Column-oriented Database Systems

Daniel J. Abadi
Yale University
New Haven, CT, USA
dna@cs.yale.edu

Peter A. Boncz
CWI
Amsterdam, The Netherlands
p.boncz@cwi.nl

Stavros Harizopoulos
HP Labs
Palo Alto, CA, USA
stavros@hp.com

ABSTRACT

Column-oriented database systems (column-stores) have attracted a lot of attention in the past few years. Column-stores, in a nutshell, store each database table column separately, with attribute values belonging to the same column stored contiguously, compressed, and densely packed, as opposed to traditional database systems that store entire records (rows) one after the other. Reading a subset of a table's columns becomes faster, at the potential expense of excessive disk-head seeking from column to column for scattered reads or updates. After several dozens of research papers and at least a dozen of new column-store start-ups, several questions remain. Are these a new breed of systems or simply old wine in new bottles? How easily can a major row-based system achieve column-store performance? Are column-stores the answer to effortlessly support large-scale data-intensive applications? What are the new, exciting system research problems to tackle? What are the new applications that can be potentially enabled by column-stores? In this tutorial, we present an overview of column-oriented database system technology and address these and other related questions.

1. INTRODUCTION

The intended audience of the tutorial is database system researchers, designers, and practitioners with a keen interest on database system architecture and performance, and on database system support for large-scale, data-intensive applications (especially data warehousing and business intelligence). The expected main learning outcomes are: (a) increased understanding of column-based data management system architectures, and in what situations and domains they are suitable for large-scale data analysis, (b) in-depth coverage of advanced storage and processing techniques (several of which have not yet been transferred into commercial systems) and of promising avenues for academic and industrial research, and (c) surveying the state-of-the-art in today's academic and commercial offerings, along with promising applications enabled by those systems. We organize the tutorial into three parts which are described next.

VLDB '09, August 24-28, 2009, Lyon, France.

2. PART A

Fundamentals, application areas, and performance tradeoffs

The roots of column-store DBMSs can be traced in the 1970s, when transposed files were first studied, followed by investigations on vertical partitioning as a form of a table attribute clustering technique. By the mid 1980s, the advantages of a fully decomposed storage model (DSM — a predecessor to column-stores) over NSM (traditional row-based storage) were documented [5] and subsequent research on join and projection indices further strengthened the advantages of DSM over NSM. Despite the suitability of the DSM layout for analytical queries, market needs and non-favorable technology trends helped row-based database systems maintain their foothold. It was not until the 2000s that column-store research and commercial systems took off. We trace the history of column-stores and examine in detail those technology and application trends that lead to the resurgence of research and commercialization of column-stores.

A columnar data layout dictates much of the basic architectural design in a column-store. Each column is stored contiguously on a separate location on disk, typically using large disk pages (or large read units) to amortize disk head seeks when scanning multiple columns on hard drives. To improve read efficiency, columnar values are typically densely packed, foregoing the explicit storage of record IDs, and using light-weight compression schemes whenever possible (e.g., [7]). Column scan operators differ from row-scanners in that they are responsible for translating value position information into disk locations and for combining and reconstructing (when needed) partial or entire tuples out of different columns. Join operators can either rely on column-scanners for receiving reconstructed tuples, or they can operate directly on columns by first computing a join index and then fetching qualifying values [8]. We cover architectural considerations and implementation of basic query processing mechanisms that are common across column-stores.

We then provide an overview of the application areas for column-stores. We first cover the traditional areas like data warehousing, business intelligence and data mining, and discuss the trends in these, such as personal data marts (MS Gemini) and the need for on-line updates and loads. Another promising direction is scientific data management, where we showcase some of the results with MonetDB on the Sloan Digital Sky Survey. Finally, we describe recent work in addressing alternative data models, such as object data models, array data models, XML and RDF using column-stores.

We conclude the first part with an examination of performance tradeoffs in row, column, and hybrid data representations, both on disk and in main memory. The resurgence of on-disk columnar

storage was preceded by interest in investigating in-memory data layouts for addressing the growing speed disparity between CPU and RAM in the late 1990s. The main-memory advantages of DSM were extensively demonstrated in the MonetDB and PAX projects. The recent growing popularity of full-blown column-stores has sparked an interest in understanding performance tradeoffs of read-optimized databases that utilize columnar representation and efficient compression schemes (e.g., [6]).

3. PART B

Column-store internals & advanced query processing techniques

One of the most-often cited advantages of column-stores is data compression. The intuitive argument for why column-stores compress data well is that compression algorithms perform better on data with low information entropy (high data value locality). We discuss some compression algorithms that work particularly well in column-store databases, and how one can architect a query executer to operate directly on compressed data (so that the system can achieve the I/O performance benefits of compression without paying the CPU cost of decompression) [1, 9]. We also discuss some recent work that argues that much of the compression benefits of column-stores are not necessarily unique to columns-stores; if fact, they can also be applied to row-stores.

