VOLTDB
SMART DATA FAST.™

TECHNICAL OVERVIEW

HIGH PERFORMANCE, SCALE-OUT RDBMS FOR FAST DATA APPS REQUIRING REAL-TIME ANALYTICS WITH TRANSACTIONS

Overview

VoltDB is a fast in-memory relational database system (RDBMS) for high-throughput, operational applications requiring:

- Orders of magnitude better performance than a conventional RDBMS – 100x faster
- SQL as the native data language, simplifying development and enabling richer interaction
- ACID transactions to ensure data consistency, integrity and always-accurate results
- Built-in high availability and fault tolerance
- Cross-datacenter replication for HA and disaster recovery in geographically-distributed clusters
- Geospatial query support for location-based apps

For fast, data-driven applications, VoltDB offers substantial performance and cost advantages as illustrated by the results of the TPC-C-like benchmark below, in which VoltDB and a well-known OLTP DBMS were compared running the same test on identical hardware (Dell R610, 2x 2.66Ghz Quad-Core Xeon 5550 with 12x 4GB (48GB) DDR3-1333 Registered ECC DIMMs, 3x72GB 15K RPM 2.5in Enterprise SAS 6GBPS Drives):

	Nodes	VoltDB	DBMSx	VoltDB Advantage
"TPC-C-like" workload (VoltDB lab)	1	53,000 TPS	1,555 TPS	45x better throughput
	12	560,000 TPS	NA	Near-linear (.9) scaling

As shown above, VoltDB's massively parallel architecture also provides elastic scalability running on low-cost commodity servers. It allows application developers to scale applications simply by adding servers to a VoltDB cluster, instead of building complex, costly sharding layers. VoltDB's ACID compliance ensures developers never need to sacrifice data consistency to gain high performance or scalability.

This white paper is for technical readers, and explains:

- **The reasons why traditional databases are difficult and expensive to scale**
- **The "scale-out" VoltDB architecture and what makes it different**
- **VoltDB application design considerations**

RDBMS Scaling Alternatives and Costs

As an application's popularity and usage increases, scaling it up to support heavier operational workloads and run 24x7x365 can be painful. The scalability bottleneck is often the DBMS.

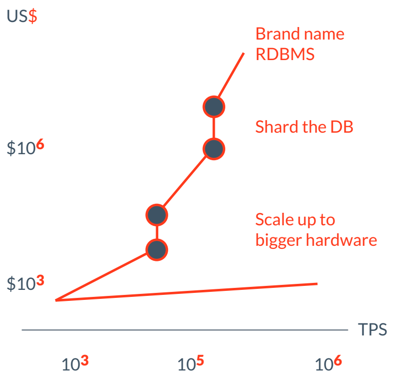
To adapt to heavier workloads, an application may experience a number of disruptive scaling events during its lifecycle, including:

- Migration from an inexpensive commodity server to a very expensive SMP server
- Re-designing the database (and corresponding application data access logic)
- Implementing a data partitioning or "sharding" scheme—manually dividing the database into many smaller databases running on different servers and modifying application code to coordinate data access across the partitions
- Implementing a key-value store (KV store), thereby forfeiting transactional consistency and the ability to use SQL
- Migration from an open source DBMS to a higher-scale commercial DBMS such as Oracle

Dealing with a fast-growing user workload is a good problem to have, but the popular "scale-up" approaches to scalability are costly, requiring expensive hardware and DBMS upgrades. Scale-up approaches also add development complexity, and increase overhead and maintenance costs. And no matter what scheme is used, the opportunity cost is also high—time spent fixing performance problems means less time spent implementing higher-value business functionality.

What's needed is a DBMS that "scales out" linearly and limitlessly by adding new commodity servers to a shared-nothing DBMS cluster. VoltDB is exactly that DBMS.

RDBMS Scaling Alternatives and Costs



Why Traditional DBMSs have Difficulty Scaling

In 2008, a team of researchers led by Mike Stonebraker published a seminal paper in the ACM SIGMOD entitled "*OLTP Through the Looking Glass, and What We Found There*." The paper exposed key sources of processing overhead that plague traditional disk-based RDBMS products, and concluded that removing those overheads and running the database in main memory would yield orders of magnitude improvements in database performance. Along with other noteworthy research of that period, Through the Looking Glass signaled a sea change in thinking around in-memory operational systems.

Traditional DBMSs have five sources of processing overhead:

Index Management: B-tree, hash and other indexing schemes require significant CPU and I/O.

Write-ahead Logging: Traditional databases write everything twice; once to the database and once to the log. Moreover, the log must be forced to disk to guarantee transaction durability. Logging is, therefore, an expensive operation.

