TRIMMER: Transitive Indexing in Main-Memory Systems

ABSTRACT

We study main-memory indexing tailored for modern applications that need to do both efficient reads and efficient updates (both arriving at massive amounts and rates). We show that state-of-the-art main-memory static indexing methods require too much time and workload knowledge in order to go through the preparation phase for a new batch of data, i.e., in order to create a main-memory tailored tree (or trie) structure or to sort a main-memory resident array. Adaptive indexing solves this problem by avoiding expensive initialization costs and incrementally refining indexes; however, we show that state-of-the-art adaptive indexing methods are not resilient, i.e., they fail to maintain their adaptive properties when they are phased with massive sequences of queries and updates. As a result no current method can support efficient reads and writes.

We identify the root of the problem at the rigid design of indexing methods so far. We observe that for each stage at a workload there is a different indexing design which is the best fit. In this paper, we present Trimmer, a new main-memory indexing approach that we call transitive indexing. Trimmer changes each shape as the workload evolves. Initially, an index is a collection of unordered vectors. As more queries arrive, during query processing, Trimmer refines the hot vectors and starts inducing order and structure. Gradually, query after query, Trimmer morphs into a Trie structure for the hot part of the data. As a result Trimmer can absorb massive updates while at the same time it provides fast lookup times and it does not need any initialization cost. A detailed experimental analysis on both synthetic and real-life scenarios from the astronomy domain shows that Trimmer successfully overcomes the limitations of existing adaptive and static main-memory indexing – it enables resilient, adaptive and interactive data exploration under massive updates.

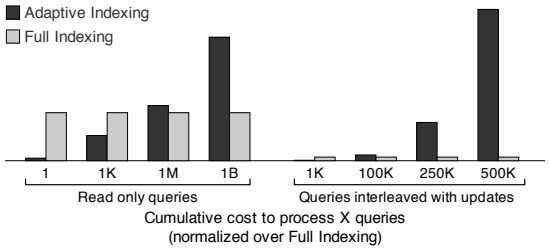

Figure 1: The need for transitive indexing: A different design is best for different stages of a workload.

1. INTRODUCTION

Main memory sizes have grown significantly to the degree that they can hold big chunks of data main-memory resident, often all the data of modern database instances. This has lead to an increased attention towards main-memory tailored techniques. Even though data can be accessed fast in main-memory, e.g., with scans over dense arrays, still proper indexing provides significant benefits for range or point queries.

Past research has shown that main-memory tailored indexing techniques can significantly outperform standard techniques such as binary search over a dense sorted array [6, 5].

The Problem: Read-write Intensive Workloads. In this paper, we address the need of modern applications to handle workloads which require both efficient read and efficient update support in the same system. There is an increasing attention to this problem at various levels, e.g., in terms of database architectures with efforts such as SAP HANA [1] and Hyper [4] or also at the indexing level with efforts such as ART main-memory indexing [5]. A problem with traditional static indexing approaches is that one has to know exactly which data should be indexed and then also have the time and resources to go through a very expensive indexing phase. As applications become more and more real-time and with ad-hoc querying demands, such static approaches become a bottleneck to having a quick access to new data. A number of adaptive indexing approaches have been proposed lately to address exactly this problem of static indexing [3, 7]. With adaptive indexing, indexes are built and refined incrementally during query processing by treating each query as an advice of how data should be stored. However, we show that although existing adaptive

indexing techniques do provide significant improvement for ad-hoc workloads, minimizing or even eliminating initialization costs, they are not resilient in the same way that non-adaptive indexing is for read-write intensive workloads.

Motivating Example. Figure 1 demonstrates this problem: it depicts the relative time needed (y-axis) to answer the first X queries (x-axis) in a long sequence of random range queries and updates over a single column of 10^8 tuples (10 updates arrive every 10 read queries). Each graph is normalized (independently) over its static indexing costs. In this case, for static indexing we use the state-of-the-art in memory index ART [5] while for adaptive indexing we use the state-of-the-art stochastic cracking index [2]. Figure 1 shows that while early in the query sequence adaptive indexing significantly outperforms static indexing, as the sequence evolves, adaptive indexing loses its advantage over static indexing, i.e., it is not resilient. The first query for adaptive indexing is more than 10 times faster than static indexing; Full indexing needs to fully index the column, while adaptive indexing performs only small index refinement actions with every query. However, as more queries and updates arrive, adaptive indexing eventually becomes more than 10 times slower than static indexing.

Ideally, we would like to maintain the performance of adaptive indexing early in the query sequence and the performance of static indexing later on.

Contributions: Transitive Indexing. Our contributions are as follows.

- We propose a new main-memory indexing research area, termed *transitive indexing*. The idea is that a data structure representing an index does not have a static shape. Instead, its shape morphs over time as the workload evolves.
- We present Trimmer, a transitive indexing approach which initially requires no preparation or initialization steps. Data is simply appended in the form of vectors with no order forced both inside each vector and across vectors. As the workload evolves, Trimmer starts forcing order and structure. Each vector is independently indexed and refined, while data starts flowing from vector to vector. Gradually, the vectors start morphing into a Trie structure optimized for main-memory. Only the hot part of the data morphs and everything happens during query processing without any need for any administration or control.
- We show that Trimmer maintains all the good properties of both existing adaptive indexing approaches and those of non-adaptive approaches, i.e., it is lightweight, it continuously adapts by incrementally refining indexes and it requires zero workload knowledge and preparation costs. At the same time it is resilient, i.e., it maintains its performance advantages unfettered even under long strings of queries and continuous data updates; it is geared towards continuous on-the-fly data exploration under massive updates.

We experimentally demonstrate the advantages of Trimmer over existing static indexing and adaptive indexing techniques. We use both synthetic benchmarks as well as real world data and queries. For example, in experiments with a 4 Terabyte instance of data and queries from the Sloan Digital Sky Survey/SkyServer from the astronomy domain, we show that Trimmer can handle a combined workload of $15 * 10^4$ queries and $5 * 10^8$ updates in roughly 1 minute, while state-of-the-art adaptive indexing needed more than 16 hours and state-of-the-art non adaptive indexing needs ??.

2. REFERENCES

[1] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[2] F. Halim, S. Idreos, P. Karras, and R. H. C. Yap. Stochastic database cracking: Towards robust adaptive indexing in main-memory column-stores. *PVLDB*, 5(6):502–513, 2012.

[3] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *CIDR*, pages 68–78, 2007.

[4] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE*, 2011.

[5] V. Leis, A. Kemper, and T. Neumann. The adaptive radix tree: Artful indexing for main-memory databases. In *ICDE*, pages 38–49, 2013.

[6] J. Rao and K. A. Ross. Making b^+-trees cache conscious in main memory. In *SIGMOD*, pages 475–486, 2000.

[7] S. Richter, J.-A. Quian-Ruiz, S. Schuh, and J. Dittrich. Towards zero-overhead static and adaptive indexing in hadoop. *VLDBJ*, 2013.