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 8

Conclusions

The research presented in this thesis, summarized in Section 8.1, introduces a number of techniques that improve the query execution efficiency in databases by taking advantage of modern hardware trends. The resulting MonetDB/X100 system, used to validate our ideas, achieves high performance in database processing, as demonstrated with the TPC-H benchmarks presented throughout this thesis and summarized in Section 8.2.1. Additionally, thanks to its efficiency, it makes it possible for database technology to be successfully applied in other areas, as discussed in Section 8.2.2. Finally, many ideas presented in this thesis lead to interesting research problems, discussed in Section 8.3.

8.1 Contributions

This thesis introduces a number of techniques in the field of data-intensive query processing, focusing on in-memory query execution and processing of disk-resident data. The proposed methods are presented in the context of a balanced database architecture.

8.1.1 Improving in-memory query processing

The major contribution in this area is the vectorized execution model introduced in Section 1.2. This model combines the best features of the previously proposed tuple-at-a-time and column-at-a-time models, taking the scalability from the former, and the raw processing performance from the latter. This high
performance is achieved by spending most of the CPU time in data processing
primitives designed and implemented to execute efficiently on modern super-
scalar CPUs. Additionally, in-cache execution allows good use of hierarchical
memory structures. As a result, the proposed model has been shown to pro-
vide performance often one or two orders of magnitude better than the existing
solutions.

Research on the efficient implementation of the vectorized execution model,
presented in Chapter 5 resulted in a number of algorithms and implementation
methods beneficial to a wide range of applications. First, a number of algo-
rithm design and implementation techniques for the vectorized model has been
presented. These techniques demonstrate how to decompose algorithms into a
sequence of vectorized steps and how to efficiently implement primitives per-
forming these steps. Additionally, we discuss how different data organization
models have different performance characteristics for different processing tasks,
and introduce a novel idea of dynamic data reorganization inside the query
processing pipeline to better exploit these properties. These techniques are syn-
thesized in a fully vectorized implementation of a generic hash-join, which is
demonstrated to achieve performance comparable with a hand-written, special-
ized implementation. A proposed technique of cache-efficient best-effort parti-
tioning additionally allows applying algorithms using hash-tables to data sizes
exceeding the capacity of the CPU cache. Finally, we propose new implemen-
tations of a set of important, but less-frequently researched, data processing
algorithms, such as exception handling, string processing and binary search,
showing how the vectorized versions can provide significant performance im-
provement over traditional approaches.

8.1.2 Improving processing of disk-resident data

For disk-resident data, this thesis proposes to remove the need for random disk
accesses by following the idea of \textit{scan-only} query processing. In this way large
volumes of data can be efficiently provided to the query execution layer. To
reduce the impact of the increasing imbalance between the CPU and disk per-
formance, this thesis additionally introduces two techniques that allow better
exploitation of the available disk bandwidth.

The first technique follows the idea of using data compression to reduce the
volume of data that needs to be fetched from disk. This thesis introduces a set of
new compression algorithms that are focused on efficiently exploiting the prop-
erties of modern CPUs to achieve high compression and decompression speeds.
Additionally, we propose keeping the data compressed in the buffer manager and
decompressing it on the boundary between RAM and the CPU cache. This technique allows caching more data, avoids in-memory materialization, and can even improve the performance of the memory bandwidth on multi-core machines.

The second major contribution is the research on improving the performance of concurrently running scan queries. Within the general framework of cooperative scans we investigate the properties of various scheduling algorithms in databases. We also introduce a new relevance policy that dynamically schedules I/O requests by analyzing the entire system activity and, thanks to dynamic adaptation to changing conditions, achieves significant performance improvements for a large class of queries. Finally, this work discusses why efficient scanning is more complex in the DSM world, with queries scanning not only a subset of rows but also a subset of columns.

8.1.3 Balanced database system architecture

Performance benefits of the vectorized execution model can only be observed if the data delivery layer can match its processing speed. Similarly, improvements to the I/O infrastructure will not be visible if the performance of the processing layer is low and the system is not I/O bound.

The design of the MonetDB/X100 architecture is based around the idea of different optimizations cooperating to achieve high performance. An example of this approach is the compression layer: it decompresses data per-vector saving it into the CPU cache, perfectly matching the vectorized execution model.

The importance of balancing both processing and storage optimizations is presented with an experiment in Figure 8.1. Here, TPC-H Query 6 (1GB scale factor) is executed on a 2.8GHz Intel Nehalem system with an efficient disk subsystem. We compare the performance using different vector sizes, I/O buffer sizes, and two queries: one operating on raw data (ca. 160 MB) and one operating on compressed data (ca. 36 MB). With fully buffered data the overhead of decompression adds a small, but visible overhead (left plot). However, compression provides two benefits. First, since the data volume is reduced, a smaller buffer size is required to have data fully cached (middle plot). Secondly, even if the data needs to be fully read from the disk, the I/O overhead is significantly smaller for the compressed data (right plot). In all cases, small vector sizes cause the execution to be dominated by the interpretation overhead, making the impact of data format and buffer size negligible. This experiment demonstrates that compression can scale vectorized execution to larger data sets, and

1 both queries perform a fully sequential scan, hence do not benefit from the partially buffered data
192 Chapter 8: Conclusions

Figure 8.1: Performance of the TPC-H Query 6 with raw and compressed data using different I/O buffer sizes

at the same time high-performance execution is needed to make the benefit of compression visible.