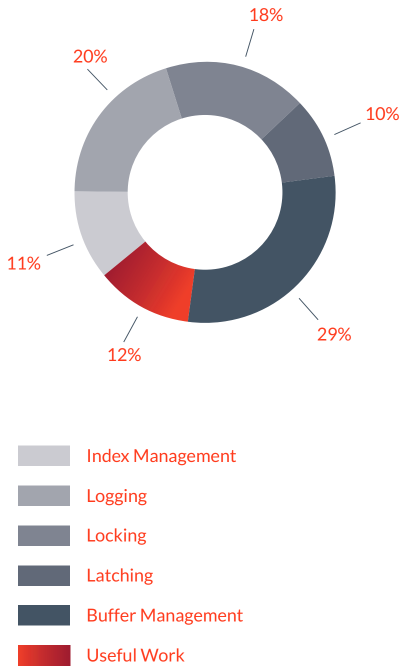
Locking: Before touching a record, a transaction must set a lock on it in the lock table. This is an overhead-intensive operation.

Latching: Updates to shared data structures (B-trees, the lock table, resource tables, etc.) must be done carefully in a multi-threaded environment. Typically, this is done with short-duration latches, which are another considerable source of overhead.

Buffer Management: Data in traditional systems is stored on fixed-size disk pages. A buffer pool manages which set of disk pages is cached in memory at any given time. Moreover, records must be located on pages and the field boundaries identified. Again, these operations are overhead-intensive.

Designed originally for data integrity, this overhead prevents traditional databases from scaling to meet contemporary data volumes and workloads.

General Purpose RDBMS Processing Profile



VoltDB Eliminates All of these Legacy DBMS Overheads. With VoltDB:

- Data and the processing associated with it are partitioned together and distributed across the CPU cores ("virtual nodes") in a shared-nothing hardware cluster
- Data is held in memory for maximum throughput and to eliminate the need for buffer management
- Each single-threaded partition operates autonomously, eliminating the need for locking and latching
- Data is automatically replicated for intra- and inter-cluster high availability

VoltDB Architecture

VoltDB leverages the following architectural elements to achieve its performance, scaling and high availability objectives:

- Automatic partitioning (sharding) across a shared-nothing server cluster
- Main-memory data architecture
- Elimination of multi-threading and locking overhead
- Automatic replication and command logging for high availability and durability
- Ad-hoc SQL access via JDBC and client language bindings
- Multi-statement transactions via Stored Procedures

Automatic Partitioning Across Shared-nothing Server Clusters

VoltDB embraces shared-nothing architecture. To achieve database parallelism, data and the processing associated with it (in the form of a single-threaded VoltDB execution "engine") are distributed among all the CPU cores within the servers composing a single VoltDB cluster. By extending its shared-nothing foundation to the per-core level ("virtual nodes"), VoltDB exploits and scales with the increasing core-per-CPU counts on modern commodity servers. This architecture also allows VoltDB to run easily in virtualized and cloud infrastructures.

Every VoltDB table is either:

- Partitioned across these virtual nodes, or
- Cloned at every virtual node

Partitioning is appropriate for large, frequently updated or accessed tables, while cloning of small read-almost-always tables can significantly improve performance.

Main Memory Data Architecture

The size of databases is increasing at the rate that business expands. For example, a purchase order database increases in size at the rate that active purchase orders increase. In almost all database applications, this rate is slower than the rate at which main memory decreases in price. Therefore almost all database applications are (or will soon be) candidates for main-memory deployment.

VoltDB is a main memory DBMS.

In VoltDB, each partition is stored in main memory and processed by its associated, single-threaded execution engine at in-memory speed. In-memory processing eliminates disk waits from within VoltDB transactions, along with the need for buffer management overhead.

VoltDB can save data snapshots and command logs to disk for backup and recovery purposes and also spool data to a data warehouse database for analysis and querying. This feature is explained in more detail below.

Elimination of Multi-threading and Locking Overhead

Conventional databases experience disk and user stalls within transactions. Rather than letting the CPU idle during stalls, those DBMSs interleave SQL execution from multiple transactions during waits so the CPU is always busy. This requires much complex latching and locking overhead.

VoltDB doesn't experience user stalls (since transactions happen within stored procedures) or disk stalls (because VoltDB processes data in main memory). Therefore, VoltDB is able to eliminate the overhead associated with multi-threading (latching) and locking. Each VoltDB execution engine is single-threaded and contains a queue of transaction requests, which it executes sequentially—and exclusively—against its data. Elimination of stalls and associated locking and latching overhead allows typical VoltDB SQL operations to complete in microseconds.

For single-partition transactions, each VoltDB engine operates autonomously. For multi-partition transactions, one engine distributes and coordinates work plans for the other engines. VoltDB assumes that an application designer can construct a partitioning or cloning scheme and transaction design that localizes a large majority of transactions to a single virtual node. Many fast data applications such as telco billing, personalization, game state, sensor management, and capital market risk share this profile.

Automatic Replication and Recovery for High Availability

VoltDB achieves high availability for 24x7x365 operations very simply and economically through automatic intra-cluster and inter-cluster replication. Data is synchronously committed to replicated partitions within the cluster before transactions commit. This provides durability against single-node failures. To provide durability against cluster failures (e.g., in the event of a data center catastrophe), transactions are asynchronously committed to a replica cluster (usually in a different geographic location).

