View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by CWI's Institutional Repository

The Database Architectures Research Group at CWI

Martin Kersten Stefan Manegold Sjoerd Mullender
CWI
Amsterdam, The Netherlands
first.last@cwi.nl
http://www.cwi.nl/research-groups/Database-Architectures/

1. INTRODUCTION

The Database research group at CWI was established in 1985. It has steadily grown from two PhD students to a group of 17 people ultimo 2011. The group is supported by a scientific programmer and a system engineer to keep our machines running. In this short note, we look back at our past and highlight the multitude of topics being addressed.

2. THE MONETDB ANTHOLOGY

The workhorse and focal point for our research is *MonetDB*, an open-source columnar database system. Its development goes back as far as the early eighties when our first relational kernel, called *Troll*, was shipped as an open-source product. It spread to ca. 1000 sites world-wide and became part of a software case-tool until the beginning of the nineties. None of the code of this system has survived, but ideas and experiences on how to obtain a fast relational kernel by simplification and explicit materialization found their origin during this period.

The second half of the eighties was spent on building the first distributed main-memory database system in the context of the national *Prisma* project. A fully functional system of 100 processors and a, for that time, wealthy 1 GB of main memory showed the road to develop database technology from a different perspective. Design from the processor to the slow disk, rather than the other way around.

Immediately after the Prisma project, a new kernel based on Binary Association Tables (*BATs*) was laid out. This storage engine became accessible through *MIL*, a scripting language intended as a target for compiling SQL queries. The target application domain was to better support scientific databases with their (archaic) file structures. It quickly shifted to a more urgent and emerging area. Several datamining projects called for better database support. It culminated in our first spin-off company, Data Distilleries, in 1995, which based their analytical customer relationship suite on the power provided by the early MonetDB implementations. In the years following, many technical innovations were paired with strong industrial maturing of the software base. Data Distilleries became a subsidiary of SPSS in 2003, which in turn was acquired by IBM in 2009.

Moving MonetDB Version 4 into the open-source domain required a large number of extensions to the code base. It became of the utmost importance to support a mature implementation of the SQL-03 standard, and the bulk of application programming interfaces (PHP, JDBC, Python, Perl, ODBC, RoR). The result of this activity was the first official open-source release in 2004. A very strong XQuery front-end was developed with partners and released in 2005 [1].

MonetDB remains a product well-supported by the group. All its members carry out part of the development and maintenance work, handling user inquiries, or act as guinea pigs of the newly added features. A thorough daily regression testing infrastructure ensures that changes applied to the code base survive an attack of ca. 20 platform configurations, including several Linux flavors, Windows, FreeBSD, Solaris, and MacOS X. A monthly bugfix release and ca. 3 feature releases per year support our ever growing user community. The web portal[1] provides access to this treasure chest of modern database technology. It all helped us to create and maintain a stable platform for innovative research directions, as summarized below. The MonetDB spin-off company was set up to support its market take-up, to provide a foundation for QA, support, and development activities that are hard to justify in a research institute on an ongoing basis.

3. HARDWARE-CONSCIOUS DATABASE TECHNOLOGY

A key innovation in the MonetDB code base is its reliance on hardware conscious algorithms. For,

[1] `http://www.monetdb.org/`

advances in speed of commodity CPUs have far outpaced advances in RAM latency. Main-memory access has therefore become a performance bottleneck for many computer applications, including database management systems; a phenomenon widely known as the "memory wall." A revolutionary redesign of database architecture was called for in order to take advantage of modern hardware, and in particular to avoid hitting this memory wall. Our pioneering research on columnar and hardware-aware database technology, as materialized in MonetDB, is widely recognized, as indicated by the VLDB 2009 10-year Best Paper Award [19, 2] and two DaMoN best paper awards [22, 6]. Here, we briefly highlight important milestones.

Vertical Storage. Whereas traditionally, relational database systems store data in a row-wise fashion (which favors single record lookups), MonetDB uses a columnar storage, which favors analysis queries by better using CPU cache lines.

