VoltDB vs. Redis Benchmark

Motivation and Goals of this Evaluation

- Compare the performance of several distributed databases that can be used for state storage in some of our applications
  - Low latency is expected
  - Persistence on disk is not required
- The comparison of the features of these databases is not covered in this presentation
- This is part of a technology investigation, not a product decision
- Popular tools such as YCSB (Yahoo! Cloud Serving Benchmark) focus on typical cloud workloads for key-value stores and NoSQL databases: access by primary key, no locking.
- But we want to compare the databases under a more demanding scenario…

Data Record Operation Used in this Test

Our test scenario:

- Requires strict locking (opportunistic concurrency control is not allowed);
- Makes most database requests using secondary keys (a.k.a. alternate keys) instead of the primary key for each record. Partitioning is done on the primary key, so the secondary keys do not match the partitioning scheme.

The following sequence (operation) is repeated by all test clients:

1. Select a secondary key
2. "LockAndRead": Lock and read the record associated with that secondary key
3. "Wait": Simulate the interaction with external nodes (delay from 1 to 5ms) to update the data
4. "WriteAndUnlock": Update the modified record in the database and release the lock
5. Wait (1ms to 5ms) before accessing the same record again 4 times, then another record

The time that we measure for each "operation" covers steps 1 to 4, including the "wait" of ~3ms.

Test Setup and Test Procedure

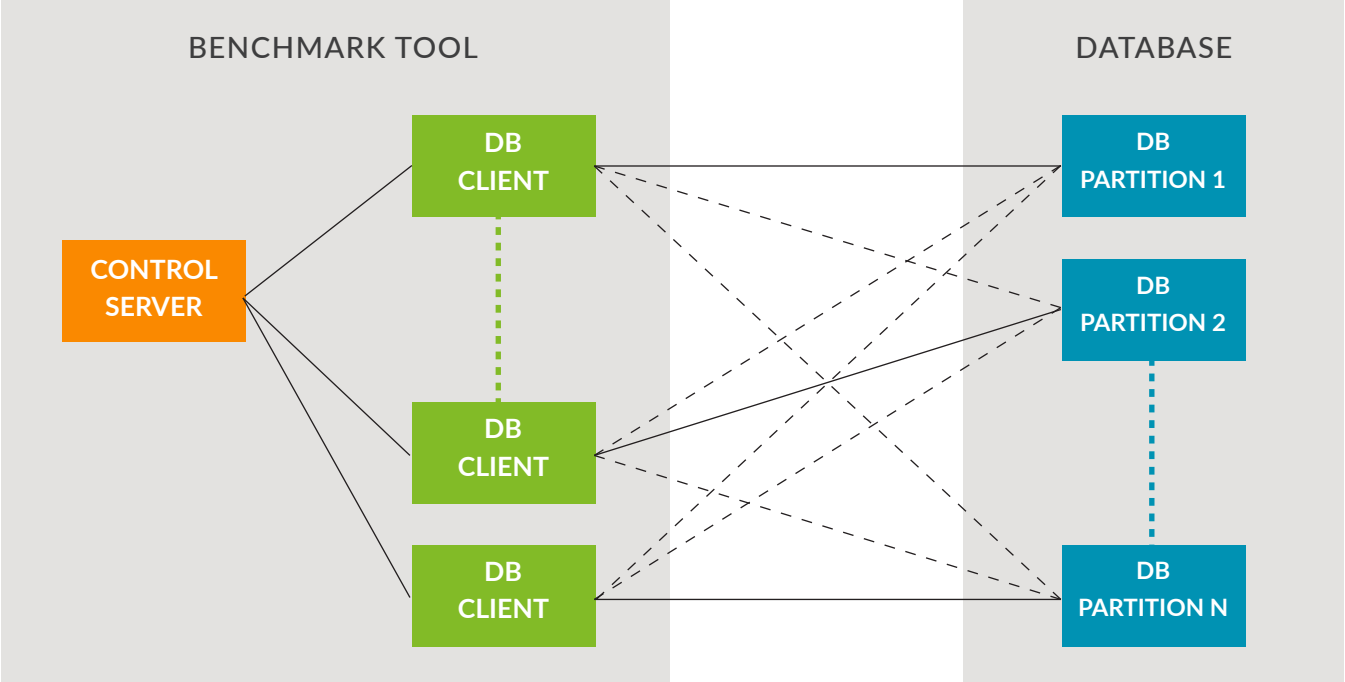


Figure 1

Test Setup

3 client hosts and 3 server hosts, HP DL360 Gen6

- 2 quad-core CPUs: Intel Xeon E5540 @ 2.53GHz for a total of 16 hyperthreads (8 real cores)
- 72 GB RAM
- 2 * 10 Gbit + 2 * 1 Gbit Ethernet adapters, although most traffic goes through the 1st adapter
- connection to a dedicated switch (full non-blocking 10 Gbit)

