VoltDB The NewSQL database for high velocity applications

VoltDB and Node.js

695K TPS on Amazon Cloud

Andy Wilson, VoltDB, Inc.
awilson@voltdb.com
www.voltdb.com

@voltdb

June 2012

QCon

Context

- On April 17, 2012, VoltDB announced benchmark results for our Node.js client driver.
- Andy Wilson is the primary maintainer of VoltDB's Node.js driver. Henning Diedrich, Founder of Eonblast, designed and ran the benchmark tests.

Agenda and Format

- Intro to VoltDB and the Node.js client driver
- Discussion of the Node.js/VoltDB benchmark
- Questions & Answers

Intro to VoltDB

Who am I?

- VoltDB Field Engineer, AKA Solutions Architect
- 12 years of developing web applications
  - + Learning management systems
  - + Educational content editorial management
  - + Pharmaceutical sales analytics
  - + Harvard Business Publishing's Higher Education ecommerce site architect
- Using Java, J2EE, JSF/MyFaces/Ajax4JSF, Hibernate, JBoss Seam, Spring, CSS, JavaScript and Google App Engine with Python

What Is VoltDB?

- In-memory relational DBMS
- Ultra-high performance
  - + Millions of ACID TPS
  - + Single-millisecond latencies

> VoltDB is very scalable; it should scale to 120 partitions, 39 servers, and 1.6 million complex transactions per second at over 300 CPU cores.
>
> ***Baron Schwartz***
> ***Chief Performance Architect***
> ***Percona***

- Scale "up" and "out" on commodity gear
  - + Choose a partitioning key, VoltDB does the heavy lifting
- Built-in fault tolerance and crash recovery
- Open source and commercial distros

Who Uses VoltDB?

	Data Source	High-frequency operations	Lower-frequency operations
Financial trade monitoring	Capital markets	Write/index all trades, store tick data	Show consolidated risk across traders
Telco call data record management	Call initiation request	Real-time authorization	Fraud detection/analysis
Website analytics, fraud detection	Inbound HTTP requests	Visitor logging, analysis, alerting	Traffic pattern analytics
Online gaming micro transactions	Online game play	Rank scores: •Defined intervals •Player "bests"	Leaderboard lookups
Digital ad exchange services	Real-time ad trading systems	Match form factor, placement criteria, bid/ask	Report ad performance from exhaust stream
Wireless location-based services	Mobile device location sensor	Location updates, QoS, transactions	Analytics on transactions

Some VoltDB Customers

VoltDB Thesis

- At Scale everything changes
  - + "One-size-fits-all" datastores do not work
- Database Specialization – transactional workloads
  - + H-store academic prototype – http://hstore.cs.brown.edu/
  - + Keep functionality of RDBMS
  - + Leverage modern architectures (memory, CPU, network, etc.),
  - + Design for scale and performance
- Target a (mostly) new class of data problems
  - + Data arrives at a very fast rate
  - + Must be ingested, stored and acted upon in real-time

Availability and Durability

- High Availability
  - + Data replicated on multiple servers (synchronous, multi-master)
  - + Failed nodes can exit/rejoin cluster on the fly
  - + No single point of failure
- Durablilty
  - + Continuous database snapshots
  - + Between snapshots, transactions written to persistent storage
- Disaster Recovery
  - + Asynchronous, fault tolerant replication across WAN

VoltDB Transactions

- Transaction == Single SQL Statement or Stored Procedure Invocation
  - + Committed on Success

- Java Stored Procedures
  - + Java statements with embedded, parameterized SQL
  - + Efficiently process SQL at the server
  - + Move the code to the data, not the other way around