Bulk Query Algebra. Much like the CISC vs. RISC idea applied to CPU design, the MonetDB relational algebra is deliberately simplified compared to the traditional relational set algebra. Paired with an operator-at-a-time bulk execution model, rather than the traditional tuple-at-a-time pipelining model, this allows for much faster implementation on modern hardware, as the code requires far fewer function calls and conditional branches.

Cache-conscious Algorithms. The crucial aspect to overcome the memory wall is good use of CPU caches, for which careful tuning of memory access patterns is needed. This led to a new breed of query processing algorithms. Their key requirement is to restrict any random data access pattern to data regions that fit into the CPU caches to avoid cache misses, and thus, performance degradation. For instance, *partitioned hash-join* [2] first partitions both relations into H separate clusters that each fit into the CPU caches. The join is then performed per pair of matching clusters, building and probing the hash-table on the inner relation entirely inside the CPU cache. With large relations and small CPU caches, efficiently creating a large number of clusters can become a problem in itself. If H exceeds the number of TLB entries or cache lines, each memory reference will trigger a TLB or cache miss, compromising the performance significantly. With *radix-cluster* [17], we prevent that problem by performing the clustering in multiple passes, such that each pass creates at most as many new sub-clusters as there are TLB entries or cache lines. With *radix-decluster* [18], we complement partitioned hash-join with a projection (tuple reconstruction) algorithm with a cache-friendly data access pattern.

Memory Access Cost Modeling. For query optimization to work in a cache-conscious environment, and to enable automatic tuning of our cache-conscious algorithms on different types of hardware, we developed a methodology for creating cost models that takes the cost of memory access into account [16]. The key idea is to abstract data structures as *data regions* and model the complex data access patterns of database algorithms in terms of simple compounds of a *few basic data access patterns*. We developed cost functions to estimate the cache misses for each basic pattern, and rules to combine basic cost functions and derive the cost functions of arbitrarily complex patterns. The total cost is then the number of cache misses multiplied by their latency. In order to work on diverse computer architectures, these models are parametrized at run-time using automatic *calibration* techniques.

Vectorized Execution. In the "X100" project, we explored a compromise between classical tuple-at-a-time pipelining and operator-at-a-time bulk processing [3]. The idea of vectorized execution is to operate on chunks (vectors) of data that are large enough to amortize function call overheads, but small enough to fit in CPU caches to avoid materialization into main memory. Combined with just-in-time light-weight compression, it lowers the memory wall somewhat. The X100 project has been commercialized into the Actian/VectorWise company and product line [2].

4. DISTRIBUTED PROCESSING

After more than a decade of rest at the frontier of distributed database processing, we embarked upon several innovative projects in this area again.

Armada. An adventurous project was called Armada where we searched for technology to create a fully autonomous and self regulating distributed database system [5]. The research hypothesis was to organize a large collection of database instances around a dynamically partitioned database. Each time an instance ran out of resources, it could solicit a spare machine and decide autonomously on what portion to delegate to its peer. The decisions were reflected in the SQL catalog which triggered continuous adaptive query modification to hunt after the portions in the loosely connected network of workers. It never matured as part of the MonetDB distribution, because at that time we did not have all the basic tools to let it fly.

Since, the Merovingian toolkit developed and now provides the basis for massive distributed process-

[2] `http://www.actian.com/products/vectorwise/`

ing. It provides server administration, server discovery features, client proxying and funneling to accommodate large numbers of (web) clients, basic distributed (multiplex) query processing, and failover functionality for a large number of MonetDB servers in a network. It is the toolkit used by partner companies to build distributed datawarehouse solutions. With Merovingian we were able to open two new research tracks: DataCyclotron and Octopus. Our new machine cluster [3] provides a basis to explore both routes in depth.

