Balancing vectorized query execution with bandwidth-optimized storage
Zukowski, M

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1

Introduction

The continuous evolution of computer hardware in the past decades has resulted in a rapid increase of available computing power. However, not all application areas benefited from this improvement to the same extent – most new hardware features are targeted at computation-intensive tasks, including computer games, multimedia applications and scientific computing. On the other hand, general-purpose database systems have been shown to have problems with fully exploiting today’s hardware performance potential [ADHW99].

Over the last two decades, CPUs evolved from relatively simple, single-pipeline in-order devices that were easy to program into highly complex elements. These new processors introduce technologies like superscalar out-of-order execution, SIMD instructions and multiple cores. To achieve optimal performance on such hardware, the application code needs to follow new hardware-conscious patterns and be amenable to compiler optimizations. Furthermore, the increase of CPU frequencies resulted in an increasing imbalance between the processor speed and memory latency. As a result, computers depend more and more on multi-level cache memories that improve the memory access time, but, again, often require the programmer to tune data access patterns in the program.

In disk storage two trends can be observed that introduce new challenges for system designers. First, random disk access latency improves significantly more slowly than sequential disk bandwidth. Secondly, both disk latency and bandwidth improve more slowly than the computing power of modern processors, especially with the advent of multi-core CPUs.
1.1 Problem statement

Database engines have been shown to adapt poorly to the hardware developments presented above, in both query processing and storage layers.

Query execution. In this layer, many database systems continue to follow the tuple-at-a-time pipelined model working with N-ary tuples. This makes the CPU spend most time not on the actual data processing, but on traversing the query operator tree. Such program behavior causes problems for modern processors, since it can lead to poor instruction-cache performance and frequent branch mispredictions, significantly reducing the performance. Even worse, the tuple-at-a-time execution model makes it impossible for compilers to apply many performance-critical optimization techniques such as loop-unrolling and SIMDization. This is in contrast with other application areas, such as scientific computing, where data-intensive approaches, spending most time on the actual data processing, can be optimized into highly efficient programs. Some of the database performance problems can be partially solved with techniques that have been published in the area of architecture-conscious query processing. However, most of the previous work in this field concentrates on improving isolated problems within an existing execution framework, often limiting the achieved performance gains to single operations.

An alternative approach to query execution has been presented in the MonetDB system. Here, instead of working on single tuples, the system uses column-at-a-time materializing operators, internally working as simple operations on arrays of values. This results in bulk processing, improving performance by removing the per-tuple interpretation overhead and exposing multiple compiler optimization opportunities. However, the full materialization implied by this model often results in large intermediate results. This causes extra memory or disk traffic and degrades the performance of this system when working with large data volumes. Also, a processing unit of a full column is often too large to apply some of the existing optimization techniques such as memory prefetching. Finally, the column algebra used in this model makes it hard to implement multi-attribute operations, resulting in extra processing steps.

Storage. Also in this layer database systems do not fully adapt to the changing hardware properties. While the relative performance of random I/O with respect to sequential I/O gets worse, many database systems still often rely on random-access methods, such as unclustered indices, even in data-intensive operations. To keep this method efficient, database systems use storage facilities that contain more and more disks to provide enough throughput of random-
access operations. This approach is unsustainable in the long run, but it is not always clear how scan-based strategies could replace random-access ones. Additionally, while random-access methods easily scale to handle heavy query loads by using RAID systems and request batching, scalability of scan-based approaches requires more investigation.

Another problem is that with computing power increasing at a faster pace than disk performance, data delivery becomes a bottleneck even with sequential-access approaches. This is especially visible in systems using the \textit{N-ary storage model}, where entire tuples need to be fetched from disk, even if only a small subset of attributes is actually used. An alternative to this model are \textit{column stores}, which only read relevant attributes, requiring lower disk bandwidth. In both storage models disk performance can also be improved with \textit{data compression}. Here, it is crucial that the decompression is highly efficient, so it does not dominate the actual data processing. Yet another challenge is making a system efficiently support multiple concurrent users. In such scenarios, the performance of current systems often degrades due to queries competing for resources, instead of benefiting from the potential of performing the common tasks once for many users.

\section{Research direction}

The above analysis leads to the general research question addressed in this thesis:

\begin{quote}
How can various architecture-conscious optimization techniques be combined to construct a coherent database architecture that efficiently exploits the performance of modern hardware, for both in-memory and disk-based data-intensive problems?
\end{quote}

As the stated research question is very general, the research track presented in this thesis originally focused on improvements to the MonetDB query execution layer. In the author’s master’s thesis on parallel query execution \cite{Zuk02}, the fully materializing approach was identified as a significant performance and scalability problem, and a more iterative approach was proposed, still working within the MonetDB framework.

This idea evolved into a completely new \textit{vectorized in-cache execution} approach that became the core of the MonetDB/X100 system \cite{BZN05, ZBNH05, Znk05a}. This new approach extends the pipelined model by making the operators work on a set of tuples, represented by \textit{vectors}, each consisting of hundreds
or thousands of values of a single attribute. The execution is divided into generic operator logic and specialized, highly efficient data processing primitives similar to the MonetDB operators. This allows the system to achieve the high performance that bulk-processing delivers, without sacrificing system scalability.

