THE EXPERT GUIDE TO FAST DATA

Why VoltDB is the solution to “Fast”
ACM Highlights

- **Learning Center** tools for professional development: [http://learning.acm.org](http://learning.acm.org)
  - 1,400+ trusted technical books and videos by **O’Reilly, Morgan Kaufmann**, etc.
  - Online training toward top vendor certifications (CEH, Cisco, CISSP, CompTIA, PMI, etc)
  - Learning Webinars from thought leaders and top practitioner
  - ACM Tech Packs (annotated bibliographies compiled by subject experts)
  - Podcast interviews with innovators and award winners

- Popular publications:
  - Flagship *Communications of the ACM (CACM)* magazine: [http://cacm.acm.org/](http://cacm.acm.org/)
  - *ACM Queue* magazine for practitioners: [http://queue.acm.org/](http://queue.acm.org/)

- **ACM Digital Library**, the world’s most comprehensive database of computing literature: [http://dl.acm.org](http://dl.acm.org).

- International conferences that draw leading experts on a broad spectrum of computing topics: [http://www.acm.org/conferences](http://www.acm.org/conferences).

- Prestigious awards, including the **ACM A.M. Turing** and **ACM - Infosys Foundation Award**: [http://awards.acm.org/](http://awards.acm.org/)

- And much more... [http://www.acm.org](http://www.acm.org).

THE EXPERT GUIDE TO FAST DATA

Why VoltDB is the solution to “Fast”
OUR SPEAKERS

Dr. Mike Stonebraker of MIT  
Co-founder of VoltDB

John Hugg  
Senior Software Engineer, VoltDB
OUTLINE

• Characteristics of fast data
• Non-workable solutions
• VoltDB solution
• Lambda architecture solution
FAST DATA

• Comes from humans
  • State management in multi-player internet games
  • E.g., leaderboards
• Comes from the Internet of Things (IoT)
  • Real-time geo-positioning
  • E.g., Waze
• Comes from both
• E.g., stock market transactions
FAST DATA RATES

- 10 messages (transactions) per second
  - Use your cell phone
- 1,000 transactions per second
  - Use RDBMS (or whatever)
- 100,000 transactions per second
  - Now it gets interesting…

- From now on we will use “transaction” and “message” interchangeably
REQUIREMENTS FOR FAST DATA APPLICATIONS

• Keep up
  • Obviously
  • And continue to do so when your load changes

• Only game in town is “scale out”
  • Not “scale up”

• Avoid pokey products
  • Product 1 executes 1,000 messages per core
  • Product 2 executes 25,000 messages per core
  • Difference between P1 and P2 on 100,000 messages per second is 4 cores versus 100 cores
REQUIREMENTS FOR FAST DATA APPLICATIONS

• High level language
  • SQL!
  • Don’t want to code in “message assembler”

• Augmented by windowing operations
  • E.g., moving average of IBM stock price every over last 10 trades
  • So-called windowed aggregates
REQUIREMENTS FOR FAST DATA APPLICATIONS

• High availability (HA)
  • I don’t know anybody who will take down time these days

• Requires a backup machine
  • And real-time failover
  • As well as restore on recovery
REQUIREMENTS FOR FAST DATA APPLICATIONS

• Never lose my data
  • Unacceptable to lose my airline reservation
  • Or my standing on the leaderboard
• Requires no data loss during failover
  • Unacceptable to drop transactions on the floor
REQUIREMENTS FOR FAST DATA APPLICATIONS

• Data Consistency
  • Unacceptable to sell the last widget to multiple customers
  • Or do a money transfer, where only half of it gets done
  • Or produce an incorrect leaderboard
• Requires standard ACID transactions
REQUIREMENTS FOR FAST DATA APPLICATIONS

• Data Consistency for replicas
  • Unacceptable to sell the last widget to multiple customers during a node failure
  • Or do a money transfer, where only half of it gets done during a node failure