Test Procedure

- The control server launches several DB clients that measure several statistics and generate increasing load on the database until too many errors occur
- Many combinations of parameters are tested: number of clients, payload size, burstiness of the requests, rate of load increase, etc.
- Databases: VoltDB (Enterprise and Community editions), Redis and others.

Technical Procedure and Results

Test procedure and results (1/2)

- The control server launches several DB client processes on the client hosts
- Each DB client generates requests to the DB and measures several statistics that are reported to the control server
- The load is progressively increased by all clients until too many errors occur

Test procedure and results (2/2)

- Each test is repeated 30 to 90 times over 6-12 hours, increasing the number of clients and the payload size:
  - 1 to 21 clients (21 clients = 7 per host; hosts have 8 physical cores)
  - 100 to 4000 bytes of payload
  - Requests sent in bursts of 10ms, 20ms, 50ms or more
- Graphs are generated after each test
  - Around 130 to 300 graphs per series of tests
  - Available on request if you are interested

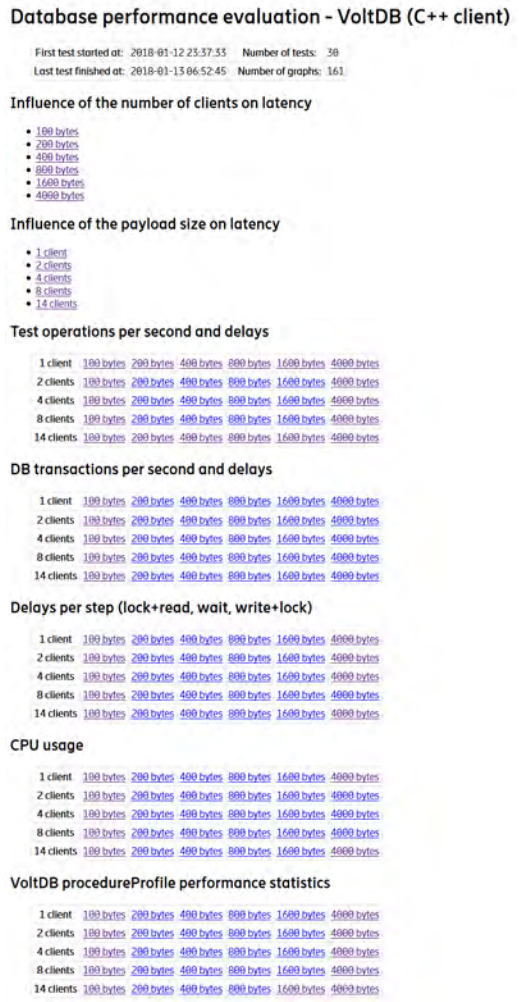


Figure 2

Databases Covered in this Report

- VoltDB, clients in C++ and Go
  - Enterprise edition and Community edition — We did not use command logging or other features
  - Several partitioning setups were tested — 14 sites per host (42 partitions) seemed to give the best results
- Redis, clients in C and Go
- Several other databases not disclosed here, with clients in C++, Java or Go
  - 8 clients in total, plus 2 from older tests

