Vectorwise: a Vectorized Analytical DBMS

Marcin Zukowski, Actian Netherlands
Mark van de Wiel, Actian Corp
Peter Boncz, CWI

***Abstract*— Vectorwise is a new entrant in the analytical database marketplace whose technology comes straight from innovations in the database research community in the past years. The product has since made waves due to its excellent performance in analytical customer workloads as well as benchmarks. We describe the history of Vectorwise, as well as its basic architecture and the experiences in turning a technology developed in an academic context into a commercial-grade product. Finally, we turn our attention to recent performance results, most notably on the TPC-H benchmark at various sizes.**

I. Introduction

Even though someone once said that database research is like "polishing a round ball" or some academic researchers might believe that they are no longer able to impact a mature industry like database systems dominated by multi-billion dollar behemoth companies, the story of Vectorwise tells that one should not stop dreaming and setting high goals. The CWI database research group as whole has long worked on analytical database systems, and achieved significant academic acclaim with that, yet reaching the top of the TPC-H leaderboard in per-core performance through its Vectorwise spin-off, in the first months of 2011 was a special kind of success, if one realizes how such a small team of people managed to beat products that are years on the market, backed by deep R&D budgets and the collective brains of many smart research colleagues. In this presentation, we will first discuss the history of the Vectorwise project. Subsequently we will describe its technical architecture, and in particular, how the X100 system developed at CWI was integrated into the Ingres RDBMS. Finally, we will turn our attention to some of the performance characteristics of the Vectorwise system.

A. Abridged History of VectorWise

The CWI database research group develops MonetDB, a main-memory columnar database system designed for analytical workloads, whose column-at-a-time processing model materializes full intermediate results. This materialization may lead to very significant, avoidable, resource consumption. By 2005 the X100 follow-up project at CWI, was started to remove that problem, yet keep the efficient architecture-conscious execution model that characterizes MonetDB.

The X100 engine pioneered the concept of *vectorized query execution* [1], which strikes a balance between the full materialization of MonetDB and tuple-at-a-time pipelining used in most other database systems. While tuple-pipelining is a logical and elegant model, straightforward implementations of it, that transport just a single tuple-at-a-time through a query pipeline are bound to spend most execution time in interpretation overhead rather than query execution. Vectorwise tends to be more than 10 times faster than pipelined query engines in terms of raw processing power, and since it avoids the penalties of full materialization, is also significantly faster than MonetDB. The X100 query execution engine turned out to be so fast that to keep it I/O balanced, the research focus shifted to providing fast I/O, leading to novel compression schemes (e.g. PFOR [2]), hybrid PAX/DSM storage [3], and bandwidth sharing by concurrent queries (Cooperative Scans [4]). Additionally, column-friendly differential update schemes (PDTs [5] were devised.

After the project had reached a certain level of maturity, CWI founded Vectorwise in August 2008 as a spin-off and entered into a collaboration agreement with Ingres Corp. (now Actian) to integrate X100 in its Ingres RDBMS in order to develop a new analytical database product. The product was initially developed by a small team of core engineers and researchers in Amsterdam, with help of Ingres engineers to integrate the two systems. The project was announced in summer 2009, went alpha at the end of 2009, and beta in March 2010. Version 1.0 was released in June 2010 and version 2.0, which also runs on Windows, in October 2011. Vectorwise is currently targeting mid-sized (multi-TB) data warehouses and data marts where its performance is very high and it holds a simplicity and cost advantage to MPP alternatives. The product has been well received by the market, and especially by a rapidly increasing community of customers.

B. Technical Architecture

Forced by the restrictions of 16-bits machines (!) the architecture of Ingres [6] was historically decomposed in multiple server processes, a fact that remains in the 2011 64-bits release of Ingres 10. This multi-process setup made it easy to add the X100 engine into it. Vectorwise combines both the Ingres execution and storage engine and X100 execution and storage, and allow to store data in either kind of table, where "classic" Ingres storage favors OLTP style access and Vectorwise storage favors OLAP. To this end, the Ingres `CREATE TABLE` DDL which already supports various table types, was extended with a (default) `VECTORWISE` type.

Significant modification were made to the Ingres query optimizer, mostly to improve its handling of complex analytical queries; such modifications, e.g. functional dependency tracking and subquery re-use, also benefit Ingres 10. A fully new component in the Ingres architecture, finally, is the cross compiler [7] that translates optimized relational plans into

algebraic X100 plans. The Ingres query optimizer provides quite accurate histogram-based query estimation, and the long lead time in writing a new optimizer from scratch made us choose to improve rather than re-implement it for Vectorwise. Adding features in the optimizer was sometimes possible, but due to the high complexity of this component and its relative isolation from the query execution it was ofter easier to implement new features in other layers. In particular, we also implemented a column-oriented *rewriter* module inside the X100 system. It is a rule-based rewriting system using the Tom pattern matching tool [8]. The presence of a separate, column-oriented rewriter has aided the project as certain functionality can be engineered with much less impact to the main Ingres optimizer. The rewriter has been used to implement a variety of functionalities, which include among others null handling and multi-core parallelization.

