Efficiently applying Fast Succinct Tries in OLTP Systems

Abstract
Online Transaction Processing Systems (OLTP) have high requirements regarding throughput and are insert and update-intensive. Moreover, they are used by a multitude of users, thus requiring concurrent access. As main memory still is comparably expensive, data structures that can index data with a memory consumption near the information theoretic minimum are a hot topic in data structure engineering research. One such recently published data structure is the Fast Succinct Tree (FST). However, the FST is a static data structure, thus requiring an extension to be used in OLTP Systems. Therefore, we propose an architecture that applies inserts and deletes to an FST in batches. With this project, we want to evaluate, whether the proposed architecture is still able to deliver adequate performance, while at the same time significantly reducing memory consumption for indexing in OLTP Systems.

Problem: The FST is a static data structure that does not support inserts, deletes or updates, making it unsuitable for OLTP systems.

Goal: Develop an architecture that benefits from the minimal memory consumption of the FST, while supporting the requirements of a OLTP System.

Solution: We apply inserts, deletes and updates to the dynamic Adaptive Radix Tree (ART) that is optimized for performance. We then regularly apply batch merges to the FST, thereby selecting an interval that minimizes the number of merges, while at the same time keeping the ART significantly smaller than the FST, in order to keep up the memory savings.

Architecture Proposal

Queries

Insert

Approximate Membership Filter
Checks if key is in ART
\[\text{count}(	ext{ART}) \ll \text{count}(\text{FST})\]

Deletions
\[\text{delete(key)} \Rightarrow \text{key in ART}\]
\[\text{ART.delete(key)} : \text{ART.insert(key, 1)}\]

Updates
\[\text{update(key, val)} \Rightarrow \text{key in ART}\]
\[\text{ART.update(key, val|1)} : \text{ART.insert(key, val|1)}\]

Possible Limitations
As the FST requires rebalancing/reordering depending on newly inserted keys, it has to be investigated whether such a merge procedure exists or whether applying changes requires a complete rebuild of the FST. We then have to evaluate whether the memory - efficiency tradeoff that is introduced by the filter is adequate or whether the filter might be omitted, while still maintaining acceptable performance.

Fast Succinct Tries (FST)

Frequently accessed top layers optimized for performance.
(child node search = 1 array lookup)

Less frequently accessed bottom layers optimized for memory reduction.

Memory for Indexing 25M Emails

600MB

ART

400MB

FST

200MB

Latency for Point Queries

750ms

ART

500ms

FST

250ms

Multi-User Support

Goal: Allow concurrent accesses, also during merge.

Solution:
1) \(\text{ART + Optimistic Lock Coupling (Leis, 2016).}\)
2) Static FST is natively concurrent.
3.1) Apply changes that incoming during the merge phase to a new ART -> current ART becomes read-only.
3.2) Apply modifications from old ART to FST. Incrementally apply changes *(see limitations).*

Rohan Sawahn
ITSE Master Student
Database Research Lecture 2021/22
Hasso Plattner Institute, Potsdam, Germany

Sources
FST detail image taken from (1)