8.2 Evaluation

The techniques used in the MonetDB/X100 system allowed it to achieve very high efficiency in data-analysis tasks, as demonstrated with the TPC-H results that are summarized in Section 8.2.1. This performance, often close to hand-written solutions, has led to the investigation if the proposed architecture can also be applied in other fields where databases were traditionally not used due to their poor performance. Section 8.2.2 discusses how MonetDB/X100 successfully competed with specialized solutions in one of such areas: large-scale information retrieval.

8.2.1 TPC-H performance

TPC-H is currently the standard benchmark suite for decision support systems [Tra06]. It simulates systems that process large volumes of data and execute highly complex queries to answer critical business questions. This is a typical example of data-intensive problems. Since research presented in this thesis focuses on improving database performance in such applications, TPC-H has
Table 8.1: TPC-H SF-100 results on MonetDB/X100: memory-resident benchmarks (from [BZN05]), disk-resident benchmarks (from [ZHNB06]). DB2 results for comparison (2006 results from www.tpc.org)

Table 8.1 demonstrates the performance achieved by MonetDB/X100 on a subset of the TPC-H benchmark for both memory- and disk-resident data. It also present the results of the IBM DB2 benchmark running on a significantly more powerful hardware configuration. While these systems are not fully comparable, as MonetDB/X100 at the moment of these benchmarks did not provide all the capabilities required by TPC-H and the query plans were hand-crafted, the results indicate that MonetDB/X100 performs significantly better in terms of the absolute performance. This performance advantage becomes even more visible when taking into account the difference in the available processing and storage power.

### 8.2.2 Information retrieval with MonetDB/X100

Integration of information retrieval and databases is seen as one of the major goals of the database community [AY05, CRW05]. Still, these two areas have de-
developed independently from each other, even though many IR tasks can be performed using relational systems. One of the reasons for not applying database technology to information retrieval is the poor performance of standard database solutions. For example, only one attempt has been made to participate in the Terabyte TREC benchmark using a database system, and the performance of the presented solution was significantly worse than that of specialized IR systems. Since one of the goals of MonetDB/X100 was to bridge the gap between general-purpose database technology and task-specialized applications, we decided to evaluate the performance of our system in this area. This section provides only an overview of the results, more details can be found in [HZdVB06, CHZ+08, HZdVB07].

### 8.2.2.1 Expressing IR tasks as relational queries

For our experiments, we investigated a set of techniques applied in keyword search, including Boolean OR and AND queries as well as the popular Okapi BM25 formula. In these models, data is typically stored in inverted files, where for each term a sorted list of documents with this term is stored, either containing all individual occurrences, or simply a number of occurrences within the document. For this storage model, a keyword search can be expressed as a combination of these lists with some arithmetic on top, typically followed by Top-N computation. This general structure can be applied to both Boolean queries as well as to BM25. Interestingly, the task of combining inverted lists is the exact equivalent of performing a sequence of merge-join operations (inner or outer) on the matching data in a database. This allows easy mapping of these query models onto relational queries, as presented in [CHZ+08, HZdVB06, HZdVB07].

### 8.2.2.2 Performance on Terabyte TREC benchmark

A simple mapping of IR tasks onto MonetDB/X100 queries provides performance already in the same ballpark as specialized systems [HZdVB07]. However, many of the optimization techniques used in IR can also be applied in a relational system. For example, the high-performance data compression algorithms of MonetDB/X100 provide a significant performance benefit when querying disk-resident data, and allow fitting more data in RAM, reducing the number of machines needed to store the entire inverted index in a distributed setting [HZdVB07]. Also, IR techniques such as two-pass querying [BCH+03] and score materialization are easily expressible in a relational system [HZdVB07].
The discussed techniques allow MonetDB/X100 to achieve performance directly comparable with the specialized IR systems, as demonstrated with the Terabyte-TREC results presented in Table 8.2. In this benchmark, a collection of 25 millions documents with a total size of 426 GB is queried using simple keyword-search. System efficiency (speed) is measured by executing 50,000 queries, while effectiveness (accuracy) is evaluated by early precision (p@20) on a subset of 50 preselected queries for which relevance judgments are available. Systems competing in this benchmark are typically specialized IR solutions optimized for executing this type of queries. In our approach, search requests are expressed fully as relational queries and executed on MonetDB/X100 without any specific IR optimizations. The results show that, surprisingly, MonetDB/X100 easily competes with the best participating systems, achieving high efficiency without hurting the effectiveness. Additional general-purpose optimizations, e.g. using PAX storage\(^2\) to reduce random disk access cost, would make this comparison even more favorable for MonetDB/X100.