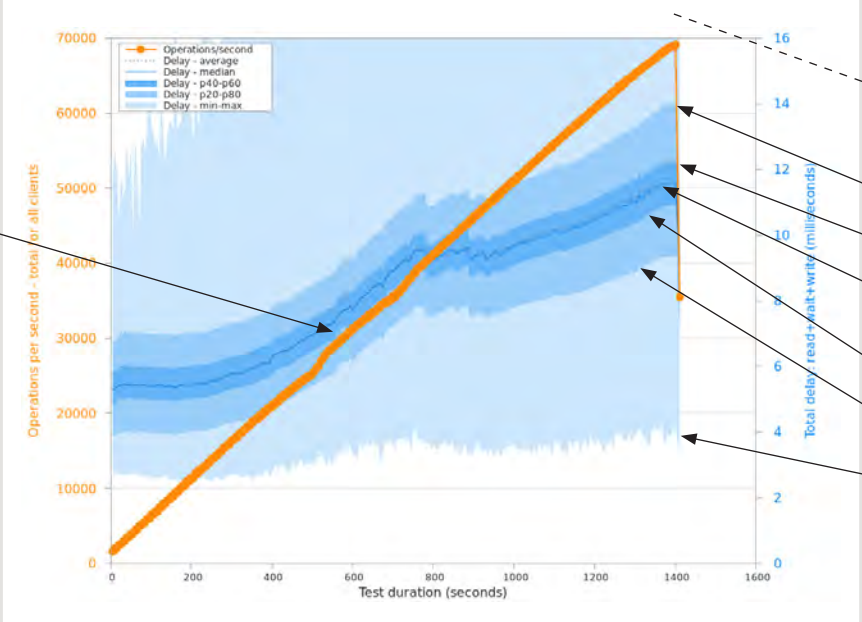
Understanding the Throughput/Latency Results

Left Y axis =Throughput

Orange line + dots = total number of operations per second for all clients, increasing over time

One "operation" =
- lock + read,
- wait,
- write + unlock

= several DB requests



Right Y axis =Total delay

Maximum delay out of range because graph is scaled on 90th percentile

80th percentile of delays

60th percentile of delays

Solid line = median = 50th percentile of delays

40th percentile of delays

20th percentile of delays

Minimum delay

Figure 3

Test Results — VoltDB

VoltDB Results, No Replication — 6 clients

- Test done without replication in the database
- 6 C++ clients
- The total delay (including average 3 ms "wait" step) is rather stable as the load increases
- An analysis of the CPU load shows that the client side is the limit, so we need more clients

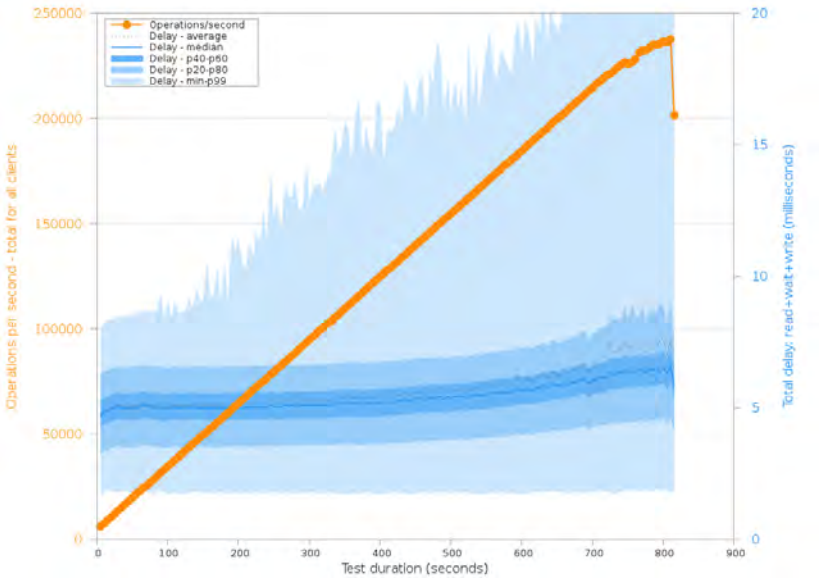


Figure 4: Ops/s and delay for 6 clients on 3 hosts, 100 bytes payload — VoltDB 14 sites per host

VoltDB Results, No Replication — 18 clients

- Test done without replication in the database
- 18 C++ clients
- Latency remains good until about 225K ops/s
- Maximum throughput around 350K ops/s (800K database transactions per second) when the average latency exceeds 20 ms

