Thesis Proposal: Multi-core In-Memory Key-Value Storage using Hash-Table Based Indexing.

Jakob Hanehøj

January 29, 2016

1 Thesis Research Question

How can one design an in-memory multicore key-value storage for modern hardware, using hash-table based indexing, which effectively handles arbitrary length keys, including keys with long shared prefixes? How advantageous would such a system be over current comparable multi-core systems?

2 Overview of the Proposal

In many Online Transaction Processing Systems (OLTP), the key-value storage is a key service, thus increasing the performance of it will directly increase the performance of the system. With increasing growth in volume, variety and performance demands of OLTP applications [2], the performance of the key-value storage system has increasing importance. One way of improving resource utilization is by using various specialized storage systems for different workloads.

Stonebraker et al. have shown that by using shared-nothing cluster of machines, OLTP systems can be deployed completely in main-memory [1]. This leads to large performance advantages compared to classical relational database management systems (RDBMS), as it leaves out the high I/O costs of traditional disk-based system. It does however also represent a new challenge to the way data should be stored and indexed, in order to be processed efficiently, due to the poor cache utilization of the traditional data structures. This has lead to a considerable amount of research on index data structures for main-memory systems during the last decade [3]. Among the most interesting data structures for main-memory systems are the latch-free Bw-tree [6], the adaptive radix-tree ARTful [5] (ART for short), as well as Masstree [4], a trie-like concatenation of B^+-trees, each of which handles a fixed-length slice of the variable-length key.

Masstree provides a persistent flexible data storage model, which performs well on workloads in which many keys share a common prefix, due to the trie-like structure. It uses achieves fast concurrent operations using a scheme, in which lookups use no locks, thus not invalidating shared cache lines, achieved by optimistic concurrency control (OCC). Additionally, the lookups can be done in parallel with most inserts and updates, as writes only acquire local locks on the involved nodes, hence also allowing writes to different parts of the tree to happen concurrently. Consistency and durability are achieved by logging and checkpointing to persistent storage, thus securing persistence and the ability to recover upon crashes. Finally, Masstree shares the tree among all cores in order to preserve load balance when the workload is skewed, uses a wide-fanout tree to reduce the tree depth, and prefetches nodes to overlap fetch latencies.

However, Wang et al. have shown that using a hash-based indexing can outperform a equivalent B-tree, by abstracting their key-value storage into either ordered access (using a B^+-tree) or unordered access (using a hash table) [7]. Richter et al. have found that for main-memory databases, a well-chosen hash table index is considerably faster for point-queries than a range of modern tree-structure indexes [8]. They provide an analysis of seven different factors to take into consideration when choosing the hash function, of which the five primary factors mainly considers the workload to be handled. Additionally, Thorup has recently shown good results using multiple variations of tabulation hashing [9], which could be incorporated in a new hash-based indexing method.

In summary, Masstree has combined many interesting ideas into one structure, optimized for multi-core systems, and handles variable-length keys, while the tree-structure have been shown to be inferior to a hash-based index structure for indexing in key-value storages. Moreover, none of the previous in-memory key-value storages using hash-based indexing has both targeted a multi-core implementation, while being able to handle variable length keys.

The thesis therefore aims to design an indexing structure, combining the aforementioned concepts using hash-based indexing, while covering the recent results from Thorup. The work will include research and analysis of possible hashing functions, survey and analysis of trade-offs of existing similar key-value storages, design of the key-value database, construction of experiments to test the system's performance on various workloads and providing a conclusion about the usefulness of the system.

3 Thesis plan

The thesis is planned to consist of the following work

1. Survey existing literature for hashing functions, and analyze their relevance to the design of an efficient hash-based indexing method. [1 week]

2. Survey existing literature for related work in the fields of in-memory multicore key-value storage. Analyze the existing systems' trade-offs such as scalability, easy of application building and lantency for various workloads, and identify how the trade-offs are linked to the system design. [2 weeks]

3. Design a hashtable index for multi-core, in-memory key-value storage that effectively handles arbitrary length keys, including keys with long shared prefixes. [3.5 weeks]

4. Implement the key-value storage system based on the hash-based indexing, and iterate between implementation and re-designing in order to tweak the system for improved performance. [7.5 weeks]

5. Design experiments to test the performance of the developed key-value storage system under various workloads, with focus on YCSB. The results should be used for further iterations of development, in order to tweak the system. Finally, the results must be presented, summarized and properly relating to the key properties to the system. [7 weeks]

As hinted in the two last points, the design, implementation and experimentation phases will be interleaved, in order to build on previous experiences, thus allowing optimization of key components of the system, which might be difficult to reason about theoretically. This includes modification of the system in order to optimize performance on various workloads.

4 Learning Goals

The learning goals for the thesis are the following

1. Review existing literature for existing key-value storage systems, with emphasis on multi-core, in-memory designs. Compare the trade-offs and properties of these systems, and draw conclusions neccesary for designing the target system.

2. Identify state-of-the-art hashing algorithms, and assess how they can be used for the multi-core in-memory hash-based indexing.

3. Analyze and design a hashing algorithm for indexing in the key-value storage.

4. Design the overall key-value storage system based on the hash-based indexing, satisfing the key features in accordance with the thesis question.

5. Design experiments to assess the key aspects of the system and its performance on various workloads, with focus on YCSB.

References

[1] M. Stonebraker, et al. *The end of an architectural era (it's time for a complete rewrite).* In VLDB, pages 1150-1160. ACM, 2007.

[2] M. Stonebraker. *New opportunities for new sql.* Commun. ACM , 55(11):10-11, 2012

[3] Alvarez, V.; Richter, S.; Xiao Chen; Dittrich, J. *A comparison of adaptive radix trees and hash tables* https://infosys.cs.uni-saarland.de/publications/ARCD15.pdf

[4] Mao, Y.; Kohler, E.; Morris, R. *Cache Craftiness for Fast Multicore Key-Value Storage* https://pdos.csail.mit.edu/papers/masstree:eurosys12.pdf

[5] Leis, V.; Kemper, A.; Neumann, T. *The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases* http://www3.informatik.tu-muenchen.de/ leis/papers/ART.pdf

[6] Levandoski, J. Lomet, D; Sengupta, S. *The BW-Tree: A B-tree for New Hardware Platforms* http://www.msr-waypoint.com/pubs/178758/bw-tree-icde2013-final.pdf

[7] Wang, Z.; Qian, H.; Li, J.; Chen, Haibo. *Using Restricted Transactional Memory to Build a Scalable In-Memory Database* http://ipads.se.sjtu.edu.cn/_media/publications/dbx.pdf

[8] Richter, S.; Alvarez, V.; Dittrich, J. *A Seven-Dimensional Analysis of Hashing Methods and its Implications on Query Processing* http://www.vldb.org/pvldb/vol9/p96-richter.pdf

[9] Thorup, M. *Fast and Powerful Hashing using Tabulation* http://arxiv.org/abs/1505.01523