Microsoft SQL Server's Integrated Database Approach for Modern Applications and Hardware

David Lomet
Microsoft Research
One Microsoft Way
Redmond, WA 98052
lomet@microsoft.com

ABSTRACT

Recently, there has been much renewed interest in re-architecting database systems to exploit new hardware. While some efforts have suggested that one needs specialized engines ("one size does not fit all"), the approach pursued by Microsoft's SQL Server has been to integrate multiple elements into a common architecture. This brings customers what they want by reducing data impedance mismatches between database systems that they are using for multiple purposes. This integration is, of course, more easily said than done. But this is, in fact, precisely what the SQL Server team has done.

1. INTRODUCTION

1.1 The Present Challenges

There have been two technical threads in the database community that have an impact on the way that database engines are built.

1. Modern hardware makes the classic approach to building database systems obsolete. This classic approach assumed that data resided on disk, accessed via a buffer pool of disk pages. Access was controlled via a pessimistic lock manager. Recovery was page oriented. Data was row oriented to enable rapid record update. Etc.

2. The diverse tasks to which database systems are now applied means that they need to be good at more than OLTP, the classic database workload. OLAP, data mining, etc. all put more demands on query processing. Further, the data volumes keep increasing far beyond what the original database system designers could possibly have anticipated.

1.2 A Path Not Taken

A frequent approach to dealing with the challenges our field faces is to design specialized database engines that are tailored, for example, to transaction processing. Another engine then would be designed to handle analytic queries. This is a very convenient dichotomy in some respects, as we more or less understand how to solve these problems in isolation.

Unfortunately, our customers have data that they wish to flexibly process, in several potential ways. They would rather not learn how to deal with multiple pieces of technology—preferring to simplify administration and management of their database infrastructure. Data that may start as transaction processing data can become the basis for subsequent data analysis. So customers have data management needs that are not so easily partitioned.

Proceedings of the VLDB Endowment, Vol. 6, No. 11

1.3 SQL Server's Unified Approach

At Microsoft, we decided to take an integrated approach, because our assessment is that, other things being equal, that is what customers would prefer. The "other things being equal" is, of course, the "fly in the ointment". The SQL team knew what technologies needed to be employed at the 40,000 ft. level.

1. Main memory focused row store for transaction processing.
2. Column store for compressing and efficiently processing huge volumes of analytic data.
3. Mixed (perhaps conventional) techniques for queries that are not so conveniently separated.

So the technical challenge was to build this collection of technologies so as to integrate them into a single database engine. Further, of course, since SQL Server has a very large base of existing customers, we needed to avoid disrupting these customers. So compatibility in some form for existing customers was essential.

The rest of this short paper outlines the new technologies that we added to SQL Server "classic edition", and sketches how they were integrated into a single database engine. Section 2 describes the SQL column store data analysis focused facilities, while section 3 describes the main memory transaction processing focused facilities. Section 4 then briefly describes how they are integrated to give our customers a "unified experience".

2. COLUMN FORMATTED DATA

2.1 The Data Analytics Problem

So what is the problem that needs to be solved in the data analysis space? It is data volumes and their impact on the cost and performance of the systems that process what is mostly analytic data. The amount of this data is now growing at a furious pace. Column formatted data attacks this problem in two fundamental aspects.

1. Only the attributes of relational data that play a role in a query need be accessed by the query as the attributes are stored and can be accessed independently of other attributes.

2. In addition to isolating the data involved to only the attributes needed, column formatted data enables data compression to squeeze down the bytes that are moved and compared and processed by another significant factor.

2.2 The SQL Apollo Approach

In SQL Server, we wanted columnar data to be integrated with row data, as even analytic queries can access data that is more conveniently stored as rows. This resulted in SQL Server's design in which columns are treated as index data [2, 3]. This means that queries can access data in both column and row format. The query optimizer will then choose a plan that optimizes across this now expanded physical database design space.

The columns can be compressed when stored, with run length encoding being an option that can drastically reduce data footprint. Microsoft has unique technology, adopted from its BI group, which produces astounding levels of compression. Further, it is possible to process column data in "batch mode", where aggregates can be computed directly from the compressed data.

3. MAIN MEMORY ROW DATA

3.1 The OLTP Problem

The nature of the OLTP problem is different in kind from the data analysis problem. There, the sheer volumes of data made it difficult for conventional database systems to provide any reasonable support. In the transaction processing space, database systems have been very nicely handling customer problems for 40 plus years.

