How to Build a High-Performance Data Warehouse

By David J. DeWitt, Ph.D.; Samuel Madden, Ph.D.; and Michael Stonebraker, Ph.D.

Over the last decade, the largest data warehouses have increased from 5 to 100 terabytes, according to Winter Corp., and by 2010, most of today's data warehouses will be 10 times larger, according to The Data Warehouse Institute (TDWI). As data warehouses grow in size to accommodate regulatory requirements and competitive pressures, ensuring adequate database performance will be a big challenge in order to answer more ad hoc queries from more people. This article examines the various ways databases are architected to handle the rapidly increasing scalability requirements, and the best combination of approaches for achieving high performance at low cost.

Obviously, there are limits to the performance of any individual processor (CPU) or individual disk. Hence, all high-performance computers include multiple CPUs and multiple disks. Similarly, a high-performance DBMS must take advantage of multiple disks and multiple CPUs. However, there are significant differences between various databases concerning how they are able to take advantage of resources.

In the first section of this article (Better Performance through Parallelism: Three Common Approaches), we discuss hardware architectures and indicate which ones are more scalable. As well, we discuss the commercial DBMSs that run on each architecture. In the second section (Going Even Faster: Hardware Acceleration), we turn to proprietary hardware as an approach to providing additional scalability, and indicate why this has not worked well in the past – and why it is unlikely to do any better in the future. Lastly, the third section (Achieving Scalability through Software) discusses software techniques that can be used to enhance data warehouse performance.

Better Performance through Parallelism: Three Common Approaches

There are three widely used approaches for parallelizing work over additional hardware:

- shared memory
- shared disk
- shared nothing

Shared memory**:** In a shared-memory approach, as implemented on many symmetric multi-processor machines, all of the CPUs share a single memory and a single collection of disks. This approach is relatively easy to program: complex distributed locking and commit protocols are not needed, since the lock manager and buffer pool are both stored in the memory system where they can be easily accessed by all the processors.

Unfortunately, shared-memory systems have fundamental scalability limitations, as all I/O and memory requests have to be transferred over the same bus that all of the processors share,

causing the bandwidth of this bus to rapidly become a bottleneck. In addition, shared-memory multiprocessors require complex, customized hardware to keep their L2 data caches consistent. Hence, it is unusual to see shared-memory machines of larger than 8 or 16 processors unless they are custom-built from non-commodity parts, in which case they are very expensive. Hence, shared-memory systems offer very limited ability to scale.

Shared disk**:** Shared-disk systems suffer from similar scalability limitations. In a shared-disk architecture, there are a number of independent processor nodes, each with its own memory. These nodes all access a single collection of disks, typically in the form of a storage area network (SAN) system or a network-attached storage (NAS) system. This architecture originated with the Digital Equipment Corporation VAXcluster in the early 1980s, and has been widely used by Sun Microsystems and Hewlett-Packard.

Shared-disk architectures have a number of drawbacks that severely limit scalability. First, the interconnection network that connects each of the CPUs to the shared-disk subsystem can become an I/O bottleneck. Second, since there is no pool of memory that is shared by all the processors, there is no obvious place for the lock table or buffer pool to reside. To set locks, one must either centralize the lock manager on one processor or resort to a complex distributed locking protocol. This protocol must use messages to implement in software the same sort of cache-consistency protocol implemented by shared-memory multiprocessors in hardware. Either of these approaches to locking is likely to become a bottleneck as the system is scaled.

To make shared-disk technology work better, vendors typically implement a “shared-cache” design. Shared cache works much like shared disk, except that, when a node in a parallel cluster needs to access a disk page, it:

1) First checks to see if the page is in its local buffer pool (“cache”)
2) If not, checks to see if the page is in the cache of any other node in the cluster
3) If not, reads the page from disk

Such a cache appears to work fairly well on OLTP, but has big problems with data warehousing workloads. The problem with the shared-cache design is that cache hits are unlikely to happen, since warehouse queries are typically answered through sequential scans of the fact table (or via materialized views.) Unless the whole fact table fits in the aggregate memory of the cluster, sequential scans do not typically benefit from large amounts of cache, thus placing the entire burden of answering such queries on the disk subsystem. As a result, a shared cache just creates overhead and limits scalability.

In addition, the same scalability problems that exist in the shared memory model also occur in the shared-disk architecture: the bus between the disks and the processors will likely become a bottleneck, and resource contention for certain disk blocks, particularly as the number of CPUs increases, can be a problem. To reduce bus contention, customers frequently configure their large clusters with many Fibre channel controllers (disk buses), but this complicates system design because now administrators must partition data across the disks attached to the different controllers.

As a result, there are fundamental scalability limits to any database system based on a shared-disk or shared-cache model.

Shared Nothing**:** In a shared-nothing approach, by contrast, each processor has its own set of disks. Data is "horizontally partitioned" across nodes, such that each node has a subset of the rows from each table in the database. Each node is then responsible for processing only the rows on its own disks. Such architectures are especially well suited to the star schema queries present in data warehouse workloads, as only a very limited amount of communication bandwidth is required to join one or more (typically small) dimension tables with the (typically much larger) fact table.

