Brandon Lasher
CSIT565 – Midterm Paper Summary

Self-selecting, self-tuning, incrementally optimized indexes: Summary

Introduction

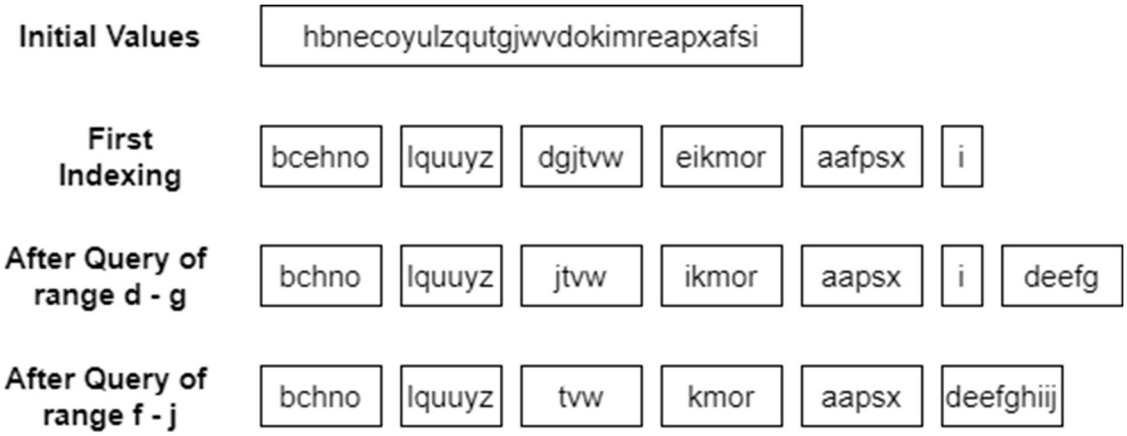
Modern relational database warehouses have a size problem. With hundreds of tables and even more columns, the number of possible indexes becomes unwieldy. This makes the creation and optimization of relevant indexes extremely important, though it can be very difficult. Due to unpredictable queries, defining too few or non-useful indexes causes large latency due to database scans and defining too many indexes can place a high demand on resources. A common approach is to modify and create the indexes based on the current workload and incoming queries. In *Self-selecting, self-tuning, incrementally optimized indexes* [1] they authors explore the older method for building and optimizing indexes on the fly and then explain how they've made improvements to it both in memory usage and speed.

Background

The method proposed has its basis in database cracking. Database cracking utilizes focused, incremental, and automatic optimization of indexes. The more a key range is queried more the indexes are optimized for its representation. There are several methods for database cracking but he general idea of it remains the same. When data is queried, an index is made on the column selected and that index is arranged in unsorted partitioned key ranges. When it is subsequently queried the incoming key ranges are used to flag which partitions to search and further refine. If both keys fall with a given partition it is, then divided into 3 partitions based on the start and end of the incoming keys. If only 1 key is in each partition, then 2 new partitions are made for each of the incoming keys, segmenting on the key value. As a result of this method, key ranges which are never queried are never partitioned beyond the initial creation of the index. A major drawback to database cracking is that it will take several iterations and queries for the index to reach its optimized state. When an initial index is created it does not perform very well.

Adapting Merging

Adapting Merging builds on database cracking with the idea of using the range selection to do an additional sort and merging. The steps are as follows. First, the initial index is created with sorted but partition values overlap. As ranges are queried the data indexes are pulled out in order create a new sorted partition, which is appended on to the partition list. When the next query occurs, it is merged with the sorted partition. A summary of the steps is shown in the figure below.



If all values in the index have queried, the result will be a single sorted partition. If a key range is never accessed, it is never sorted and as such no time is wasted on it. It is important that the indexing service, like in database cracking, keeps track of ranges and indexes have already been sorted, as to not waste time looking in the other partitions.

Conclusions

Traditional indexing has the problem of computation time and the possibility of being poorly chosen. Dynamic indexing techniques such as database cracking and adaptive merging are attempts to alleviate this issue by having the index created dynamically based recent database queries. Database cracking uses a partitioning technique similar to quicksort generate an optimized index for ranges which are queried more than once. Each time it I queried new partitions are made and the search of that index becomes quicker. Adaptive merging builds upon database cracking. However rather than simply creating new partitions, it creates a sorted partition of the values which have been queried. And as new ranges are queried, they are merged in via a merge sort to the sorted partition. The end of the paper includes a large section on performance comparing database cracking and adaptive merging. Comparing their overhead per query in various types of situations. In all cases adaptive merging performed better and reached an optimized state faster than that of database cracking.

References

[1] G. Graefe and H. Kuno, “Self-selecting, self-tuning, incrementally optimized indexes,” in *Proceedings of the 13th International Conference on Extending Database Technology - EDBT ’10*, Lausanne, Switzerland, 2010, p. 371. doi: 10.1145/1739041.1739087.

[2] “Merging what’s cracked, cracking what’s merged: adaptive indexing in main-memory column-stores: Proceedings of the VLDB Endowment: Vol 4, No 9.” https://dl.acm.org/doi/10.14778/2002938.2002944 (accessed Oct. 19, 2022).