After obtaining very high in-memory performance results on the 100GB TPC-H benchmark it became clear that the high processing bandwidth of the query execution layer (reaching over one gigabyte per second on a single CPU core) is hard to match with typical disk systems. This resulted in shifting the focus of this research to the storage layer, with the goal of researching new disk storage techniques and disk access strategies able to satisfy these high requirements [Zuk05b]. Consecutively, this led to the development of a number of methods that improve data delivery performance for scan-based applications, resulting in a ColumnBM storage system.

The techniques proposed in this thesis have been evaluated in two application areas: data warehousing and decision support, represented by the TPC-H benchmark [Tra06], and large-scale information retrieval, represented by the Terabyte TREC benchmark [CSS].

1.3 Research questions

The research directions presented above reflect the following set of the underlying research questions:

1. Is it possible to combine the benefits of the tuple-at-a-time model and bulk-processing in a coherent query execution model?

2. What techniques allow database engines to rely more on sequential scans instead of on random I/O?

3. What techniques can improve database I/O performance for individual queries?

4. What techniques can improve database I/O performance for heavy query loads?

This thesis tries to provide answers to these questions. However, it does not look at proposed improvements in isolation, but rather investigates how different optimizations, both new and existing ones, can cooperate within a coherent database architecture. Additionally, it has a goal of making the proposed improvements readily applicable to database systems.
1.4 Research results and thesis outline

The research presented in this book leads to the following thesis statement:

> With the vectorized execution model database systems can minimize the instructions-per-tuple cost on modern CPUs and achieve high in-memory performance, but **bandwidth-optimizing improvements** in the storage layer are required to scale this performance to disk-based datasets.

This thesis statement is supported with the following scientific contributions:

**Vectorized in-cache execution model.** Addresses research question 1, parts published in CIDR’05 [BZN05], DAMON’06 [ZHB06] and DAMON’08 [ZNB08], discussed in Chapters 4-5.

The thesis proposes a new execution model that combines the best properties of the previously applied approaches. Benchmarks have demonstrated that it brings numerous performance benefits, including reduced interpretation overhead and multiple performance optimization opportunities. However, the strict separation of the relational operator logic and the actual data processing, which is the key feature of this model, makes it hard to provide fully vectorized relational operator implementations. This thesis proposes various methods of tackling this problem, presenting how typical processing tasks can be efficiently vectorized. It also introduces new hardware-conscious techniques, for example improved hash-based processing. The resulting execution engine efficiently exploits modern CPUs and cache-memory systems and achieves in-memory performance often one or two orders of magnitude higher than the existing approaches.

**Bandwidth-optimizing disk access model.** Addresses research question 2, parts published in CIDR’05 [BZN05], BNCOD’05 PhD Workshop [Zuk05b] and VLDB’07 [ZNB07], discussed in Chapter 4.

With the high performance of the vectorized execution kernel, it becomes hard to provide sufficient data delivery bandwidth from disk. With the increasing imbalance between disk latency and bandwidth, strategies based on random disk access are infeasible for most applications processing large data volumes. This thesis discusses a number of approaches that avoid random accesses and allow a scan-mostly query execution model. Additionally, even with scan-based approaches, it is crucial to optimize the use of available disk bandwidth. The DSM storage model, improved with **lightweight compression**, can reduce data volumes that need to be transferred. Additionally, intelligent data sharing be-
tween queries minimize the number of times the same data needs to be fetched from disk.

**Ultra-lightweight data compression.**¹ Addresses research question 3, published in ICDE’06 [ZHNB06], discussed in Chapter 6. This thesis introduces a set of compression algorithms that allow trading some CPU power for an increased perceived disk bandwidth. This approach is especially useful in column-stores, as the contiguously stored data from the same domain offers good compression opportunities, and only the used columns need to be decompressed. The proposed algorithms are carefully tuned for modern CPUs, achieving decompression bandwidth in the order of gigabytes per second, which is one or two orders of magnitude higher than popular compression solutions. Moreover, they are optimized for the vectorized in-cache execution pipeline: data is decompressed on a vector granularity, and it is materialized only in the CPU cache, from where it is immediately consumed for processing. These two techniques make the decompression overhead minimal, leaving enough CPU time to process the decompressed data. As a result the query performance for disk-based datasets is significantly improved.

**Cooperative scans.** Addresses research question 4, published in VLDB’07 [ZHNB07], discussed in Chapter 7. Since sequential data access is the preferred access method for data intensive workloads, it is important to optimize scenarios with multiple concurrent queries performing scans at the same time. The introduced “cooperative scans” technique extends the traditional buffer manager by dynamically managing query activity, buffer content and I/O operations to maximize data sharing between queries and minimize disk activity. It outperforms existing shared scans methods both in query latency and system throughput, as demonstrated for various scenarios on PAX and DSM datasets.

¹ Joint work with Sándor Héman, parts of this research might appear in his PhD thesis.