To avoid making all query execution operators and functions NULL-aware, and therefore more complex and slower, Vectorwise internally represents NULLs as two columns: a binary NULL indicator and a value column, with a "safe" value for NULLs; stored together in PAX. In the rewriter phase, operations on NULLable inputs are rewritten into equivalent operations on two "standard" relational inputs.

The original X100 prototype was single-threaded, but given the multi-core trend it became obvious, that a new analytical database system could only go to market with multi-core capability. The Vectorwise rewriter was used to implement a Volcano-style query parallellizer, which realized this feature quite well in a short time span.

Transactions in Vectorwise are based on Positional Delta Trees (PDT [5]). PDTs are an advanced form of differential data structures, which help keeping updates fast in column-oriented database systems. The idea is not to perform updates in place, which for each modified record would lead to one I/O for each column, plus additional recompression overhead. Rather, updates are gathered in a differential structure, that contains differences with respect to the stable table image. Incoming queries hence need to merge in the differences from these structures, while they scan data from disk. In case of PDTs, this differential structure annotates changes (insertions, deletions, updates) by tuple *position* rather than by tuple key values. The advantage of positional deltas is that incoming queries can merge in changes faster and queries do not need to scan the key columns to do so. In order to provide full ACID properties, Vectorwise uses a Write Ahead Log that logs PDTs as they are committed and performs optimistic PDT-based concurrency control. Further, it ensures its transactional model is in sync with that of the existing Ingres RDBMS.

C. VectorWise Performance

There is hardly a single customer or prospect testing Vectorwise who does not quickly note its extremely high performance; which is the main reason for its popularity. To demonstrate this, a number of audited TPC-H benchmarks scores were released:

- a 100GB result on a 144GB HP DL380 (2-socket, 12-core) producing a 251K QphH score on Linux. This score is quite comparable to a SQLserver score on almost the same hardware that yielded 74K on Windows.
- another 100GB result on a 192GB Dell R610 (also a 2-socket, 12-core) producing a 303K QphH score on Linux. The higher score is partially due to using a slightly newer version of Vectorwise (1.6 vs 1.5).
- a 300GB result on a 512GB Dell R910 (a 4-socket, 32-core) producing a 400K QphH score on Linux.
- a 1TB result on a 1024GB Dell R910 (also a 4-socket, 32-core) producing a 436K QphH score on Linux.

All results were still leading for non-clustered database systems at time of this writing.

One might note that in all cases the RAM size is larger than the database size. While we already named many innovations to boost the I/O performance of Vectorwise, and the TPC-H benchmark could run on machines with much less memory, the reason to choose this configuration was that *avoiding* I/O is in absolute terms always better than coping with it efficiently, and achieving highest overall performance is the goal of such benchmark exercises.

Limited optimization effort was spent to achieve these TPC-H performance results, mostly in producing better parallellizing plans for the power run, as well as better load balancing, which benefits the throughput run. Some other effort was spent in making updates faster, this was especially relevant in the throughput runs.

ACKNOWLEDGMENT

We would like to thank the entire Vectorwise team, including Sandor Heman, Michal Szafranski, Giel de Nijs, Gosia Wrzesinska, Kamil Anikiej, Hui Li, Willem Konijnenberg, Juliusz Sompolski, Michal Switakowski and Ala Luszczak, as well as Doug Inkster, Zelaine Fong, Karl Schendel and Ian Kirkham and Keith Bolam for their contributions.

REFERENCES

[1] P. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-pipelining query execution." in *CIDR*, 2005.
[2] M. Zukowski, S. Heman, N. Nes, and P. Boncz, "Super-Scalar RAM-CPU Cache Compression," in *ICDE*, 2006.
[3] M. Zukowski, "Balancing vectorized query execution with bandwidth-optimized storage," 2009.
[4] M. Zukowski, S. Héman, N. Nes, and P. Boncz, "Cooperative scans: Dynamic bandwidth sharing in a dbms," in *Proceedings of VLDB*. VLDB Endowment, 2007, pp. 723–734.
[5] S. Héman, M. Zukowski, N. Nes, L. Sidirourgos, and P. Boncz, "Positional update handling in column stores," in *Proceedings of SIGMOD*. ACM, 2010, pp. 543–554.
[6] M. Stonebraker, *The INGRES Papers: Anatomy of a Relational Database System*. Addison-Wesley Longman Publishing Co., Inc., 1986.
[7] D. Inkster, M. Zukowski, and P. Boncz, "Integration of VectorWise with Ingres," *SIGMOD Record*, vol. 40, no. 3, Sept. 2011.
[8] Tom, http://tom.loria.fr.