Modern Multi-Dimensional In-Memory Index Structures

Student Research Project Exposé

Quentin M. Kniep — `kniepque@hu-berlin.de`
Supervisor: Ulf Leser

January 7, 2021

1 Introduction

For modern analytical data processing the main task is efficient querying of multi-dimensional data in a main memory database setting. Whereas online transaction processing (OLTP) uses mostly exact match queries and does not always use multi-dimensional data for querying, the focus in online analytical processing (OLAP) is especially on multi-dimensional range queries (MDRQs) [14].

Simple ideas for building multi-dimensional indices only work for certain settings or are limited in their performance. For example, linearizing multiple dimensions onto a single-dimensional index via a space-filling curve has performance problems [7]. An approach like hashing all relevant fields only works well for point queries, not for range queries.

Early examples of multi-dimensional indexing structures (MDISs) are the R-Tree [8], the R*-Tree [2], the kd-Tree [3], and VA-files [17]. We will look at the more recent developments instead. Specifically, we will analyze the following research prototypes:

- BB-Tree [14] (adapts B-Tree concepts for in-memory multi-dimensional use)
- Elf [5] (new fixed-depth tree-based approach that is cache conscious)
- Cracking KD-Tree [11] (applies DB cracking to adaptively build a kd-Tree)
- QUASII [13] (incrementally slices the space to adaptively build an R-Tree)

All of these reasonably claim to perform much better in our setting than their older competitors. We will evaluate their performance in direct comparison by performing a set of benchmarks, with different real-world and synthetic datasets and workloads.

Finally, there are claims that an adaptive indexing approach (for unpredictable workloads) has lower upfront index-building cost, lower cost in total, and thus scales better than a static index [11]. It is also argued that when only part of the data was used, building a full index would be wasteful. [13]. There are also contrary claims, that adaptive indexing only pays off for certain workloads: As Broneske [5] states, "[o]nly when enough diverse queries are issued, cracking pays off in the long run." Here he specifically refers to QUASII [13] and the Cracking KD-Tree [11], two of the index structures we look at. Therefore, we will also evaluate the difference between adaptive indexing and static index structures, and the differences between these two adaptive indexes.

2 Related Work

Looking at the early developments of MDISs, there are two very extensive surveys [4, 7]. The MDISs considered in these surveys were designed for older hardware featuring a single CPU core and small main memory. Many trade-offs are therefore made in a way which is not efficient in a main memory database setting. On modern hardware, Das et al. [6] and Sprenger et al. [15] showed that an accelerated in-memory full-table scan is faster than the most popular early MDISs (R*-Tree [2], kd-Tree [3], VA-file [17]), even for pretty low selectivities (around 1%). The main reason is that scanning can easily exploit SIMD parallelization and use CPU caches efficiently.

There also is one survey by Alvarez et al. [1] looking at more recent developments. However, it only considers different adaptive indexing approaches, and does not include in its comparison any of the indexing structures (ISs) we look at.

To the best of our knowledge, the dissertation by Broneske [5], which describes the Elf, contains the only direct comparison of any of the MDISs we will compare here; namely, it compares the Elf with the BB-Tree.

Out of the MDIS papers, the selection of datasets used in the Cracking KD-Tree paper's [11] benchmark is the most comprehensive. It consists of a diverse combination of real-world datasets and synthetic data generation methods, all of which they provide together with the implementation. Broneske [5] uses selected queries from the TPC-H benchmark to evaluate his IS, the Elf. Pavlovic et al. [13] use a real-world neuroscience dataset, and skewed and uniform synthetic datasets, with fully synthetic query workloads.

Additionally, very fitting for our evaluation is the *genomic multidimensional range query benchmark (GMRQB)* proposed by Sprenger et al. [15]. It is based on real-world MDRQs on genomic variations data, which is publicly available. Also, it was already mentioned as future work by the authors of some of the ISs [11, 5].

Another interesting development are Machine Learning based ISs [12, 16]. We will not look at these in this survey, as this branch of ISs is a very new and loosely populated field so far.

Theoretical background for analyzing and evaluating indexing schemes is given by Hellerstein et al. in two papers [9, 10]. They define storage redundancy and access overhead as key metrics for the evaluation.

3 Research Questions

The MDISs mentioned above all claim to perform better than most earlier developments. They compare themselves mostly with the R-Tree [8], the R*-Tree [2], the kd-Tree [3], and VA-files [17]. However, almost no comparisons between them have been performed so far. The first research question for this project is thus:

How do the different index structures perform compared to each other?

We will focus on read-only workloads because the main use cases for MDISs are analytical [15], i.e. frequent time-critical requests are read-only and insertions are made in bulk and are mostly not user-facing. One subtask then is to reevaluate the already existing comparison between the Elf and the BB-Tree. Also an interesting performance metric for this is memory usage, not just query time.