Client Application Interfaces

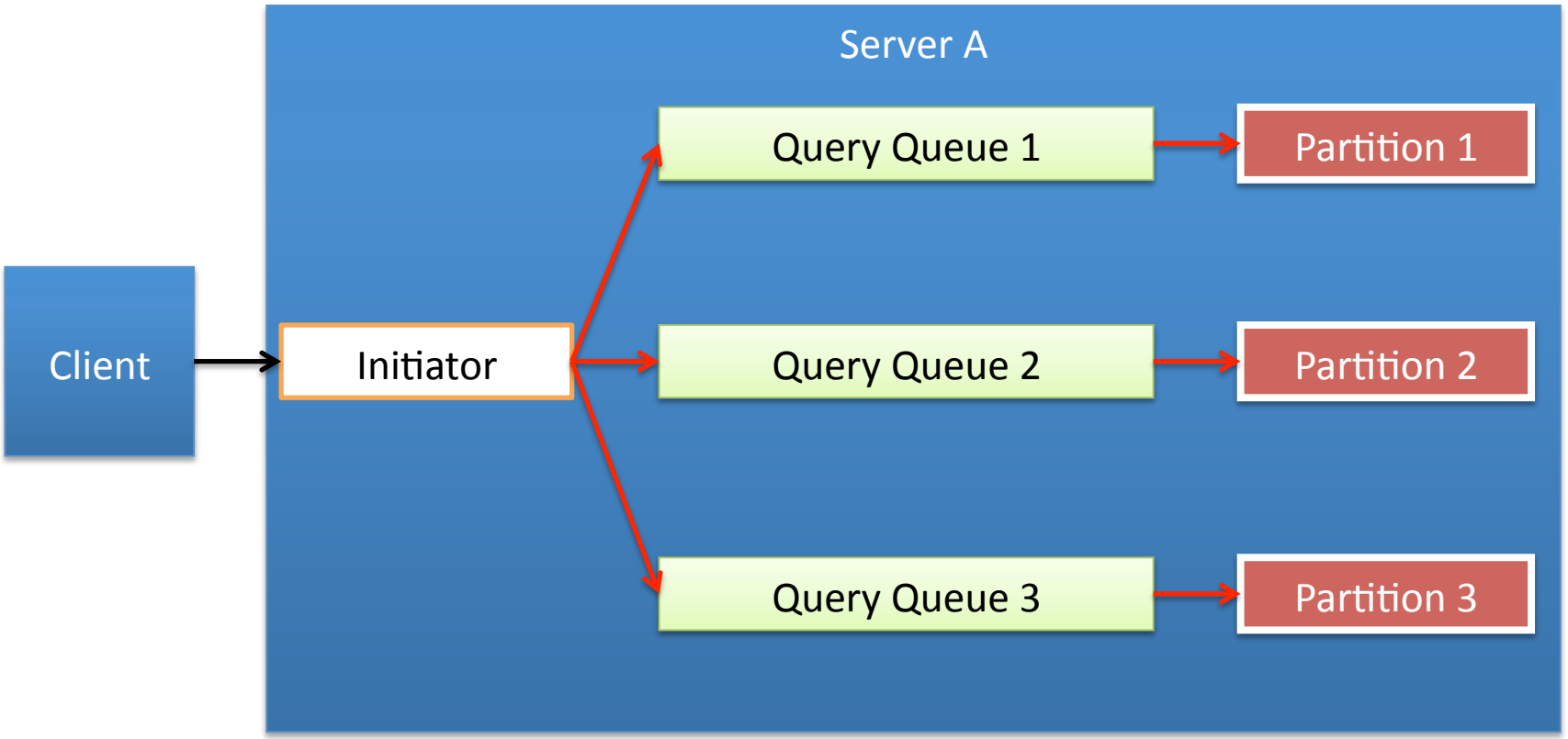
- Standard programming interfaces
  - + Build apps in the language of your choice
  - + Call Java stored procedures with parameterized, embedded SQL
- Client app connects to the cluster
  - + Data location is transparent
  - + Topology is transparent
  - + Cluster manages routing, data movement and consistency

VoltDB and Node.js Architectures

- VoltDB runs best when using an asynchronous client
  - + The more work you give to VoltDB, the better it runs, especially when acting upon several single partition queries
  - + Partitions? What's that? (stay tuned)
- Node is better when run asynchronously
  - + Events, nextTick, non-blocking
  - + Volt client is event driven, more so in future versions
  - + nextTick
  - + Non-blocking