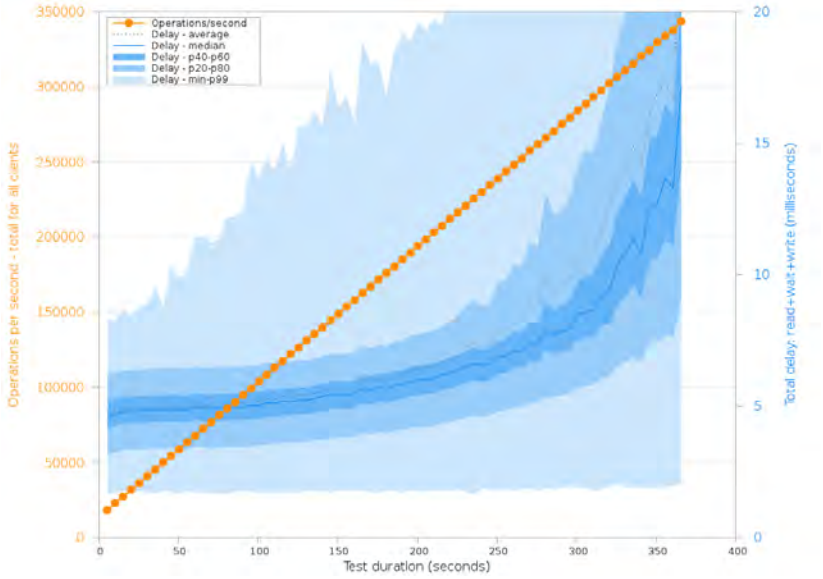


Figure 5: Ops/s and delay for 18 clients on 3 hosts, 100 bytes per payload — VoltDB 14 sites per host

VoltDB Results, No Replication — N clients

- With 3 or 6 clients, the limits in our tests come from the client side (maximum load on a few CPU cores)
- The results are very similar if the load comes from 9 or 21 clients:
- The maximum throughput is slightly below 350K ops/s
- Until around 225K ops/s, the average latency remains under 7ms ("comfort zone")

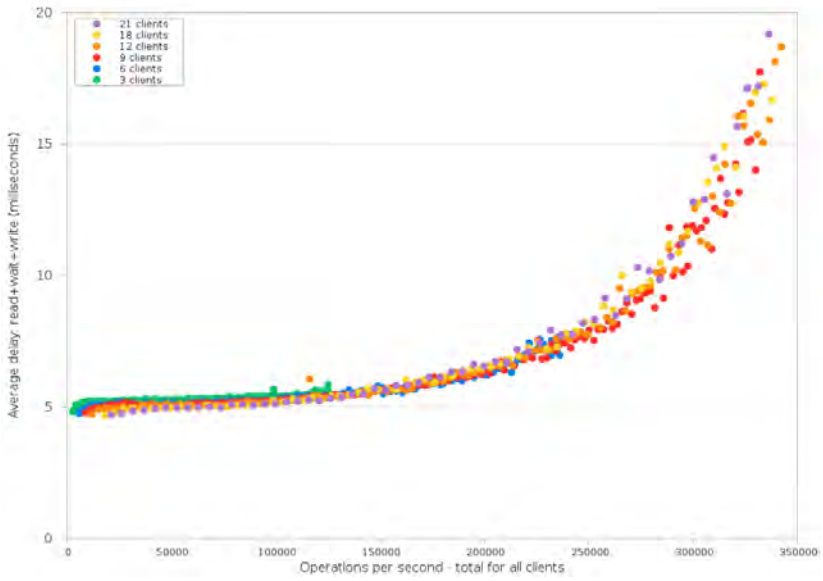


Figure 6: Influence of the number of clients on latency (payload: 100 bytes)

VoltDB Results, with Replication — 18 clients

- Test done with replication factor 1 in the database
- 18 C++ clients
- Latency remains good until about 73K ops/s
- Maximum throughput around 105K ops/s (240K database transactions per second) when average latency exceeds 20 ms

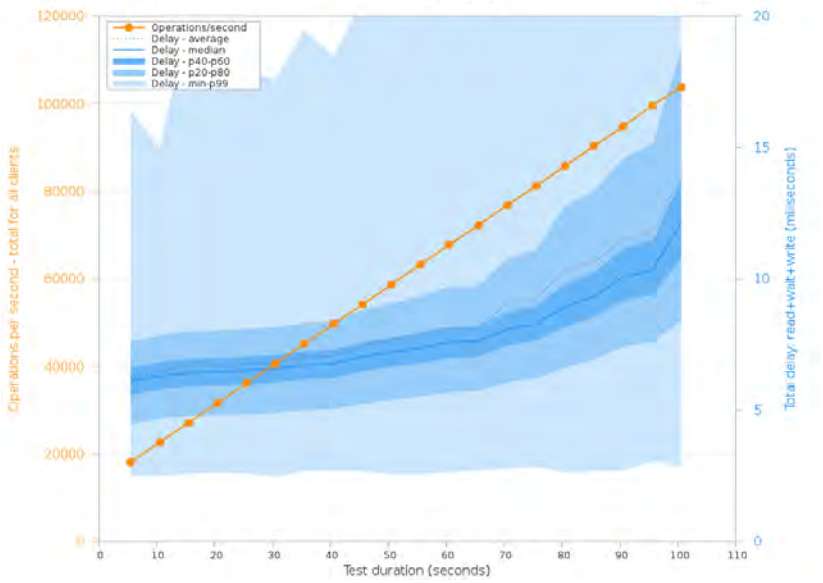


Figure 7: Ops/s and delay for 18 clients on 3 hosts, 100 bytes payload — VoltDB (C++) repl, 14x3 sites