VoltDB automatically guarantees that every replica, whether in the same cluster or a different one, runs transactions in the same order and thereby achieves the same global order.

VoltDB runs an active-active configuration within a cluster. If a node failure occurs, VoltDB automatically (and seamlessly) switches over to a replica node. Hence, an application is unaware that a problem occurred. Recovery is performed automatically; the recovered node (or cluster) queries the running VoltDB cluster to recover its data.

VoltDB's passive database replication feature, parallel binary replication between the master and replica clusters, has been augmented with geo-distributed, cross-datacenter replication (XDCR). This bidirectional database replication, also called active-active replication, allows enterprises to maintain separate, active copies of the database in two separate locations.

VoltDB implements a concept called command logging for transaction-level durability. Unlike traditional write-ahead logs, VoltDB logs the instantiation of commands to the database rather than all resulting actions. This style of logging greatly reduces the load on the disk system while providing either synchronous or asynchronous logging for transaction-level durability.

Interfaces for ACID Transactions

VoltDB exposes stored procedure, ad hoc and JDBC interfaces. With stored procedures, only a single round trip occurs between the client and the server per transaction. Hence, the run-time interface to VoltDB is to execute a stored procedure, substituting transaction-specific constants for parameters.

Stored procedures are written in Java, with embedded SQL calls for database services. VoltDB supports ANSI-standard SQL with application-enhancing extensions.

Geospatial SQL Support

VoltDB also provides support for geospatial SQL queries. Geospatial applications deal with polygons, a set of latitude/longitude points that define a region (perhaps the boundary of a shopping mall, a sports arena, or a neighborhood or city), and points, which are a latitude/longitude pair defining a specific location (of, say, a person or device). VoltDB SQL supports two new datatypes to represent polygons and points: GEOGRAPHY and GEOGRAPHY_POINT, and provides the ability to perform queries on the relationships between them, such as 'contains' or 'distance'. Thus VoltDB v6 has added a set of geospatial column functions, including CONTAINS, DISTANCE, and AREA, among others.

+
-
Pueto Street
Industrial Drive
Chalan Pale Arnold
Pueto Street
Pantalan Circle
Fabot Way
Isla Drive
Publiko Drive
Part Ida Drive
Gloria Drive
Delegado Drive
Micro Beach Road
Coral Tree Avenue
Royal Palm Avenue
Beach Road
Cactus Street
Tamantigi Street
Orchid Street
Beach Road
Hospital Drive

Leaflet | Map data © OpenStreetMap contributors

VoltDB Versus Other DBMS Alternatives

VoltDB isn't the first attempt at overcoming the performance and scalability limitations of traditional databases. Two alternatives to VoltDB include running conventional databases in memory, or using a "NoSQL" key-value store (KV store).

Alternative in-memory systems may remove buffer management and logging (at the expense of durability). Even with these features removed, the maximum performance improvement is roughly 2x. To achieve the 50-100x speedup of VoltDB, all legacy OLTP time-syncs must be removed (buffer management, logging, latching and locking).

To deliver better performance on scale-out hardware, some databases, such as NoSQL KV stores, eliminate some of this overhead — and sacrifice SQL interactivity and data integrity as well (delivering "eventual consistency"). Unfortunately, since KV stores don't execute SQL (although some NoSQL offerings have begun to provide limited, proprietary SQL alternatives), functionality that would normally be executed by the database must be implemented in the application layer.

The table below summarizes the differences between VoltDB, NoSQL KV stores and traditional databases:

	VoltDB	NoSQL	Traditional RDBMS
Scale-out architecture	●	●	
Built-in high availability	●	●	
Multi-master replication	●	●	
ACID compliant	●		●
SQL data language	●		●
Cross-partition joins	Automatic	In app code	In app code
Cost at web scale	$	$$$	$$$$

VoltDB and Data Warehousing

VoltDB is specially designed to handle fast, large volumes of operational database transactions. As an in-memory database, VoltDB is not well suited to the petabyte+ data volumes increasingly found in data warehousing. However, organizations often want to store and process the information collected by VoltDB in a data warehouse database. Therefore, VoltDB includes an export integration subsystem that spools VoltDB data to analytic DBMS products (such as columnar data stores and Hadoop) to enable deep reporting and analysis via these tools.

Summary

VoltDB leverages observations derived from research and industry expertise about OLTP workloads to achieve linear scaling as nodes are added to a VoltDB cluster. This partitioning, across multiple machines and multiple cores, is central to the design of VoltDB; it mitigates the contention for resources that limit legacy DBMS scalability. The main-memory storage of all data greatly reduces network and disk latency. Combined, these departures from the status quo allow VoltDB to offer next-generation DBMS scalability, performance and manageability, liberating organizations from expensive shared-memory, shared-disk database systems that do not scale.

Next Steps

To learn more about VoltDB, visit www.voltdb.com. Product documentation, developer support forums and an open source version of VoltDB are freely available.