In addition, every node maintains its own lock table and buffer pool, eliminating the need for complicated locking and software or hardware consistency mechanisms. Because shared nothing does not typically have nearly as severe bus or resource contention as shared-memory or shared-disk machines, shared nothing can be made to scale to hundreds or even thousands of machines. Because of this, it is generally regarded as the best-scaling architecture [4].

Shared-nothing clusters also can be constructed using very low-cost commodity PCs and networking hardware – as Google, Amazon, Yahoo, and MSN have all demonstrated. For example, Google's search clusters reportedly consist of tens of thousands of shared-nothing nodes, each costing around $700. Such clusters of PCs are frequently termed "grid computers."

In summary, shared nothing dominates shared disk, which in turn dominates shared memory, in terms of scalability.

Today's data warehouse databases can be grouped by the parallelism approach they take. In the first class, the least scalable one, are DBMSs that run only on the oldest shared-memory architectures. Vendors of these systems have not expended the substantial effort to extend their software to either of the other two newer architectures. In the second class are systems that began life using shared memory and were subsequently extended to run on a shared-disk architecture. However, systems in this class have not been modified to take advantage of the most scalable shared-nothing architecture. The third class of systems runs natively on shared-nothing configurations. The second class is more scalable than the first class, but the third class is much more scalable than the second class. Table 1 summarizes the leading commercial vendors in each class.

Shared Memory (least scalable)	***Shared Disk (medium scalable)***	***Shared Nothing (most scalable)***
Microsoft SQL Server	Oracle RAC	Teradata
PostgreSQL	Sybase IQ	Netezza
MySQL		IBM DB2
		EnterpriseDB
		Greenplum
		Vertica

Table 1: Parallelism approaches taken by different data warehouse DBMS vendors.

Going Even Faster: Hardware Acceleration

The best scalability is available from shared-nothing architectures, and there are two flavors of shared-nothing hardware. The first flavor is a grid configuration from IBM, Hewlett-Packard or other vendor, who assembles a collection of single-board computers into a grid and packages them in a rack or cabinet. Low-end grid systems (i.e., a single-core CPU with a low-end disk) cost about $700 per node. This is the technology employed by most of the large e-commerce sites. The second flavor is a grid configuration using high-end grid cells (e.g., a dual-core, dual-CPU Opteron with multiple disks), which cost more than $2,000. In either case, you are opting for commodity parts, assembled into a large computing complex.

The alternative to these commodity shared-nothing systems is to purchase a specialized database appliance from Teradata, Netezza or DATAllegro. Currently the nodes in such appliances vary in price dramatically, but a typical price is $10K, for a $700 PC plus a modest amount of proprietary hardware. A simple calculation indicates that there must be 14X acceleration from such appliances just to break even against the alternative of using commodity parts. There is no evidence that any previous or current hardware database company, from Britton Lee in the early 1980's to today's companies, has achieved or achieves 14X acceleration. As such, hardware acceleration has never been shown to be sufficiently good to compete against the steamroller of commodity parts. For a more detailed discussion of this point, see [5].

In summary, hardware appliances offer good scalability, but are likely to result in a long-term guided tour through your wallet. It's simply a much better deal to stick to shared-nothing DBMSs that run on commodity parts.

Achieving Scalability Through Software

The above discussion makes it clear that the data warehouse market is in need of technology that offers better scalability without resorting to custom hardware. Fortunately, there are two software tactics that offer the possibility of dramatically better performance. This section discusses these tactics, which can be used individually or together.

Vertical partitioning via column-oriented database architectures [6]: Existing shared-nothing databases partition data "horizontally" by distributing the rows of each table across both multiple nodes and multiple disks on each node. Recent research has focused on an interesting alternative: partitioning data vertically so that different columns in a table are stored in different files. While still providing an SQL interface to users, these "column-oriented" databases, particularly when coupled with horizontal partitioning in a shared-nothing architecture, offer tremendous performance advantages.

For example, in a typical data warehouse query that accesses only a few columns from each table, the DBMS need only read the desired columns from disk, ignoring the other columns that do not appear in the query. In contrast, a conventional, row-oriented DBMS must read all columns whether they are used in the query or not. In round numbers, this will mean that a

column store reads 10 to 100 times less data from disk, resulting in a dramatic performance advantage relative to a row store, both running on the same shared-nothing commodity hardware.

Of the systems mentioned in the above table, only SybaseIQ and Vertica are column stores. However, Vertica is alone in employing the more scalable shared-nothing architecture.

Compression-aware databases [1]: It is clear to any observer of the computing scene that CPUs are getting faster at an incredible rate. Moreover, CPUs will increasingly come packaged with multiple cores, possibly 10 or more, in the near future. Hence, the cost of computation is plummeting. In contrast, disks are getting much bigger and much cheaper in cost per byte, but they are not getting any faster in terms of the bandwidth between disk and main memory. Hence, the cost of moving a byte from disk to main memory is getting increasingly expensive, relative to the cost of processing that byte. This suggests that it would be smart to trade some of the cheap resource (CPU) to save the expensive resource (disk bandwidth). The clear way to do this is through data compression.