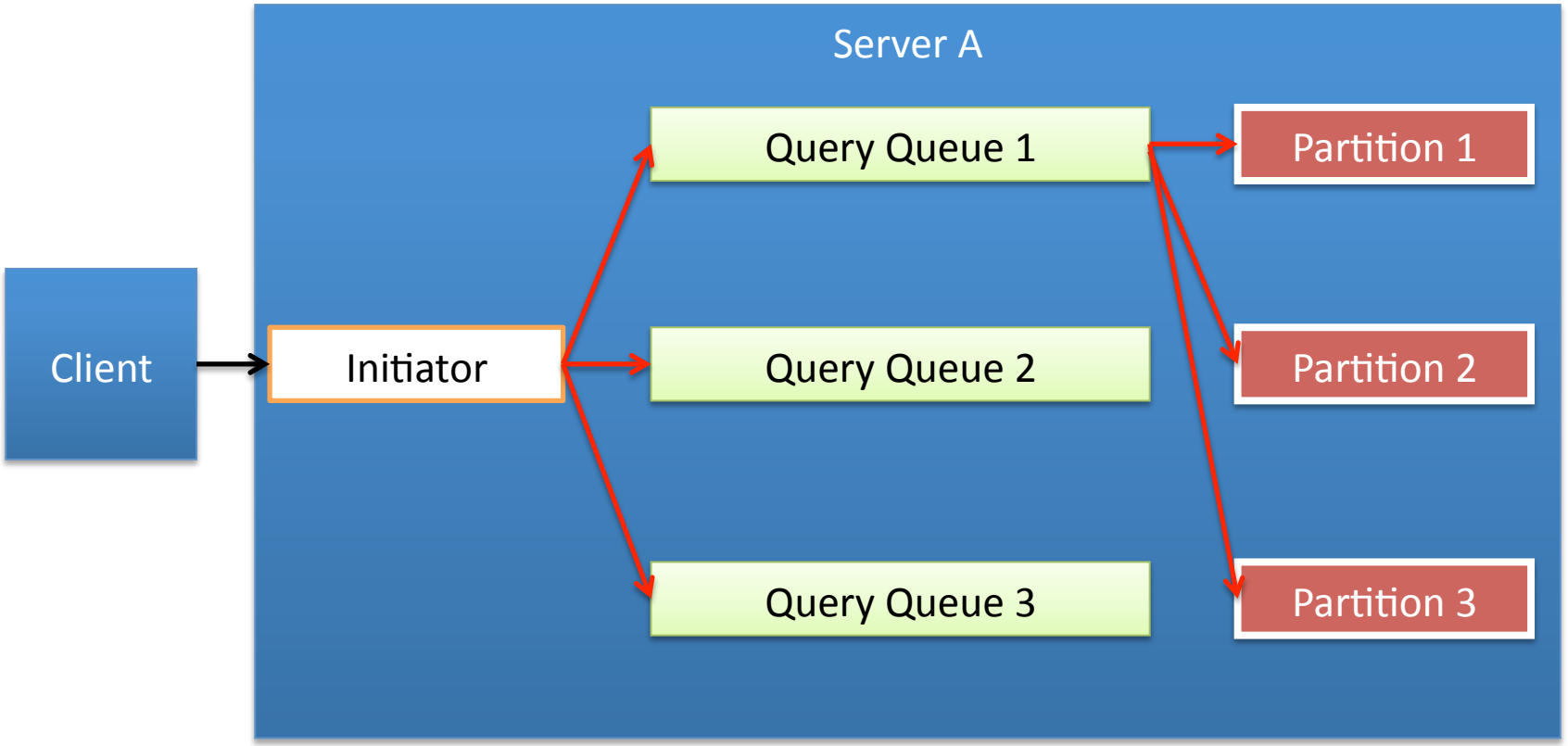
Single Partition Query

Queries run against their partition only and can execute in parallel



Multi-partition Query

Queries can span partitions, even when partitions are located on different nodes in the cluster (not shown in diagram)



The VoltDB Node.js Driver and Benchmark

The Original Driver: Voltjs

- Written by Jacob Wright
- Works well for limited number of transactions
- Supports all but one data type
- Only connects to one server
- Jacob donated the code to VoltDB

The New Driver

- Updated by Andy Wilson
- Optimized query coordinator
  - + Round-robin strategy
  - + Back-pressure management
- Adds varbinary data type support
- Better error handling
- Code reviewed by Felix Geisendörfer (Node.js expert)
  - + Implemented most recommendations

Benchmark Setup

- Amazon EC2
- Operating system: Ubuntu
- Node.js version: 0.6.10
- VoltDB Node.js driver version: 0.1.1
- VoltDB DBMS version: 2.2
- Client-side benchmark script: 0.71 – 0.74
- EC2 High-Memory Instances: m2.4xlarge
  - + 68 GB memory
  - + 8 virtual cores with 3.25 EC2 Compute Units each
  - + 64-bit

Benchmark Application: Voter

Simulates American Idol Voting System

- Massive transaction peak (millions of simultaneous callers)
- Each transaction (vote) executes 4 SQL statements
  1. Get the caller's location (read)
  2. Verify that the caller has not exceeded vote maximum (read)
  3. Verify that caller is voting for a valid contestant (read)
  4. If yes to all of the above, cast vote (write)

Benchmark Results

Amazon EC2:

64 core node.js cluster + 96 core VoltDB cluster

Results:

- **695,000 transactions per second (TPS)**
- 2,780,000 operations per second
- 100,000 TPS per 8 core client
- **12,500 TPS per node.js core**
- Stable even under extreme load
- Near linear scale

More on Scaling

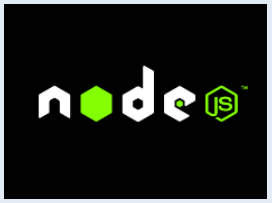
VoltDB Sites[1]	Node.js Threads[2]	Total TPS	TPS/Node Thread
16	64	87k	See note[3]
62	32	442k	13.8k
64	37	507k	13.7k[4]
42	43	511k	See note[3]
72	54	695k	12.8k[4]