Regarding adaptive indexing approaches (in our case: QUASII and the Cracking KD-Tree), we want to reevaluate the claims they make about performing better than static alternatives (in our case: Elf and BB-Tree). Further, assuming that adaptive indexing will pay off for some workloads but not for all possible workloads, we phrase our second research question as:

For which workloads does adaptive indexing outperform a static index?

One interesting subquestion we will answer here is: Does adaptive indexing come at essentially no cost (as is sometimes claimed in the respective papers), or are there workloads for which it is significantly worse than using a static index?

4 Methodology

All the reference implementations are written in C++, thus there should be no differences based on the programming languages used. Furthermore, we were already able to compile all implementations on the same system under the same compiler version (g++ 7.5.0, SUSE Linux), using the highest optimization level (O3).

Single-threaded benchmarks might be most meaningful, as the Cracking KD-Tree implementation for example is implemented as a single-threaded program only [11]. Multi-threaded benchmarks can also be achieved by horizontally partitioning the data [15].

We will focus only on multi-dimensional read-only data queries with a strong focus on range queries, but look at both geometric/spatial data (2–4 dimensions) and at high-dimensional data (10 or more dimensions). The benchmarking environment is a compute-server with the following specifications: four Intel E7-4880 (each: 37.5 MB Cache, 15 cores at 2.5 GHz, 30 threads via hyper-threading), 1 TB RAM, SuSE Leap 15.

We will gather and adapt to our needs some of the benchmarks mentioned under Related Work, especially the datasets and workloads also used by Holanda et al. [11], the GMRQB [15], and maybe TPC-H.

The Elf uses *uint32_t* (unsigned 32-bit integers) for both, tuple IDs and search keys (one integer for each dimension). On the other hand, the BB-Tree uses *uint32_t* only for tuple IDs, and uses a *float* (32-bit floating point number) in each dimension as search keys. QUASII and the Cracking KD-Tree also use a *float* in each dimension as search keys, but they use a *size_t* (most likely a unsigned 64-bit integer) value as tuple ID. We need to keep these differences in mind in direct comparisons and when building the benchmark.

References

[1] V. Alvarez, S. Richter, X. Chen, and J. Dittrich, "A comparison of adaptive radix trees and hash tables," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 1227–1238.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[3] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[4] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys (CSUR)*, vol. 33, no. 3, pp. 322–373, 2001.

[5] D. Broneske, "Accelerating mono and multi-column selection predicates in modern main-memory database systems," https://hu.berlin/PQWireGuard, 2019.

[6] D. Das, J. Yan, M. Zait, S. R. Valluri, N. Vyas, R. Krishnamachari, P. Gaharwar, J. Kamp, and N. Mukherjee, "Query optimization in Oracle 12c database in-memory," *Proc. VLDB Endow.*, vol. 8, no. 12, p. 17701781, Aug. 2015. [Online]. Available: https://doi.org/10.14778/2824032.2824074

[7] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 170–231, 1998.

[8] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 4757. [Online]. Available: https://doi.org/10.1145/602259.602266

[9] J. M. Hellerstein, E. Koutsoupias, D. P. Miranker, C. H. Papadimitriou, and V. Samoladas, “On a model of indexability and its bounds for range queries,” *Journal of the ACM (JACM)*, vol. 49, no. 1, pp. 35–55, 2002.

[10] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou, “On the analysis of indexing schemes,” in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1997, pp. 249–256.

[11] P. Holanda, M. Nerone, E. de Almeida, and S. Manegold, “Cracking KD-Tree: The first multidimensional adaptive indexing,” in *DATA 2018*, Jul. 2018, pp. 393–399.

[12] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska, “Learning multi-dimensional indexes,” *CoRR*, vol. abs/1912.01668, 2019. [Online]. Available: http://arxiv.org/abs/1912.01668

[13] M. Pavlovic, D. Sidlauskas, T. Heinis, and A. Ailamaki, “QUASII: Query-aware spatial incremental index,” 2018. [Online]. Available: http://infoscience.epfl.ch/record/254867

[14] S. Sprenger, P. Schäfer, and U. Leser, “BB-Tree: A main-memory index structure for static multidimensional workloads,” https://www.wireguard.com/papers/wireguard-formal-verification.pdf, 2017, accessed June 25, 2020.

[15] ——, “Multidimensional range queries on modern hardware,” in *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, 2018, pp. 1–12.

[16] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, “Learning to hash for indexing big dataa survey,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 34–57, 2015.

[17] R. Weber, H.-J. Schek, and S. Blott, “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces,” in *VLDB*, vol. 98, 1998, pp. 194–205.