• Requires standard ACID transactions
  • On replicas as well as data
  • Eventual consistency does not work!
NON-SOLUTIONS FOR FAST DATA

• RDBMSs (Oracle, MySQL, …)
• NoSQL (Cassandra, Mongo, …)
NON-SOLUTIONS FOR FAST DATA -- RDBMSS

• Four major sources of overhead (assuming data sits in main memory)
  • Buffer pool overhead
  • Locking overhead
  • Write-ahead log overhead
  • Threading overhead

• In aggregate these account for ~90% of the total time
NON-SOLUTIONS FOR FAST DATA -- RDBMSS

• Slow, slow, slow, slow
  • Disk-based system (buffer pool overhead)
  • Record-level locking too expensive
  • Aries-style write ahead logging too expensive
  • Multi-threading latches are killing

• Limited to a few thousand transactions per second

• If you know you will never need to go faster, then this will work
NON-SOLUTIONS FOR FAST DATA -- NOSQL

• Low level language (message assembler)
• No ACID!!!!
• Buffer pool and threading overhead still present
• Worst of all worlds – low performance and low function
SOLUTIONS FOR FAST DATA

• High performance main memory SQL-ACID DBMS (VoltDB, Hekaton, Hana, …)
• Complex event processing engine (CEP) (Storm, Streambase, …)
EXAMPLE OPERATION ON FAST DATA

• First hedge fund example
  • Find me a strawberry followed within 5 msec by a banana followed with 10 msec by a grape
  • Look for complex patterns in a fire hose
  • CEP is a natural here
EXAMPLE OPERATION ON FAST DATA

• Second hedge fund example
  • In a worldwide trading system
  • Keep the global state on the enterprise
    • For or against every stock in real time (msecs)
  • And ring the red telephone if there is too much risk
  • And don’t lose any messages!!!

• Sweet spot for SQL-ACID-main-memory
CHARACTERIZATION

• CEP natural for “big pattern little state” applications
• Main memory SQL natural for “big state little pattern” applications
• Note that analytics applications are all in the second bucket
• Anecdotal evidence that there are 3-4 big state problems for every big pattern problem
  • “an unnamed but reliable source”
VOLTDB SOLUTION

- SQL plus windows
- Main memory
- Scale out on N nodes
- Very high performance
  - Figure 40,000 messages/transactions per core per second
VOLTDB SOLUTION

• ACID
  • With a lot of detailed trickery

• ACID on local replicas
  • With more trickery

• Optional ACID on remote replicas
  • Nobody is willing to pay the latency cost…. 
VOLTDB FAST DATA DEMO

John Hugg
VoltDB Founding Engineering
The Lambda Architecture

Now with an Example
LAMBDA OVERVIEW

- Batch processing is well understood and robust. Latency is pretty horrific.
- Stream processing is immediate. Complex and not as robust to hardware or user failure.
- Lambda Architecture says do both in parallel to compensate.

Speed Layer & Batch Layer
EXAMPLE LAMBDA STACK

Speed Layer

STORM

cassandra

Batch Layer

amazon web services™ S3

cascading

kafka
EXAMPLE PROBLEM
HOW MANY PEOPLE USED MY APP TODAY?
HOW MANY UNIQUE USERS INTERACTED WITH MY APP TODAY?
Open Cupcake Time

App Identifier
Unique Device ID

appid = 87
deviceid = 12
Open Cupcake Time

App Identifier
Unique Device ID

appid = 87
deviceid = 12
1 MILLION

APPID,DEVICEID

PAIRS PER SECOND
Enter HyperLogLog

A method of estimating cardinality.

\[
\text{blob} = \text{update}(\text{integer, blob})
\]

\[
\text{integer} = \text{estimate}(<blob>)
\]

Fixed blob size.