DataCyclotron. The grand challenge of distributed query processing is to devise a self-organizing architecture which exploits all hardware resources optimally to manage the database hot-set, to minimize query response time, and to maximize throughput without single point global coordination. The Data Cyclotron architecture [4] addresses this challenge using turbulent data movement through a storage ring built from distributed main memory and capitalizing on the functionality offered by modern remote-DMA network facilities. Queries assigned to individual nodes interact with the storage ring by picking up data fragments that are continuously flowing around, i.e., the hot-set.

The storage ring is steered by the *level of interest* (*LOI*) attached to each data fragment. The LOI represents the cumulative query interest as it passes around the ring multiple times. A fragment with *LOI* below a given threshold, inversely proportional to the ring load, is pulled out to free up resources. This threshold is dynamically adjusted in a fully distributed manner based on ring characteristics and locally observed query behavior. It optimizes resource utilization by keeping the average data access latency low. The approach is illustrated using an extensive and validated simulation study. The results underpin the fragment hot-set management robustness in turbulent workload scenarios.

A fully functional prototype of the proposed architecture has been implemented using modest extensions to MonetDB and runs within a multi-rack cluster equipped with Infiniband. Extensive experimentation using both micro benchmarks and high-volume workloads based on TPC-H demonstrates its feasibility. The Data Cyclotron architecture and experiments open a new vista for modern in-the-network distributed database architectures with a plethora of research challenges.

Octopus. In the Octopus project, we deviate from the predominant approach in distributed database processing, where the data is spread across a number of machines one way or another before any query processing can take place. We start from a single master node in control of the database, and with a variable number of worker nodes to be used for delegated query processing. Data is shipped just-in-time to the worker nodes using a need-to-know policy, and reused, if possible, in subsequent queries. A bidding mechanism among the workers yields the most efficient reuse of parts of the original data, available on the workers from previous queries.

The adaptive distributed architecture uses the master/workers paradigm: the master hosts the database and computes a query by generating distributed subqueries for as many workers as it has currently available. The workers recycle the data they have processed in the past as much as possible to minimize the data transfer costs. Due to the just-in-time replication, the system easily harvests non-dedicated computational resources, while supporting full SQL query expressiveness.

Our experiments show that the proposed adaptive distributed architecture is a viable and flexible approach for improving the query performance of a dedicated database server by using non-dedicated worker nodes, reaching benefits comparable to traditional distributed databases.

5. ADAPTIVE INDEXING

Query performance strongly depends on finding an execution plan that touches as few superfluous tuples as possible. The access structures deployed for this purpose, however, are non-discriminative. They assume every subset of the domain being indexed is equally important, and their structures cause a high maintenance overhead during updates. Moreover, while hard in general, the task of finding the optimal set of indices becomes virtually impossible in scenarios with unpredictable workloads.

With **Database Cracking**, we take a completely different approach. Database cracking combines features of automatic index selection and partial indexes. Instead of requiring a priori workload knowledge to build entire indices prior to query processing, it takes each query predicate as a hint how to physically reorganize the data. Continuous physical data reorganization is performed on-the-fly during query processing, integrated in the query operators. When a column is queried by a predicate for the first time, a new cracker index is initialized. As the column is used in the predicates of further queries, the cracker index is refined by range partitioning until sequentially searching a partition is faster than binary searching in the AVL tree guiding a search to the appropriate partition.

[3] `http://www.scilens.org/platform/`

Keys in a cracker index are partitioned into disjoint key ranges, but left unsorted within each partition. Each range query analyzes the cracker index, scans key ranges that fall entirely within the query range, and uses the two end points of the query range to further partition the appropriate two key ranges. Thus, in most cases, each partitioning step creates two new sub-partitions using logic similar to partitioning in quicksort. A range is partitioned into 3 sub-partitions if both end points fall into the same key range. This happens in the first partitioning step in a cracker index (because there is only one key range encompassing all key values) but is unlikely thereafter [7].

Updates and their efficient integration into the data structure are covered in [8]. Multi-column indexes to support selections, tuple reconstructions and general complex queries are covered in [9]. In addition, [9] supports partial materialization and adaptive space management via partial cracking.

