Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes

Karl Allik
Institute of Computer Science
University of Tartu
karl.allik@ut.ee

Abstract—Improving memory efficiency of main-memory database management systems (DBMS) reduces overall operational and hardware costs for anyone wishing to use main-memory DBMSs for high performance on-line transaction processing (OLTP). One way of reducing the memory consumption of these systems is to improve the memory efficiency of index-structures. "Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes” [Zhang, Andersen, Pavlo, Kaminsky, Ma and Shen, 2016] proposes a new index architecture for in-memory DBMSs that can reduce the memory overhead of index-structures by up to 70% while achieving comparable throughput to the original index-structures. By demonstrating the performance of the proposed architecture on an existing main-memory DBMS (H-Store), the authors prove that the dual stage hybrid index architecture can be used in practice for significantly improving the memory-efficiency with low performance overhead.

I. INTRODUCTION

Database indexes are crucial for the performance of any database, but introduce a large memory overhead, which presents serious drawbacks especially for main-memory databases. When running TPC-C on an in-memory DBMS called H-Store for example, indexes consume about 55% of the total memory [2]. The authors of the presented article propose a new dual-stage hybrid index architecture, which employs a dynamic and a more compact static stage to substantially reduce the size of the index-space while introducing little to no performance overhead [1].

The article also describes a set of guidelines for Dual-Stage Transformation (DST), used for converting any order-preserving index structure into a hybrid index. To performance of hybrid indexes is evaluated by applying DST guidelines to four order-preserving indexes: B+tree, Massstree, Skip List, and ART.

Using Yahoo! Cloud Serving benchmark (YCSB) and built-in H-Store OLTP DBMS benchmarks, it turns out that the memory consumption of index structures can be reduced up to 70% with no significant performance overhead [1].

II. THE DUAL-STAGE HYBRID INDEX ARCHITECTURE

As seen on Figure 1, the dual stage hybrid index architecture consists of two stages: the dynamic and the static stage. All new entries are added to the smaller dynamic stage, which keeps the most recent entries for faster access and modification. The merge process is triggered periodically to migrate older entries from the dynamic stage to the static stage. Because the static stage uses a more space-efficient data structure to hold the majority of the indexes, it does not allow direct key additions or modifications and has to be updated in batches through the merge process. Each read request to the hybrid index is served by searching the stages in order. A Bloom filter is used for the keys in the dynamic stage, to speed up these read requests. If a key is found in the Bloom filter, the index checks the dynamic stage and if not found, the static stage is checked. If the key is not found in the Bloom filter, the dynamic stage is passed and the key is directly searched from the static stage.

A. Update and remove operations

If the value is present in the dynamic stage, it is updated in place. Otherwise a new value is inserted to the dynamic stage, which effectively overwrites the old value in the static stage, as it is overwritten during the next merge. When removing a value, it is removed immediately if found in the dynamic stage. Otherwise the value is searched and deleted from the static stage during the next merge process.

B. The merge process

To avoid implementing a more complex non-blocking merging, a more general blocking merge algorithm was chosen. Taking into account the insert-intensive workloads patterns of OLTP databases, the authors chose to implement merge-all, which is a more general and suitable alternative to merge-cold. The merge process is triggered automatically when the size ratio of the two stages passes a certain threshold. Based on analysis, 10 was chosen as the default ratio by the authors [1].
III. DYNAMIC-TO-STATIC RULES

Many different compact data-structures can be used to construct the dual stage index-structure. To provide better interoperability and merge performance between the stages, same data structures are used for both stages. Since the static stage is a read-only structure, it can be made more memory efficient, by applying the following guidelines presented in the article:

- **Rule #1: Compaction** - duplicate content is removed and all allocated memory blocks are filled to 100% capacity.
- **Rule #2: Structural Reduction** - all pointers and other elements unnecessary for a read-only structure are removed.
- **Rule #3: Compression** - internal nodes or leaf nodes are compressed. (This step creates excessive performance overhead to be used in practice according to the results)

IV. EVALUATION

Using the proposed DST guidelines, five hybrid indexes were created and evaluated: Compact B+tree, Compressed B+tree, Compact Massstree, Compact Skip List and Compact ART. The stand-alone performance of the indexes was measured using the YCSB. The performance inside an actual DBMS was measured by implementing the indexes into an in-memory DBMS called the H-Store.

A. Compaction

It was established that the read throughput for the compact indexes after applying DST is up to 20% higher in most cases compared to the original data structures, thanks to smaller structural overhead. The memory footprint was reduced by 30-71%. The Compact Masstree showed the biggest space savings (70%) while ART had the best performance.

B. Hybrid indexes vs. originals

To emulate typical large-scale cloud service index workloads, the default workloads of YCSB were used: A (read/write, 50/50), C (read only), and E (scan/insert, 95/5). For all workloads 3 different key-types were used: 64-bit random int, 64-bit mono-inc int and email string. When comparing original index structures to their hybrid counterparts it was established that the hybrid indexes provide comparable throughput while occupying 30-70% less memory. Due to the periodic merging and the key uniqueness checking, insert-only workloads showed a 30% throughput drop when using hybrid indexes. However in the case of read/write workloads, hybrid indexes almost always outperformed the original structures for all key types thanks to the dynamic stage. Read-only workloads suffered only a slight performance degradation, while having to check both stages for an index.

C. Full DBMS evaluation

From all presented data structures the performance in a full DBMS setting was measured only for the Compact Hybrid and Compressed Hybrid B+tree as the Original H-Store uses B+tree as its default index data structure [2]. The performance was evaluated using three H-Store built-in benchmarks: TPC-C [3], Voter and Articles. As seen from figure 2, the Hybrid B+tree consumed 40-55% less memory than the original index structure during each benchmark trial, while throughput was decreased at most by 10%.

V. CONCLUSION

The authors demonstrated a dual-stage hybrid index architecture, which combines two differently optimized data structures, that can be used to reduce the size of the index structure up to 70% while maintaining comparable throughput. By using the proposed DST guidelines, any existing index-structure can be converted into a hybrid index architecture to improve memory-efficiency of an existing main-memory DBMS. Based on a full DBMS evaluation of the presented architecture, the authors demonstrated that the dual stage hybrid indexing can be implemented into an existing state-of-the-art main-memory database to save up to 50% of the memory normally consumed by index-structures.

REFERENCES