VoltDB Results, with Replication– N clients

- Test done with replication factor 1 in the database
- The results are very similar regardless of the number of clients:
  - The maximum throughput is slightly above 120K ops/s
  - Until around 70K ops/s, the average latency remains under 7ms
- The limit is the server CPU even with 3 clients

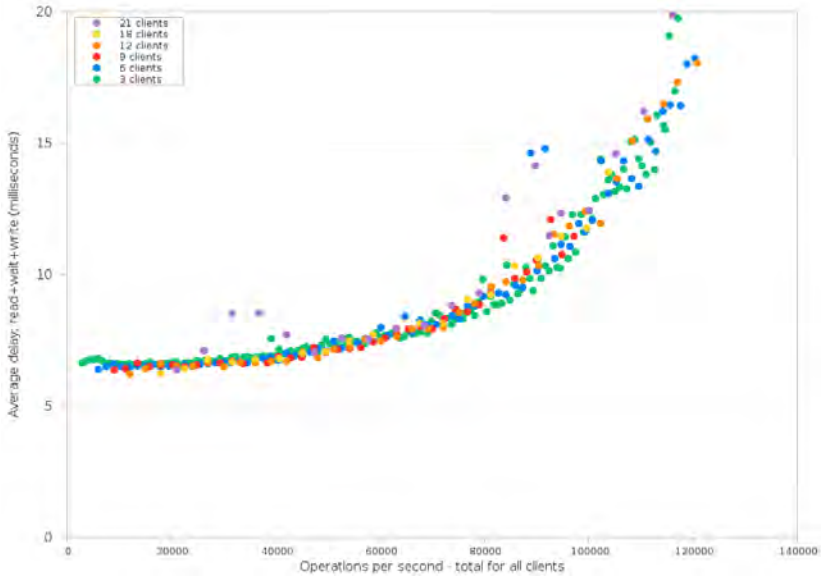


Figure 8: Influence of the number of clients on latency (payload: 100 bytes)

Comparison with Other Databases

Redis Results, with Replication — N clients

- Replication factor 1, asynchronous
- Redis client written in Go
- Similar results for C client
- Latency lower than VoltDB but uses asynchronous replication: risk of inconsistencies in case of node or network failures

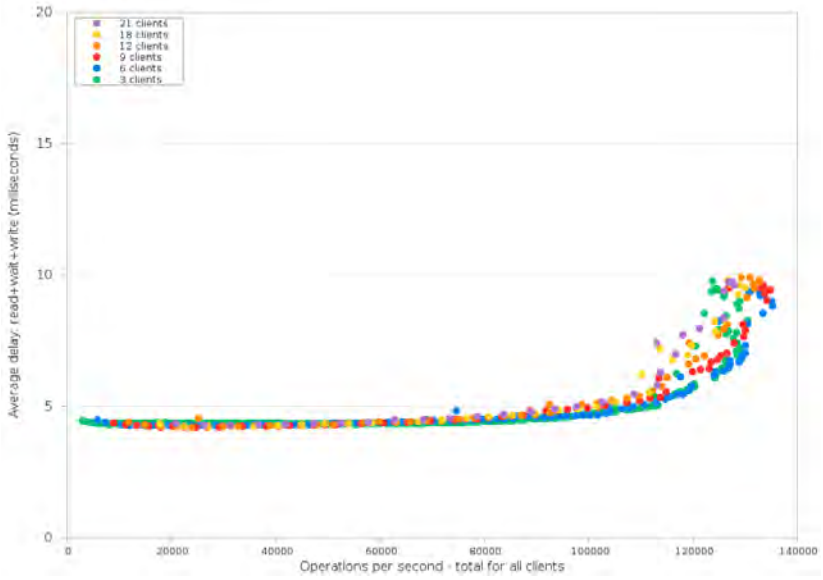


Figure 9: Influence of the number of clients on latency (payload: 100 bytes)