Notes:

1. VoltDB "sites" ≈ cores used
2. Node.js threads ≈ cores used
3. Intentionally starving set-up
4. Partially starving

Details of the Final Test

- 8 Node.js client machines, each with 8 virtual cores: 64 cores
- Each Node.js client using 8 workers: 64 threads
- 10 threads starving: 54 active threads
- Total 695k TPS (2.8 million SQL statements)



- 12 VoltDB server machines with 8 virtual cores: 96 cores
- Each VoltDB host using 6 partitions: 72 partitions
- K factor 0
- Server idle 68% (66-72) with round robin client connections

VoltDB

Detailed report:
http://community.voltdb.com/sites/default/files/NodejsBenchmarkReport_April_2012.pdf

The Schema

```
CREATE TABLE contestants
  (
    contestant_number integer     NOT NULL
  , contestant_name   varchar(50) NOT NULL
  , CONSTRAINT PK_contestants PRIMARY KEY
    (
      contestant_number
    )
  );


CREATE TABLE votes
  (
    phone_number       bigint     NOT NULL
  , state              varchar(2) NOT NULL
  , contestant_number  integer    NOT NULL
);

CREATE TABLE area_code_state
  (
    area_code smallint   NOT NULL
  , state     varchar(2) NOT NULL
  , CONSTRAINT PK_area_code_state PRIMARY KEY
    (
      area_code
    )
  );
```

SQL Operations

```
// Check if the vote is for a valid contestant
SELECT contestant_number FROM contestants WHERE contestant_number = ?;

// Check if the voter has exceeded their allowed number of votes
SELECT num_votes FROM v_votes_by_phone_number WHERE phone_number = ?;

// Check an area code to retrieve the corresponding state
SELECT state FROM area_code_state WHERE area_code = ?;

// Record a vote
INSERT INTO votes (phone_number, state, contestant_number) VALUES (?, ?, ?);
```

The Transaction

```
// Check if the vote is for a valid contestant
voltQueueSQL(checkContestantStmt, EXPECT_ZERO_OR_ONE_ROW, contestantNumber);
voltQueueSQL(checkVoterStmt, EXPECT_ZERO_OR_ONE_ROW, phoneNumber);
voltQueueSQL(checkStateStmt, EXPECT_ZERO_OR_ONE_ROW, (short)(phoneNumber /
10000000l));

// Execute queued up statements (3 operations)
VoltTable validation[] = voltExecuteSQL();

// Error conditions
if (validation[0].getRowCount() == 0)
    return ERR_INVALID_CONTESTANT;

if ((validation[1].getRowCount() == 1) &&
        (validation[1].asScalarLong() >= maxVotesPerPhoneNumber))
    return ERR_VOTER_OVER_VOTE_LIMIT;

// Post the vote (1 operation)
voltQueueSQL(insertVoteStmt, EXPECT_SCALAR_MATCH(1), phoneNumber, state,
contestantNumber);
voltExecuteSQL(true);
```

DIY Instructions

I. Create EC2 VoltDB instances, *for each:*
   1. Install VoltDB
   2. Install NTP
   3. Set cluster config for deployment and run.sh
   4. Make one the startup-lead VoltDB host
   5. Start this VoltDB cluster

II. Create EC2 Node.js client instances, *for each:*
   1. Install git
   2. Install Node.js
   3. Download benchmark script
   4. Hardwire server domains
   5. Start the benchmark script

Complete benchmark instructions:

http://community.voltdb.com/sites/default/files/NodejsBenchmarkInstructions_April_2012.pdf

Summary

Driver

- Stable
- Fast
- Complete
- Easy to use
- Copes well under extreme load
- Maintained by VoltDB

Benchmark

- 695k TPS max, ~12k TPS/core
- Near linear scale

Resources

- VoltDB Download
  http://voltdb.com/products-services/downloads
- Benchmark Blog Post
  http://voltdb.com/company/blog/695k-tps-nodejs-and-voltdb
- Benchmark Report (40pg)
  http://community.voltdb.com/sites/default/files/NodejsBenchmarkReport_April_2012.pdf
- Benchmark instructions
  http://community.voltdb.com/sites/default/files/NodejsBenchmarkInstructions_April_2012.pdf
- Benchmark Client
  https://github.com/Eonblast/voltjs-bench/blob/master/bench.js
- Voter Example
  https://github.com/VoltDB/voltdb/tree/master/examples/voter
- VoltDB's Secret Sauce
  http://nms.csail.mit.edu/~stavros/pubs/hstore.pdf

Thank You!

Questions?

Stop by table #9 to chat and enter our Kindle raffle

If you're a SpringSource user, watch @voltdb on Twitter for some interesting news