A few kilobytes to get 99% accuracy.
Open Cupcake Time

App Identifier
Unique Device ID

appid = 87
deviceid = 12

```
CREATE TABLE estimates
(
    appid bigint NOT NULL,
    devicecount bigint NOT NULL,
    hll varbinary(8192) DEFAULT NULL,
    PRIMARY KEY (appid)
);
PARTITION TABLE estimates ON COLUMN appid;
CREATE INDEX rank ON ESTIMATES
(devicecount);
```
Open Cupcake Time

App Identifier
Unique Device ID

appid = 87
deviceid = 12
DECLARE SQL STATEMENTS

```java
final static SQLStmt estimatesSelect =
    new SQLStmt("select devicecount, hll from estimates where appid = ?;");
final static SQLStmt estimatesUpsert =
    new SQLStmt("upsert into estimates (appid, devicecount, hll) values (?, ?, ?);");

public VoltTable[] run(long appId, long hashedDeviceId) throws IOException {
    // get the HLL from the db
    voltQueueSQL(estimatesSelect, EXPECT_ZERO_OR_ONE_ROW, appId);
    VoltTable estimatesTable = voltExecuteSQL()[0];

    HyperLogLog hll = new HyperLogLog(12);
    // if the row with the hyperloglog blob exists...
    if (estimatesTable.advanceRow()) {
        byte[] hllBytes = estimatesTable.getVarbinary("hll");
        hll = HyperLogLog.fromBytes(hllBytes);
    }

    // offer the hashed device id to the HLL
    hll.offerHashed(hashedDeviceId);

    // update the state with exact estimate and update blob for the hll
    voltQueueSQL(estimatesUpsert, EXPECT_SCALAR_MATCH(1), appId, hll.cardinality(), hll.toBytes());
    return voltExecuteSQL();
}
```
PARAMS ARE APP ID & DEVICE ID
```
final static SQLStmt estimatesSelect =
    new SQLStmt("select devicecount, hll from estimates where appid = ?;");
final static SQLStmt estimatesUpsert =
    new SQLStmt("upsert into estimates (appid, devicecount, hll) values (?, ?, ?);");

public VoltTable[] run(long appId, long hashedDeviceId) throws IOException {

    // get the HLL from the db
    voltQueueSQL(estimatesSelect, EXPECT_ZERO_OR_ONE_ROW, appId);
    VoltTable estimatesTable = voltExecuteSQL()[0];

    HyperLogLog hll = new HyperLogLog(12);
    // if the row with the hyperloglog blob exists...
    if (estimatesTable.advanceRow()) {
        byte[] hllBytes = estimatesTable.getVarbinary("hll");
        hll = HyperLogLog.fromBytes(hllBytes);
    }

    // offer the hashed device id to the HLL.
    hll.offerHashed(hashedDeviceId);

    // update the state with exact estimate and update blob for the hll
    voltQueueSQL(estimatesUpsert, EXPECT_SCALAR_MATCH(1), appId, hll.cardinality(), hll.toBytes());
    return voltExecuteSQL();
}
```
CREATE A HYPERLOGLOG STRUCTURE FROM THE ROW OR CREATE A NEW HLL IF NO ROW

```java
final static SQLStmt estimatesSelect =
    new SQLStmt("select devicecount, hll from estimates where appid = ?;"),
final static SQLStmt estimatesUpsert =
    new SQLStmt("upsert into estimates (appid, devicecount, hll) values (?, ?, ?);"),

public VoltTable[] run(long appId, long hashedDeviceId) throws IOException {

    // get the HLL from the db
    voltQueueSQL(estimatesSelect, EXPECT_ZERO_OR_ONE_ROW, appId);
    VoltTable estimatesTable = voltExecuteSQL()[0];

    HyperLogLog hll = new HyperLogLog(12);
    // if the row with the hyperloglog blob exists...
    if (estimatesTable.advanceRow()) {
        byte[] hllBytes = estimatesTable.getVarbinary("hll");
        hll = HyperLogLog.fromBytes(hllBytes);
