Karlsruher Institut für Technologie

Database Cache Patterns on Modern Processors

Exposé
von

Sebastian Schindler

an der Fakultät für Informatik

Erstgutachter:	Prof. Dr. Frank Bellosa
Betreuender Mitarbeiter:	Matthias Gottschlag

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 30. Mai 2017

Contents

Chapter 1

Introduction

1.1 Current Situation

Database engines have, since their inception in the mid-1960s [7], been heavily optimised to hide hard drive access latencies and keep as much data as possible in main memory, because processors at the time were slow enough that memory access latency did not represent a bottleneck in execution. This has lead to the current compute and memory bound database management systems (DBMSs) [1].

But with the emergence of higher performance processors and multiprocessors, DBMSs can no longer use the computational power effectively, as query time is dominated by off-chip or Level 3 (L3) cache memory access. The work done in [1] shows that memory stalls account for more than half of the total time spent on execution, in [8] Panda et. al. even state that only about 30% of the time is spent on computations in MySQL.

It follows, that good Level 2 (L2) cache locality could lead to a significant speed-up in the software that's available now. But current systems handle multiple queries in thread-parallel, giving up control of execution to the operating system. This makes multiple query threads fight over the available resources and causes an overall slow-down [4].

1.2 Proposed Improvements

In [5], Harizopoulos and Ailamaki created a database which splits up query processing into multiple stages, executing them on different processors. With this technique, they were able to reduce cache miss rates significantly.

But designing a new DBMS which is as stable and efficient as existing ones is difficult and replacing already deployed software can be hard, as there are a vast number of solutions already in use. The easier and more universal approach is using an unmodified DBMS and adding the ability to detect such stages in query processing to the operating system's scheduler which can run each stage on it's own processor. This could possibly be done in a way that works with most database engines without the need to adapt it to every one.

Chapter 2

Related Work

The performance of current databases is analyzed in detail in [1] and [8], which both conclude that query time is most of all hindered by memory latency, because the cache hierarchy is ineffectively utilized in modern DBMSs. As these papers were released 17 years apart, this also shows that there has not been much change in this regard.

[5] and [4] look at breaking up query processing into multiple stages and and show that using this technique and managing concurrent queries to maximize cache utilization can increase the throughput of existing DBMSs by up to 400%.

Chapter 3

Scope

The aim of this work is to find access and miss patterns in cache utilization and with this data, propose points in execution where it would be beneficial to continue on a different processor.

The end goal is to build a scheduler around this data to increase the L2 cache locality. But as this lies beyond the scope of this effort and will be done after it's completed, further analysis on potential speedup or efficiency gains can not be done.

Chapter 4

Methodology and Software

For this, a single DBMS will be used. The system of choice is MariaDB [6], since it's open-source and can be recompiled with debug symbols. From a database benchmark, for example TPC-C [9], some typical queries will be selected to run on the database to test various patterns in executing them. This way, we can analyze the behavior for running multiple different or similar queries concurrently or sequentially.

To analyse the memory behavior, these queries will be executed using either gem5 [3] or Cachegrind [2] to record the data.

Hopefully, it will be possible to find periods that are compute-intensive with good L2 cache locality, resulting in less misses, and periods where the cache seems to be filled with useless data. Switching processors should occur after a compute-intensive phase, so that the data can be reused for other queries.

Bibliography

[1] Anastassia Ailamaki, David J Dewitt, Mark D. Hill, and David A. Wood. DBMSs On A Modern Processor : Where Does Time Go ? *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, 1394:266–277, 1999. `http://dl.acm.org/citation.cfm?id=671662{&}CFID=763942496{&}CFTOKEN=75761375`.

[2] Cachegind. *Cache Profiler*. `http://valgrind.org/info/tools.html#cachegrind`.

[3] The gem5 Simulator. *A modular platform for computer-system architecture research*. `http://gem5.org/Main_Page`.

[4] Stavros Harizopoulos and Anastassia Ailamaki. A Case for Staged Database Systems. In *Proceedings of 1st Conference on Innovative Data Systems Research*, 2003. `http://nms.csail.mit.edu/~stavros/pubs/staged.pdf`.

[5] Stavros Harizopoulos and Anastassia Ailamaki. StagedDB: Designing Database Servers for Modern Hardware. *In IEEE Data*, pages 11–16, 2005. `http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/StagedDB/papers/ieee2005.pdf`.

[6] MariaDB.org. *Ensuring continuity and open collaboration*. `https://mariadb.org`.

[7] M. Lynne Neufeld and Martha Cornog. Database History: From Dinosaurs to Compact Discs. *Journal of the American Society for Information Science*, 37.4:183, 1986. `http://search.proquest.com/openview/c27e8bc50f1f4f78babdb6a383035062/1?pq-origsite=gscholar{&}cbl=1818555`.

[8] Reena Panda, Christopher Erb, Michael Lebeane, Jee Ho Ryoo, and Lizy Kurian John. Performance characterization of modern databases on

out-of-order CPUs. *Proceedings - Symposium on Computer Architecture and High Performance Computing*, 2016-January:114–121, 2016.

[9] TPC. *TPC-C on-line transaction processing benchmark.* `http://www.tpc.org/tpcc/default.asp`.