash function can find the optimal placement for the key to be inserted based on the current distribution of keys in the hash table. Furthermore, it can adapt to changing distributions. In practice, what this means is that if there are hot spots (e.g., keys that repeatedly hash to the same set of hash buckets), the hash function will assign different values to these keys, thereby, dissolving the hot spot. To facilitate the computation of the hash function based on the current distribution, as well as to maintain an ongoing summary of the distribution, our work relies on a combination of static hashing, counting using a sufficiently large number of buckets and binary segment trees. When the hash table needs to grow, instead of relocating the keys, it relies on a consistently identifiable coordinate scheme. Insertions, lookups and deletions are detailed, and a lazy-relocation method is introduced for cases when the distribution changes significantly such that the hash value using which a key gets inserted into the hash table starts to deviate significantly from the value it is looked up with. In short, the key aspects of skew resistant hash tables can be summarized as follows:

- Conventional hash tables rely on static hashing techniques; skew resistant hash tables rely on dynamic hashing techniques that can adapt to changing data distributions and workloads.
- Conventional hash tables adapt to changes in the data distribution by redistributing the keys which is expensive; skew resistant hash tables adapt by making adjustments to the internally used hash function which is cheaper.
- Conventional hash tables aim to minimize key redistribution by using consistent hashing techniques; skew resistant hash tables rely on consistently identifiable coordinate schemes.
- Conventional hash tables eagerly relocate keys when keys are inserted; skew resistant hash tables lazily relocate keys when the keys are looked up if and only if the hash of the key that is being looked up has deviated significantly from its original insertion position.

References

[1] G. Aluç, M. T. Özsu, and K. Daudjee. Building self-clustering RDF databases using Tunable-LSH. *The VLDB Journal*, 28(2):173–195, 2019.

[2] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, 1997.