More detailed experiments presented in [CHZ+08], demonstrate that our approach is also highly flexible. Thanks to its efficient compression, MonetDB/X100 reduces the dataset size from 29GB to 9GB, making it memory-resident on a 12GB machine. This removes the need for I/O and brings the average execution time down to 21ms. Additionally, remote query execution allows easy setup of distributed experiments, where MonetDB/X100 achieves amortized query time of just 3.2ms (using 8 nodes, each with 2GB of RAM).

These results demonstrate that MonetDB/X100 can be successfully applied not only to database tasks, but to a wider class of problems where large volumes of data need to be processed fast. In the future, we hope to investigate other such areas, including scientific data processing and data mining.

\(^2\)PAX storage was not implemented in MonetDB/X100 at the time of running these experiments.
8.3 Future research directions

The material discussed in this thesis provides valuable insights in many areas of query execution on modern hardware. Still, the dynamic nature of computer evolution, as well as the wide scope of discussed areas, lead to a number interesting problems for future research.

8.3.1 Improving the vectorized execution model

In this thesis the vectorized execution model has been shown to have good performance characteristics. Still, multiple improvements are possible.

One of the crucial parameters of the vectorized execution model is the vector size, discussed in Section 4.2.2.2. Currently, it is fixed for the entire query. However, with blocking operators inside the query tree, the query plan is effectively decomposed into semi-independent processing stages with potentially different characteristics. In such a scenario, the vector sizes can be different for various operators, to better balance the available cache size and query complexity.

In Section 5.2.2 we discussed how the storage model during query execution influences performance. While currently only DSM and NSM models were discussed, it is possible to generalize these models to a setup where different attributes are clustered into multiple groups, each represented as DSM (single-attribute groups) or NSM (multi-column groups), as discussed in [HP03]. In some situations, this can provide performance improvements. Furthermore, the research presented in [ZNB08] demonstrates that in-query on-the-fly data conversion can provide extra performance benefits. However, such a flexible data organization introduces an additional dimension to the query optimizer, especially with dynamic data reorganization. Also, efficient storage of variable-width data types, as well as efficient handling of selection (and other types of tuple-indirection) in this model need to be researched.

8.3.2 Storage-layer improvements

Light-weight compression methods discussed in Chapter 6 can significantly reduce the I/O bandwidth hunger. Still, the presented set of algorithms is relatively limited, as it does not allow efficient compression and fast decompression of many data sets. For example, similarly performing algorithms for strings or floating-point numbers need to be researched. Additionally, with the increased number of possible compression schemes (and their parameters), more efficient and flexible automatic compression algorithms need to be investigated.
The cooperative-scans technique presented in Chapter 7 currently focuses on optimizing single-table scans, as found in a typical star schema. Extending this concept to a multi-table scenario, especially in a situation when individual queries scan multiple tables, is an interesting challenge. Similarly, applying this technique to complex merge-join scenarios poses a problem. Finally, with the continuously increasing imbalance between memory and cache speeds, it is possible that some of the discussed concepts can be applied to coordinate memory accesses by concurrently running queries.

Another research direction is related to the rapidly increasing popularity of solid-state storage (see Section 2.3.3). Since this form of storage has significantly different parameters, new algorithms to access the data need to be devised [Ros08, SHWG08].

The final topic currently being researched in the storage layer of MonetDB/X100 are efficient updates for analytical applications. Since data organization for such scenarios (clustering, DSM storage, compression) is often different than for transaction processing, new techniques that allow efficient updating of such structures need to be investigated.

8.3.3 Parallel execution

The vectorized execution model provides interesting opportunities for parallel execution. With parallelism implemented e.g. with exchange operators [Gra90], the larger size of the data transfer unit leads to less overhead of the inter-process communication. Additionally, vertical fragmentation minimizes the volume of RAM-to-CPU data transfer, reducing the problem of insufficient memory bandwidth on multi-core machines.

A large processing unit in vectorized processing leads to the idea of parallelizing on the level of a single data-processing primitive, similarly to the horizontal parallelism for relational operators. With large L2 and L3 caches, holding tens of thousands of values, the overhead of synchronization might be amortized well and this approach might provide an efficient solution with relatively limited system changes.

8.3.4 Alternative hardware platforms

Previous research [HNBZ07] demonstrated that the vectorized execution model is a good foundation for database processing kernels on the hybrid STI Cell processor. Interestingly, a lot of emerging computing architectures share some characteristics with Cell. SIMD-like processing units constitute a consistently
growing fraction of the entire system computing power. For example, for many tasks modern GPUs easily provide better performance than price-comparable general purpose CPUs. Multiple processing cores already are a mainstream solution, and the number of cores in future generations of CPUs will probably grow [HBK06]. In some architectures, e.g. GPUs, parallelism is an inherent property of the computing units. Also, the heterogeneity of the computing units is increasing, with systems consisting of multiple types of cooperating devices (CPUs, GPUs etc), potentially also internally heterogeneous (e.g. Cell). Finally, the hard restrictions on code and data size, as well as explicit management of both, appear in many new architectures. We believe that the vectorized model provides multiple opportunities to address these issues, and can be a good foundation for developing data processing kernels on these new architectures.