Lukas M. Maas, Thomas Kissinger, Dirk Habich, Wolfgang Lehner

BUZZARD: A NUMA-Aware In-Memory Indexing System

Erstveröffentlichung in / First published in:

SIGMOD/PODS'13: International Conference on Management of Data, New York 22.-27.06.2013. ACM Digital Library, S. 1285–1286.

DOI: https://doi.org/10.1145/2463676.2465342

SIGMOD/PODS'13: International Conference on Management of Data. New York 2013", S. 1285–1286, ISBN 978-1-4503-2037-5
https://doi.org/10.1145/2463676.2465342

BUZZARD: A NUMA-Aware In-Memory Indexing System

Lukas M. Maas, Thomas Kissinger, Dirk Habich, Wolfgang Lehner
Database Technology Group
Technische Universität Dresden
01062 Dresden, Germany
Lukas_Michael.Maas@tu-dresden.de, {firstname.lastname}@tu-dresden.de

ABSTRACT

With the availability of large main memory capacities, in-memory index structures have become an important component of modern data management platforms. Current research even suggests index-based query processing [2] as an alternative or supplement for traditional tuple-at-a-time processing models. However, while simple sequential scan operations can fully exploit the high bandwidth provided by main memory, indexes are mainly latency bound and spend most of their time waiting for memory accesses.

Considering current hardware trends, the problem of high memory latency is further exacerbated as modern shared-memory multiprocessors with non-uniform memory access (NUMA) become increasingly common. On those NUMA platforms, the execution time of index operations is dominated by memory access latency that increases dramatically when accessing memory on remote sockets. Therefore, good index performance can only be achieved through careful optimization of the index structure to the given topology.

BUZZARD is a NUMA-aware in-memory indexing system. Using adaptive data partitioning techniques, *BUZZARD* distributes a prefix-tree-based [1] index across the NUMA system and hands off incoming requests to worker threads located on each partition's respective NUMA node. This approach reduces the number of remote memory accesses to a minimum and improves cache utilization. In addition, all indexes inside *BUZZARD* are only accessed by their respective owner, eliminating the need for synchronization primitives like compare-and-swap.

Figure 1 shows how *BUZZARD* distributes requests across NUMA nodes: (1) For each incoming request, *BUZZARD* uses an adaptive partition table to determine the partition the requested data belongs to. (2) Once the respective partition has been identified, the request is inserted into a thread-local intermediate request buffer. (3) Periodically, these intermediate buffers are atomically flushed to the partitions' main request buffers, reducing contention in the main request buffers through batch inserts and hence achieving a higher buffer throughput. (4) Worker threads (one per partition) located on the different NUMA nodes extract requests from the buffer of their respective partition and handle these requests by using a private index held in local main-memory. *BUZZARD* processes multiple requests as a batch, effectively hiding memory latency by interleaving multiple index lookups. If the workload changes, *BUZZARD* autonomously rebalances partitions to achieve an even load distribution.

First benchmarking results show that NUMA-aware index partitioning has a huge impact on the throughput of in-memory indexing systems running on NUMA hardware. Especially on machines with complex topologies and remote memory accesses that require multiple hops, resulting in further increased memory latency, *BUZZARD* speeds up index performance up to 220% compared to a single index.

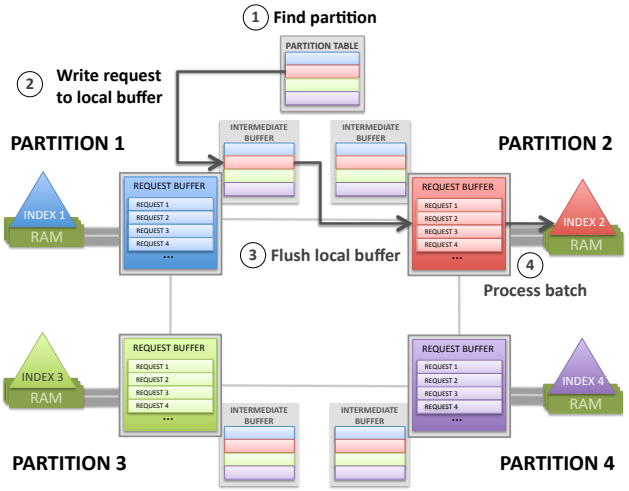

Figure 1: Request processing in *BUZZARD*

Categories and Subject Descriptors

H.2.2 [**Database Management**]: Physical Design

Keywords

NUMA; in-memory indexing; prefix trees

Acknowledgments

This work is supported by the German Research Foundation (DFG) in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

REFERENCES

[1] M. Böhm, B. Schlegel, P. B. Volk, U. Fischer, D. Habich, and W. Lehner. Efficient In-Memory Indexing with Generalized Prefix Trees. In *BTW*, 2011.

[2] T. Kissinger, B. Schlegel, D. Habich, and W. Lehner. QPPT: Query Processing on Prefix Trees. *CIDR*, 2013.