Database NNN, with Replication — N clients

- Replication factor 1, synchronous
- Client written in Go
- Maximum throughput is 20-25% lower than VoltDB or Redis
- Latency increases too quickly and is already above 10ms for a moderate load

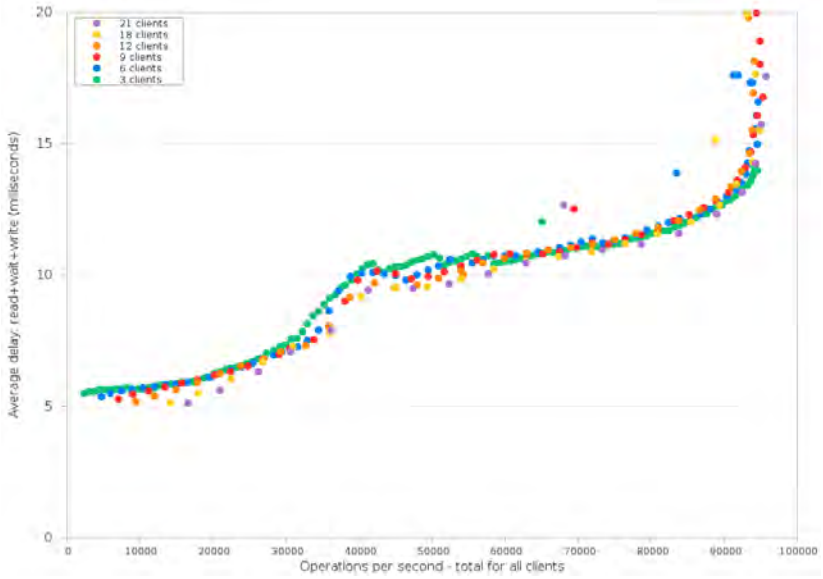


Figure 10: Influence of the number of clients on latency (payload: 100 bytes)

Technical Summary

- VoltDB performance:
  - Throughput comparable to Redis when replication is enabled, better without replication
  - Better than the other databases that we tested
  - Replication has a significant influence on the performance: throughput divided by 3 approx.
  - Synchronous replication is slower but safer than asynchronous replication. It reduces the risk of inconsistencies in case of network failures.
- Client libraries are available for Java, C++, Go, ...
  - Slightly different programming models used for each language (e.g., Java and Go rely on threads, C++ uses a single thread for I/O)

Executive Summary

- Good support received from VoltDB
- For all databases studied, the best results are obtained by adapting slightly the business logic of the application to take advantage of what each database does best: stored procedures, lightweight transactions, different partitioning models, etc.
- Comparison of features is out of scope for this presentation: replication, scalability, transaction capabilities, management and monitoring, recovery after failure, etc.

- VoltDB achieved performance and scale without sacrificing consistency
- Distributed data and computing for scaling performance, linear to allocated resources (CPU cores)
- True high availability with no data loss upon faults and failures within the cluster
- Synchronous durability to prevent data loss upon full cluster failure
- Disaster recovery for establishing failover clusters
- Cross DataCenter Replication (XDCR) for routing geo-local traffic with conflict resolution/notification
- Stored Procedures that allow complex logic to be implemented or imported in a simple Java class construct in less than 100 lines of code allowing ease of maintenance
- Co-locating code and data allows scale guaranteeing predictably low latency for a connected API driven data flow
- Import and export with pre-built connectors and extensibility for custom connectors
- Variety of client libraries (Java, C++, Golang etc.)
- Non-stop availability strategies throughout planned upgrade and maintenance activities
- Virtualization and Containerization ready
- Great support and a partner approach based engineering team

---

About VoltDB

VoltDB is the only in-memory transactional database for modern applications that require an unprecedented combination of data scale, volume, and accuracy. Unlike other databases, including OLTP, Big Data, and NoSQL, that force users to compromise, only VoltDB supports all three modern application data requirements: **1. Millions** — VoltDB processes a relentless volume of data points from users and data sources. **2. Milliseconds** — VoltDB ingests, analyzes, and acts on data in less than the blink of an eye. **3. 100%** — Data managed by VoltDB is always accurate, all the time, for all decisions. Telcos, Financial services, Ad Tech, Gaming, and other companies use VoltDB to modernize their applications. VoltDB is preparing energy, industrial, telco and other companies to meet the challenges of the IoT. VoltDB was founded by a team of world-class database experts, including Dr. Michael Stonebraker, winner of the coveted ACM Turing award.

5April2018

VOLTDB