While database cracking comes with very low overhead but slow convergence towards a fully optimized index, *adaptive merging* features faster convergence at the expense of a significantly higher overhead. **Hybrid adaptive indexing** aims at achieving a faster convergence while keeping the overhead low as with database cracking [10].

With **stochastic cracking**, we introduce a significantly more resilient approach to adaptive indexing. Stochastic cracking does use each query as advice on how to reorganize data, but not blindly so; it gains in resilience and avoids performance bottlenecks by allowing for lax and arbitrary choices in its decision-making. Thereby, we bring adaptive indexing forward to a mature formulation that confers the workload-robustness previous approaches lacked.

Ongoing work aims at combining adaptive indexing techniques with the ideas of physical design and auto-tuning tools. The goal is to exploit workload knowledge to steer adaptive indexing where possible, but keep the flexibility and instant adaptation to changing workloads of adaptive indexing.

6. SCIENTIFIC DATABASES

After the first open-source release of MonetDB, we were keen to check its behavior on real-life examples beyond the classical benchmarks. The largest, well-documented and publicly available datawarehouse was the Sloan Digital Sky Survey (SDSS) / SkyServer. Embarking on its re-implementation was a challenge. None of the other DBMSs had accomplished a working implementation, either due to its complexity or lack of resources (business drive).

Skyserver. We achieved a fully functional implementation of SkyServer [4]. It proved that the column store approach of MonetDB has a great potential in the world of scientific databases. However, the application also challenged the functionality of our implementation and revealed that a fully operational SQL environment is needed, e.g., including persistent stored modules. Its initial performance was competitive to the reference platform, Microsoft SQL Server 2005, and the analysis of SDSS query traces hinted at several techniques to boost performance by utilizing repetitive behavior and zoom-in/zoom-out access patterns that were not captured by the system.

Recycler. An immediate follow up project focused on developing a *recycler* component to MonetDB. It acts as an intelligent cache of all intermediate results. Avoiding recomputing of any subquery as often as possible, within the confines of the storage set aside for the intermediates. The results were published in 2009 at SIGMOD and received the runner up best paper award [11].

Recycling can be considered an adaptive materialized view scheme. Any subquery can be re-used, there is no a priori decision needed by a human DBA. It is also more effective than recycling only the final query result sets. Integration of the recycler with the SDSS application showed that a few materialized views had been forgotten in the original design, which would have improved throughput significantly. This was found without human intervention.

SciBORQ. Scientific discovery has shifted from being an exercise of theory and computation, to become the exploration of an ocean of observational data. This transformation was identified by Jim Gray as the 4th paradigm of scientific discovery. State-of-the-art observatories, digital sensors, and modern scientific instruments produce Petabytes of information every day. This scientific data is stored in massive data centers for later analysis. But even from the data management viewpoint, the capture, curating, and analysis of data is not a computation intensive process any more, but a data intensive one. The explosion in the amount of scientific data presents a new "stress test" for database design. Meanwhile, the scientists are confronted with new questions, how can relevant and compact information be found from such a flood of data?

Data warehouses underlying Virtual Observatories stress the capabilities of database management systems in many ways. They are filled on a daily basis with gigabytes of factual information, derived from large data scrubbing and computational in-

[4] see `http://www.scilens.org/`

tensive feature extraction pipelines. The predominant data processing techniques focus on massive parallel loads and map-reduce algorithms. Such a brute force approach, albeit effective in many cases, is costly.

In the SciBORQ project, we explore a different route [21]. One based on the knowledge that only a small fraction of the data is of real value for any specific task. This fraction becomes the focus of scientific reflection through an iterative process of adhoc query refinement. However, querying a multiterabyte database requires a sizable computing cluster, while ideally the initial investigation should run on the scientist's laptop.

We work on strategies on how to make biased *snapshots* of a science warehouse such that data exploration can be instigated using precise control over all resources. These snapshots, constructed with novel sampling techniques, are called *impressions*. An impression is selected such that either the statistical error of a query answer remains low, or an answer can be produced within strict time bounds. Impressions differ from previous sampling approaches because of their *bias* towards the focal point of the scientist's data exploration.