A multitude of compression approaches, each tailored to a specific type and representation of data, have been developed, and there are new database designs that incorporate these compression techniques throughout query execution. In round numbers, a system that uses compression will yield a database that is one third the size (and that needs one third the disks). More importantly, only one-third the number of bytes will be brought into main memory, compared to a system that uses no compression. This will result in dramatically better I/O performance.

However, there are two additional points to note. First, some systems, such as Oracle and SybaseIQ, store compressed data on the disk, but decompress it immediately when it is brought into main memory. Other systems, notably Vertica, do not decompress the data until it must be delivered to the user. An execution engine that runs on compressed data is dramatically more efficient than a conventional one that doesn't run on compressed data. The former accesses less data from main memory, and copies and/or writes less data to main memory, resulting in better L2 cache performance and fewer reads and writes to main memory.

Second, a column store can compress data more effectively than a row store. The reason is that every data element on a disk block comes from a single column, and therefore is of the same data type. Hence, a column-based database execution engine only has to compress elements of a single data type, rather than elements from many data types, resulting in a three-fold improvement in compression over row-based database execution engines.

What You Can Do

The message to be taken away from this article is straightforward: You can obtain a scalable database system with high performance at low cost by using the following tactics.

1) Use a shared-nothing architecture. Anything else will be much less scalable.

2) Build your architecture from commodity parts. There is no reason why the cost of a grid should exceed $700 per (CPU, disk) pair. If you are paying more, then you are offering a vendor a guided tour through your wallet.

3) Get a DBMS with compression. This is a good idea today, and will become an even better idea tomorrow. It offers about a factor of three performance improvement.

4) Use a column-store database. These are 10 to 100 times faster than a row-store database on star-schema warehouse queries.

5) Make sure your column-store database has an executor that runs on compressed data. Otherwise, your CPU costs can be an order of magnitude or more higher than in a traditional database.

To the extent that you are using fewer than all five tactics, you are either paying too much or unnecessarily limiting your scalability. We would encourage you to grade your current DBMS warehouse environment by giving it one point for each of the five requirements it meets above. There is at least three orders of magnitude difference in price/performance between a score of 0 and a score of 5. Hence, you can see how far off you are from what is possible.

References

[1] Daniel Abadi, Samuel Madden, and Miguel Ferreira. "Integrating Compression and Execution in Column-Oriented Database Systems." *Proceedings of SIGMOD,* 2006.

[2] Thomas Anderson, David Culler, and David Patterson. "A Case for Networks of Workstations: NOW." *IEEE Micro*, February 1995.

[3] Jeffery Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." *Proceedings of OSDI*, 2004.

[4] David J. DeWitt and Jim Gray. "Parallel Database Systems: The Future of High Performance Database Processing." *Communication of the ACM*, Vol 35(6), pp. 85-98, June 1992.

[5] Samuel Madden, "Rethinking Database Appliances." *DM Review,* scheduled for publication November 3, 2006.

[6] Michael Stonebraker et al. "C-Store: A Column Oriented DBMS." *Proceedings of VLDB*, 2005.

==========

David J. DeWitt, Ph.D., is currently the John P. Morgridge Professor of Computer Sciences at the University of Wisconsin, where he has taught and done research since 1976. His research has focused on the design and implementation of database management systems including parallel, object-oriented, and object-relational database systems. The Gamma parallel database system project, developed in the late 1980s, produced many of key pieces of technology that form the basis for today's large parallel database systems, including products from IBM, Informix, NCR/Teradata, and Oracle. Dr. DeWitt also developed the first relational database

system benchmark in the early 1980s, which became known as the Wisconsin benchmark. More recently, his research has focused on the design and implementation of distributed database techniques for executing complex queries against Internet content. Dr. DeWitt also is an advisor to various technology companies, including Vertica Systems, Inc. (www.vertica.com).

Samuel Madden, Ph.D., is an assistant professor in the EECS department at MIT and a member of MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL). His research interests span all areas of database systems; past projects include the TinyDB system for data collection from sensor networks and the Telegraph adaptive query processing engine. His current research focuses on modeling and statistical techniques for value prediction and outlier detection in sensor networks, high performance database systems, and networking and data processing in disconnected environments. Dr. Madden also is an advisor to various technology companies, including Vertica Systems, Inc. (www.vertica.com).

Michael Stonebraker, Ph.D., has been a pioneer of database technology and research for more than a quarter of a century. Dr. Stonebraker was the main architect of the INGRES relational DBMS; the object-relational DBMS POSTGRES; and the federated data system Mariposa. All three prototypes were developed at the University of California at Berkeley, where Dr. Stonebraker was a professor of computer science for 25 years. More recently, at MIT, Dr. Stonebraker was a co-architect of the Aurora stream-processing engine as well as the C-Store high-performance read-oriented database engine. He is the founder of four venture-capital-backed startups that have commercialized these prototypes: Ingres Corporation, Illustra Information Technologies (acquired by Informix Corporation), StreamBase Systems, and Vertica Systems.