Two of the most-often cited disadvantages of column-stores are write operations and tuple construction. Write operations are generally considered problematic for two reasons: (a) inserted tuples have to be broken up into their component attributes and each attribute must be written separately, and (b) the dense-packed data layout makes moving tuples within a page nearly impossible. We discuss some of the techniques that column-stores use to mitigate their fundamental write issues, such as in-memory buffering, tuple moving, and partition-merging. Tuple construction is also considered problematic since information about a logical entity is stored in multiple locations on disk, yet most queries access more than one attribute from an entity. Further, most database standards (e.g., ODBC and JDBC) access results entity-at-a-time (not column-at-a-time). Thus, at some point in a query plan, data from multiple columns must be combined into 'rows' of information about an entity. We discuss techniques developed to reduce such construction costs (e.g., [2]).

To provide a better understanding of some of the distinguishing features of the internals of column-stores, we discuss some recent work that attempts to simulate column-stores inside row-stores (e.g., [3]). This work isolates some of the key techniques that result in performance gains for column-store databases, and motivates an effort, both in research and industry, to build various types of hybrid systems, and achieve some convergence between column-store and row-store technologies. We also discuss some open research problems in the context of column-store systems, including physical database design, indexing techniques, parallel query execution, replication, and load balancing.

4. PART C

Case study and review of column-store products

We illustrate many of the techniques outlined, by taking an in-depth look at the architecture of MonetDB/X100, which is, in fact, two different column-stores, developed at CWI. Both X100 [9] and MonetDB [4], apart from pursuing columnar storage from a data access point of view, focused on exploiting column-wise data processing as a way to improve the computational efficiency of database engines, in particular as a way to better exploit the features of modern processors. In the case of MonetDB, column-wise query processing was realized by a physical column algebra — the idea being that the simplicity of columnar algebras provide opportunities for more efficient execution primitives (similar to RISC vs. CISC). We explain these opportunities in detail, covering improving instruction throughput in pipelined CPUs, CPU branch prediction efficiency and CPU cache efficiency, and also exploiting SIMD instructions and GPU hardware.

In X100, the benefits of the column algebras were conserved, but extended with pipelined query processing in its vectorized query processing model. We elaborate on the challenges and opportunities of vectorized query processing and also cover X100's new vectorized columnar compression schemes, as well as its innovations in column-store updates. Finally, we discuss the tension between magnetic disk capabilities and data access needs of data warehousing and column-stores, and outline work on intelligent I/O scheduling for column-stores (e.g., in the area of data clustering and cooperative scans [10]). This discussion also covers the potential impact of solid state drives (SSD) technology on column storage (e.g., [8]).

We conclude the tutorial with a comprehensive review of commercially available column-stores, including Kdb, MonetDB, VectorWise, Vertica/C-Store, Sybase IQ, Infobright, Exasol, ParAccel, the SAP BI accelerator, and Kickfire.

5. REFERENCES

[1] Abadi, D.J., Madden, S.R., and Ferreira, M.: Integrating compression and execution in column-oriented database systems. In *Proc. SIGMOD*, 2006.

[2] Abadi, D.J., Myers, D.S., DeWitt, D.J., and Madden, S.R.: Materialization strategies in a column-oriented DBMS. In *Proc. ICDE*, 2007.

[3] Abadi, D.J., Madden, S.R., and Hachem, N.: Column-stores vs. row-stores: how different are they really? In *Proc. SIGMOD*, 2008.

[4] Boncz, P.A. Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications. Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[5] Copeland, G.P., Khoshafian, S.N.: A Decomposition Storage Model. In *Proc. SIGMOD*, 1985.

[6] Harizopoulos, S., Liang, V., Abadi, D.J., and Madden, S.: Performance tradeoffs in read-optimized databases. In *Proc. VLDB*, 2006.

[7] Stonebraker, M. et al.: C-Store: A Column-oriented DBMS. In *Proc. VLDB*, 2005.

[8] Tsirogiannis, D., Harizopoulos, S., Shah, M.A., Wiener, J.L., and Graefe, G.: Query processing techniques for solid state drives. In *Proc. SIGMOD*, 2009.

[9] Zukowski, M., Heman, S., Nes, N., and Boncz, P.A.: Super-scalar ram-cpu cache compression. In *Proc. ICDE*, 2006.

[10] Zukowski, M., Heman, S., Nes, N., and Boncz, P.A. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *Proc. VLDB*, 2007.