7. STREAMING

DataCell. Streaming applications have been en vogue for over a decade now and continuous query processing has emerged as a promising paradigm with numerous applications. A more recent development is the need to handle both streaming queries and typical one-time queries in the same application setting, e.g., complex event processing (CEP). For example, data warehousing can greatly benefit from the integration of stream semantics, i.e., online analysis of incoming data and combination with existing data. This is especially useful to provide low latency in data intensive analysis in big data warehouses that are augmented with new data on a daily basis.

However, state-of-the-art database technology cannot handle streams efficiently due to their "continuous" nature. At the same time, state-of-the-art stream technology is purely focused on stream applications. The research efforts are mostly geared towards the creation of specialized stream management systems built with a different philosophy than a DBMS. The drawback of this approach is the limited opportunities to exploit successful past data processing technology, e.g., query optimization techniques.

For this new problem we combine the best of both worlds. In the DataCell project [14] we take a different route by designing a stream engine on top of an existing relational database kernel [15]. This includes reuse of both its storage/execution engine and its optimizer infrastructure. The major challenge then becomes the efficient support for specialized stream features.

We focus on incremental window-based processing, arguably the most crucial stream-specific requirement. In order to maintain and reuse the generic storage and execution model of the DBMS, we elevate the problem to the query plan level. Proper optimizer rules, scheduling and intermediate result caching and reuse, allow us to modify the DBMS query plans for efficient incremental processing. In extensive experiments, DataCell demonstrates efficient performance even compared to specialized stream engines, especially when scalability becomes a crucial factor.

8. GRAPH DATABASES

As database kernel hackers we can not escape the semantic web wave. RDF and triple stores requirements are also challenging the MonetDB kernel. In a recent paper [20], we showed how existing database technology can provide a sound basis for these environments. The base performance of MonetDB for graph database is superb, but perhaps we may find novel tricks when a complete SPARQL front-end emerges on top of it. Most likely, we can re-use many of the techniques developed in the context of MonetDB/XQuery, in particular run-time query optimization [12]. Nevertheless, we did not chicken out and got ourselves lured into European development projects to promote Linked-Open-Data. A step towards this goal is to carve out a benchmark that would shed light on the requirements in this field.

9. FUTURE

Despite the broad portfolio of topics, there is a strong drive and interest in pushing the boundaries of our knowledge by seeking areas hitherto unexplored. The mission for the future is to seek solutions where the DBMS *interpret queries by their intent*, rather than as a contract carved in stone for complete and correct answers. The result set should aid the user in understanding the database's content and provide guidance to continue his data exploration journey. A scientist can stepwise explore deeper and deeper into the database, and stop when the result content and quality reaches his satisfaction point. At the same time, response times should be close to instant such that they allow a scientist to *interact* with the system and explore the data in a contextualized way.

In our recent VLDB 2011 Challenges & Visions paper [13], we chartered a route for such groundbreaking database research along five dimensions:

- One-minute DBMS for real-time performance.
- Multi-scale query processing.
- Post processing for conveying meaningful data.
- Query morphing to adjust for proximity results.
- Query alternatives for lack of providence.

Each direction would serve several PhDs and produce a database system with little resemblance to what we have built over the last thirty years. We look forward to seeing members of the database research community join our mission and take up the challenges expressed.

10. ACKNOWLEDGMENTS

This research was made possible with the dedication of the 2011 research team comprised of Niels Nes, Peter Boncz, Fabian Groffen, Ying Zhang, Stratos Idreos, Milena Ivanova, Romulo Goncavles, Erietta Liarou, Lefteris Sidirourgos, Arjen de Rijke, Holger Pirk, Phan Minh Duc, Bart Scheers, Eleni Petraki, Thibault Sellam, Yagiz Kargin.

11. REFERENCES

[1] P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In *Proc. of the ACM Int'l Conf. on Management of Data (SIGMOD)*, pages 479–490, June 2006.

