From X100 to Vectorwise: opportunities, challenges and things most researchers do not think about

Marcin Zukowski, Actian Netherlands
Peter Boncz, CWI

ABSTRACT

In 2008 a group of researchers behind the X100 database kernel created Vectorwise: a spin-off which together with the Actian corporation (previously Ingres) worked on bringing this technology to the market. Today, Vectorwise is a popular product and one of the examples of conversion of a research prototype into successful commercial software. We describe here some of the interesting aspects of the work performed by the Vectorwise development team in the process, and discuss the opportunities and challenges resulting from the decision of integrating a prototype-quality kernel with Ingres, an established commercial product. We also discuss how requirements coming from real-life scenarios sometimes clashed with design choices and simplifications often found in research projects, and how Vectorwise team addressed some of of them.

1. THE MAKING OF VECTORWISE

The X100 engine prototyped by the database research group of CWI pioneered the concept of *vectorized query execution* [6, 1], which allows modern CPU to process queries more than 10 times faster than conventional query engines. The system was so fast that to keep it I/O balanced, research focus shifted to storage, leading to novel compression schemes (e.g. PFOR [8]), hybrid PAX/DSM storage [6], and bandwidth sharing by concurrent queries (Cooperative Scans [7]). Additionally, column-friendly differential update schemes (PDTs [2]) were devised.

In the following, we tell the story of how the X100 research prototype was integrated with the Ingres engine and converted into a high-performance analytical database product of Actian Corp. and what we learned on the way of that process. Completely replacing the engine of a car is a process that involves reconnecting

SIGMOD Record.

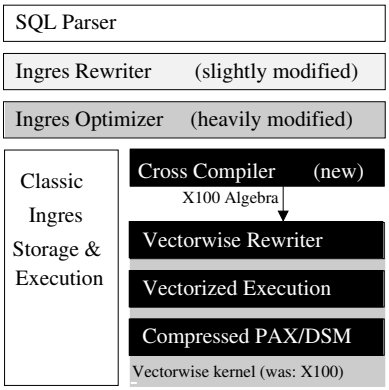

Figure 1: Architecture of Vectorwise.

many cables, tubes and levers, and changing the engine of a DBMS is similarly complex. This task was accomplished by the Vectorwise team working together with Ingres engineers. Further, significant changes where made to boost this engine further ("add turbo"), but also involved also more mundane things to make driving the new car more pleasurable (e.g. "adding cupholders").

Combining X100 and Ingres. Forced by the restrictions of 16-bits machines (!) the architecture of Ingres [4] 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, as depicted in Figure 1. 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 [3] that translates optimized relational plans into algebraic X100 plans.

Extending Vectorwise. When the X100 project was spun-off, it contained only basic features of an industry-grade DBMS. While query processing and storage pro-

vided record-breaking performance, it lacked greatly in terms of usability, feature completeness and, as one would expect from a research prototype, stability and code quality. At that time it became clear that a decision needs to be made on how to fill various functionality holes in the system. This was possible at various levels
• *Ingres query optimizer.* The Ingres query optimizer provides solid, 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. Adding features at this level was sometimes possible, but due to the high complexity of this component and its relative isolation from the query execution it was usually easier to implement things in other layers.
• *X100 kernel.* Implementing new featured directly in the X100 execution kernel often offered opportunities for optimal performance. Still, this often required a significant amount of work and some added functionality was much easier to express on a higher level.
• *X100 rewriter.* To combine benefits of using a higher-level tool and having a tightly integrated solution, we implemented a column-oriented *rewriter* module inside the X100 system. It is a rule-based rewriting system using the Tom pattern matching tool [5].

In the following, we will discuss some of the extensions performed at various levels.

Mundane things. While during research we focused on challenging and exciting things, creating a real product often required significant work on relatively mundane tasks.
• *Many Functions.* SQL standard contains a plethora of functions, in particular around strings and dates. Furthermore, many DBMSs implement non-standard functions which users migrating from these systems need to port their existing applications. This resulted in dozens of new functions added to the system. A challenge was to implement them all efficiently, both in terms of development time as well as of their final performance. Some functions were implemented in the rewriter phase, by simplifying them or expressing as combinations of other functions. For others, manual implementation was needed.
• *NULLs.* To avoid making all query execution operators and functions NULL-aware, Vectorwise internally represents NULLs as two columns: a binary null indicator and a value column, with a "safe" value for NULLs. In the rewriter phase, operations on NULLable inputs are rewritten into equivalent operations on two "standard" relational inputs.
• *Error handling and reporting.* The original X100 functions often assumed a simplified view of the world, where a user never issues a query that can fail. If it did, the system might have crashed or incorrect results might have been returned. For a production system, we had to add a significant number of extra functionality detecting issues like division by zero, incorrect function parameters, or arithmetic overflows. Naive implementation for some of these would incur a significant overhead, and special algorithms in the kernel had to be devised.
• *System monitoring.* For analyzing the system we had to extend it significantly in areas like event logging, load and resource monitoring, query listing etc.

Challenging things. Many other extensions turned out to be much more complex and challenging. Some examples include:
• *Multi-core.* 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 parallelizer; yet getting the best out of modern multi-core CPUs is not simple.
• *NULL intricacies.* While most operators are NULL oblivious, one of the exceptions were join operators. Here, intricacies of the SQL semantics of anti-joins added significant complexity to the Vectorwise rewriter and operator implementations.
• *Transactions.* Transactions in Vectorwise are based on Positional Delta Trees (PDT [2]). Implementing full transactional support in a system with complex indexing structures and background update propagation was quite complicated.
• *Query cancellation.* This was one of more unexpected feature requests. In a prototype stage it was perfectly fine to press "control-C" or issue a "kill" command to stop a query. In a production system it was, obviously, not a good solution. Performing a proper query cancellation turned out a much more complex task than initially expected, mostly due to aspects such as parallelism, asynchronous IO and memory management.

2. SUMMARY

This presentation demonstrated some more interesting challenges that we faced on our road from a research prototype to a production-grade commercial product. We hope it serves partially as a motivating example and partially as a friendly warning to our fellow researchers not to forget how different an academic project and a real-life system need to be.

3. REFERENCES

[1] Peter Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-pipelining query execution. In *CIDR*, 2005.

[2] S. Héman, M. Zukowski, N.J. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *Proceedings of SIGMOD*, pages 543–554. ACM, 2010.

[3] Doug Inkster, Marcin Zukowski, and Peter Boncz. Integration of VectorWise with Ingres. *SIGMOD Record*, 40(3), September 2011.

[4] M. Stonebraker. *The INGRES Papers: Anatomy of a Relational Database System.* Addison-Wesley Longman Publishing Co., Inc., 1986.

[5] Tom. `http://tom.loria.fr`.

[6] M. Zukowski. Balancing vectorized query execution with bandwidth-optimized storage. 2009.

[7] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative scans: Dynamic bandwidth sharing in a dbms. In *Proceedings of VLDB*, pages 723–734. VLDB Endowment, 2007.

[8] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-Scalar RAM-CPU Cache Compression. In *ICDE*, 2006.