The OLTP problem is one of missed opportunities. Modern hardware has not been exploited effectively to improve the cost/performance of transaction processing. This translates directly into customers bearing unnecessarily large costs and system management complexity.

If we can achieve an order of magnitude performance gain in transaction processing, we can simultaneously reduce infrastructure costs, improve query responsiveness, and reduce the complexity of managing large "scaled-up systems".

3.2 The SQL Hekaton Approach

The Hekaton design goal was a system where threads never block. Thread blocking introduces overhead in multiple ways. First, the code to perform a context switch can be substantial. Second, each thread has a thread context whose data is then expelled from the memory caches. This cache thrashing then degrades performance even more.

Several recent main memory database efforts have recognized this thread/memory hierarchy problem. Their starting point was to treat a large server with its multiple cores and memory hierarchy as if it were a "distributed system on a board". The appeal of this approach is that data can be isolated so as to be accessed only by a single core, leading to a "latch free" system by partitioning. But the Hekaton team considered such systems to be inherently fragile, with difficulty responding adequately to users whose transactions resisted any clean partitioning.

Thus, Hekaton [1] adopted a "classic" latch-free approach, wherein threads never block, but this is achieved using optimistic methods. The latch-free part of never blocking is to use latch-free data structures that employ "compare and swap" (CAS) instructions to make state changes to shared data. This led to the incorporation of the latch-free Bw-tree as Hekaton's key-ordered index [5].

But latch-free is not sufficient to eliminate all thread blocking, as blocking can occur at the transaction level via user access patterns in a traditional lock manager approach to concurrency control. Hekaton's concurrency [4] control differs from a traditional lock manager in four respects. (1) It is multi-version (MVCC) so that read-write conflicts are frequently avoided by satisfying the read with an earlier version. (2) There are no separate locks, the data versions themselves recording information needed for CC. (3) The lock manager data structure is the same latch-free hash table used to access the data. (3) The method is optimistic, so no blocking occurs during transaction execution, but validation can lead to aborts. (4) The validation phases is itself non-blocking by exploiting transaction dependencies.

4. THE INTEGRATED ENGINE

There is no shortage of database systems, both research prototypes and commercial systems, which attack some market segment, e.g. OLTP or OLAP, with a carefully selected technology that works well in that segment. But it is rare to find a system, commercial or otherwise, that handles a breadth of market segments with a breadth of technologies, all of which are integrated into a single system that preserves customers' existing database applications. That is what SQL has done.

There is no shortage of clever technology used in SQL Server, its data compression technology for columnar data, its latch-free Bw-tree, its non-blocking multi-version concurrency control. But as remarkable is that it has all been integrated into a single database engine that preserves compatibility for existing customers.

5. CONCLUSIONS

Column format data is already supported in SQL Server. Main memory row data support will appear in the impending release of SQL Server. We believe these new and integrated capabilities will make SQL Server a stronger and more attractive option in the very competitive database market.

A final word. The inspiration, the technology, the integration, the enormous effort were parts of a shared and challenging endeavor that included highly skilled and insightful developers and researchers. This effort is a great example of how a company can benefit by being able to bring developers and researchers together to produce outstanding results.

6. REFERENCES

[1] Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, Mike Zwilling: Hekaton: SQL server's memory-optimized OLTP engine. SIGMOD Conference 2013: 1243-1254

[2] Per-Åke Larson, Cipri Clinciu, Eric N. Hanson, Artem Oks, Susan L. Price, Srikumar Rangarajan, Aleksandras Surna, Qingqing Zhou: SQL server column store indexes. SIGMOD Conference 2011: 1177-1184

[3] Per-Åke Larson, Cipri Clinciu, Campbell Fraser, Eric N. Hanson, Mostafa Mokhtar, Michal Nowakiewicz, Vassilis Papadimos, Susan L. Price, Srikumar Rangarajan, Remus Rusanu, Mayukh Saubhasik: Enhancements to SQL server column stores. SIGMOD Conference 2013: 1159-1168

[4] Per-Åke Larson, Spyros Blanas, Cristian Diaconu, Craig Freedman, Jignesh M. Patel, Mike Zwilling: High-Performance Concurrency Control Mechanisms for Main-Memory Databases. PVLDB 5(4): 298-309 (2011)

[5] Justin J. Levandoski, David B. Lomet, Sudipta Sengupta: The Bw-Tree: A B-tree for new hardware platforms. ICDE 2013: 302-313