[2] P. A. Boncz, S. Manegold, and M. L. Kersten. Database Architecture Optimized for the New Bottleneck: Memory Access. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 54–65, Sept. 1999.

[3] P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proc. of the Int'l Conf. on Innovative Data Systems Research (CIDR)*, pages 225–237, Jan. 2005.

[4] R. Goncalves and M. L. Kersten. The Data Cyclotron Query Processing Scheme. In *Proc. of the Int'l Conf. on Extending Database Technology (EDBT)*, pages 75–86, Mar. 2010.

[5] F. Groffen, M. L. Kersten, and S. Manegold. Armada: a Reference Model for an Evolving Database System. In *Proc. of the GI-Fachtagung Datenbanksysteme in Business, Technologie und Web (BTW)*, pages 417–435, Mar. 2007.

[6] S. Héman, N. Nes, M. Zukowski, and P. A. Boncz. Vectorized data processing on the cell broadband engine. In *Proc. of the Int'l Workshop on Data Management on New Hardware (DaMoN)*, page 4, June 2007.

[7] S. Idreos, M. L. Kersten, and S. Manegold. Database Cracking. In *Proc. of the Int'l Conf. on Innovative Data Systems Research (CIDR)*, pages 68–78, Jan. 2007.

[8] S. Idreos, M. L. Kersten, and S. Manegold. Updating a Cracked Database. In *Proc. of the ACM Int'l Conf. on Management of Data (SIGMOD)*, pages 413–424, June 2007.

[9] S. Idreos, M. L. Kersten, and S. Manegold. Self-organizing Tuple Reconstruction in Column-stores. In *Proc. of the ACM Int'l Conf. on Management of Data (SIGMOD)*, pages 297–308, June 2009.

[10] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores. *Proceedings of the VLDB Endowment (PVLDB)*, 4(9):585–597, June 2011.

[11] M. Ivanova, M. L. Kersten, N. Nes, and R. Goncalves. An Architecture for Recycling Intermediates in a Column-store. In *Proc. of the ACM Int'l Conf. on Management of Data (SIGMOD)*, pages 309–320, June 2009.

[12] R. A. Kader, P. A. Boncz, S. Manegold, and M. van Keulen. ROX: Run-time Optimization of XQueries. In *Proc. of the ACM Int'l Conf. on Management of Data (SIGMOD)*, pages 615–626, June 2009.

[13] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1474–1477, Aug. 2011.

[14] M. L. Kersten, E. Liarou, and R. Goncalves. A Query Language for a Data Refinery Cell. In *Proc. of the Int'l Workshop on Event-driven Architecture, Processing and Systems (EDA-PS)*, Sept. 2007.

[15] E. Liarou, R. Goncalves, and S. Idreos. Exploiting the Power of Relational Databases for Efficient Stream Processing. In *Proc. of the Int'l Conf. on Extending Database Technology (EDBT)*, pages 323–334, Mar. 2009.

[16] S. Manegold, P. A. Boncz, and M. L. Kersten. Generic Database Cost Models for Hierarchical Memory Systems. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 191–202, Aug. 2002.

[17] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizing Main-Memory Join On Modern Hardware. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(4):709–730, July 2002.

[18] S. Manegold, P. A. Boncz, N. Nes, and M. L. Kersten. Cache-Conscious Radix-Decluster Projections. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 684–695, Aug. 2004.

[19] S. Manegold, M. L. Kersten, and P. A. Boncz. Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2):1648–1653, Aug. 2009.

[20] L. Sidirourgos, R. Goncalves, M. L. Kersten, N. Nes, and S. Manegold. Column-Store Support for RDF Data Management: not all swans are white. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 1553–1563, Sept. 2008.

[21] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *Proc. of the Int'l Conf. on Innovative Data Systems Research (CIDR)*, pages 296–301, Jan. 2011.

[22] M. Zukowski, N. Nes, and P. A. Boncz. DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In *Proc. of the Int'l Workshop on Data Management on New Hardware (DaMoN)*, pages 47–54, June 2008.