Transparent Data Transformation

Can I have my Rows and eat my Columns too?

Manos Athanassoulis
Boston University

A big dichotomy in data system design is the column- vs. row-stores one. The first supports analytical, and the latter transactional workloads. There have been several efforts to bridge the two, especially in light of new hybrid transactional analytical processing workloads.

For example, SAP HANA [11] and Oracle TimesTen [8] use in-memory column-stores to offer efficient analytical processing and employ a row-wise write-store to support ACID transactions. MemSQL uses a row-store for data ingestion in memory, and writes columns on disk to reduce future I/O [13]. IBM dashDB is a hybrid store that supports mixed workloads [4]. Academic systems [2, 3, 9] also combine columnar and row-wise architectures. They all require conversions between row-wise and columnar formats, and, hence, they have to balance between efficient analytics and data freshness. ***What if we had a way to decouple the physical data layout from the data access performance?***

In other words, what if we could store *any layout* L_1, and ship through the memory hierarchy *any layout* L_2, transparently converting rows to columns and vice versa? We now have two fundamental questions: (a) how to offer such *transparent data transformation*? (b) once we have it, how to *change the data system's architecture*? In this short note we focus on the first challenge.

Data movement is one of the biggest inefficiencies in computing [5], so the initial motivation of column-stores is to avoid unnecessary column accesses. Row-wise layouts allow transactional workloads to ensure update and insert locality and avoid updating multiple locations.

In an in-memory system, we can address both these requirements with a *smart* memory controller that *implements projection and offers access to arbitrary groups of columns*. As Moore's law is slowing down, domain-specific accelerators are becoming viable in the long run [6], hence, we envision to embed a light-weight vertical partitioner in the memory c ontroller to on-the-fly project arbitrary groups of columns. This controller will have access to a much higher bandwidth than what is exposed to the processor, as it will be able to exploit in full the parallelism of memory chips. The query processor can decide at compile-time the optimal layout for each query and request it from the memory controller. From the software side, a special type of *ephemeral variables* can act as pointers to virtual hybrid layouts (similar to a non-materialized view). Accessing them would start the projection machinery and propagate through the cache hierarchy the desired layout without creating a physical copy in main memory. This vision has similar motivation with disaggregated memory [10], however, it aims to build local *smart* memory, that can also allow cloud systems to access disaggregated *smart* memory.

Similarly, in a disk-based system, we can address these challenges by pushing projection to storage and send up the memory hierarchy the desired data layout for each query. By employing modern devices like *smart SSDs* [7], we can implement in-storage custom logic to vertically partition data. Analytical queries will access read-only versions of their optimal layout without creating permanent copies in memory. Updates will access row-oriented base data, and existing read-only versions of columns will be invalidated when relevant updates are received and old queries complete. Pages with column projections are marked as read-only, while pages with full rows are marked as read/write. The two levels of *transparent data transformation* can work synergistically.

How to achieve this? Building an FPGA prototype memory controller requires collaboration between data management and hardware researchers [1]. Near-data execution of relational operators [14] using programmable logic in the middle [12] will use an on-the-fly column extractor which compacts the useful data and ships it through the cache hierarchy. Ultimately, our goal is to build new memory and disk controllers to offer *transparent data transformation*.

What is next? If a query can consume data with the ideal layout, how should query optimization change? What about heuristics, like pushing selection? Can this be extended to allow for efficient access in arbitrary slices of tensors and matrices? *This vision will fuel strong interaction between data management, hardware, programming languages and software engineering.*

1. REFERENCES

[1] G. Alonso, T. Roscoe, D. Cock, M. Ewaida, K. Kara, D. Korolija, D. Sidler, and Z. Wang. Tackling Hardware/Software co-design from a database perspective. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2020.

[2] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki. The Case For Heterogeneous HTAP. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.

[3] J. Arulraj, A. Pavlo, and P. Menon. Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 583–598, 2016.

[4] R. Barber, G. M. Lohman, V. Raman, R. Sidle, S. Lightstone, and B. Schiefer. In-Memory BLU Acceleration in IBM's DB2 and dashDB: Optimized for Modern Workloads and Hardware Architectures. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2015.

[5] W. J. Dally. Power, programmability, and granularity: The challenges of ExaScale computing. In *Proceedings of the IEEE International Test Conference (ITC)*, page 12, 2011.

[6] W. J. Dally, Y. Turakhia, and S. Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, 2020.

[7] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt. Query processing on smart SSDs: opportunities and challenges. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1221–1230, 2013.

[8] T. Lahiri, M.-A. Neimat, and S. Folkman. Oracle TimesTen: An In-Memory Database for Enterprise Applications. *IEEE Data Engineering Bulletin*, 36(2):6–13, 2013.

[9] H. Lang, T. Mühlbauer, F. Funke, P. A. Boncz, T. Neumann, and A. Kemper. Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2016.

[10] K. T. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch. System-level implications of disaggregated memory. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 189–200, 2012.

[11] N. May, A. Böhm, and W. Lehner. SAP HANA - The Evolution of an In-Memory DBMS from Pure OLAP Processing Towards Mixed Workloads. In *Proceedings of the Datenbanksysteme fur Business, Technologie und Web (BTW)*, pages 545–563, 2017.

[12] S. Roozkhosh and R. Mancuso. The Potential of Programmable Logic in the Middle: Cache Bleaching. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 296–309, 2020.

[13] N. Shamgunov. The MemSQL In-Memory Database System. In *Proceedings of the International Workshop on In-Memory Data Management and Analytics (IMDM)*, 2014.

[14] S. L. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos. Beyond the Wall: Near-Data Processing for Databases. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, page 2, 2015.