    }

    // offer the hashed device id to the HLL.
    hll.offerHashed(hashedDeviceId);

    // update the state with exact estimate and update blob for the hll
    voltQueueSQL(estimatesUpsert, EXPECT_SCALAR_MATCH(1), appId, hll.cardinality(), hll.toBytes());
    return voltExecuteSQL();
}
```
ADD THIS UNIQUE ID TO THE HLL STRUCTURE

```java
final static SQLStmt estimatesSelect =
    new SQLStmt("select devicecount, hll from estimates where appid = ?;");  
final static SQLStmt estimatesUpsert =
    new SQLStmt("upsert into estimates (appid, devicecount, hll) values (?, ?, ?);"_generate_seq);

public VoltTable[] run(long appId, long hashedDeviceId) throws IOException {

    // get the HLL from the db
    voltQueueSQL(estimatesSelect, EXPECT_ZERO_OR_ONE_ROW, appId);
    VoltTable estimatesTable = voltExecuteSQL()[0];

    HyperLogLog hll = new HyperLogLog(12);
    // if the row with the hyperloglog blob exists...
    if (estimatesTable.advanceRow()) {
        byte[] hllBytes = estimatesTable.getVarbinary("hll");
        hll = HyperLogLog.fromBytes(hllBytes);
    }

    // offer the hashed device id to the HLL
    hll.offerHashed(hashedDeviceId);

    // update the state with exact estimate and update blob for the hll
    voltQueueSQL(estimatesUpsert, EXPECT_SCALAR_MATCH(1), appId, hll.cardinality(), hll.toBytes());
    return voltExecuteSQL();
}
```
UPDATE ROW WITH NEW HLL BYTES AND THE COMPUTED ESTIMATE

```java
final static SQLStmt estimatesSelect =
    new SQLStmt("select devicecount, hll from estimates where appid = ?;" );
final static SQLStmt estimatesUpsert =
    new SQLStmt("upsert into estimates (appid, devicecount, hll) values (?, ?, ?);" );

public VoltTable[] run(long appId, long hashedDeviceId) throws IOException {

    // get the HLL from the db
    voltQueueSQL(estimatesSelect, EXPECT_ZERO_OR_ONE_ROW, appId);
    VoltTable estimatesTable = voltExecuteSQL()[0];

    HyperLogLog hll = new HyperLogLog(12);
    // if the row with the hyperloglog blob exists...
    if (estimatesTable.advanceRow()) {
        byte[] hllBytes = estimatesTable.getVarbinary("hll");
        hll = HyperLogLog.fromBytes(hllBytes);
    }

    // offer the hashed device id to the HLL
    hll.offerHashed(hashedDeviceId);

    // update the state with exact estimate and update blob for the hll
    voltQueueSQL(estimatesUpsert, EXPECT_SCALAR_MATCH(1), appId, hll.cardinality(), hll.toBytes());
    return voltExecuteSQL();
}
```
ADVANTAGES
LESS COMPLEX OPERATIONALLY
LESS CODE IN FEWER PLACES

• HyperLogLog code is used entirely within one stored procedure.
• Client uses SQL + simple schema for queries & reporting.

SELECT appid, devicecount FROM estimates ORDER BY devicecount DESC LIMIT 10;
DEMO
WANT TO CELEBRATE MIKE?

Grab your commemorative Stonebraker Turing award t-shirt.

For more details visit:  
www.voltdb.com/stonebrakershirt
QUESTIONS?

• Use the chat window to type in your questions
• Try VoltDB yourself:
  
  ➢ Free trial of the Enterprise Edition:
    • [www.voltdb.com/download](http://www.voltdb.com/download)
  
  ➢ Try VoltDB in the Cloud
  
  ➢ Try the “Unique Devices” app
  
  ➢ Open source version of VoltDB is available on github.com
THANK YOU!
ACM: THE LEARNING CONTINUES...

• Questions about this webcast? learning@acm.org

• ACM Learning Webinars (on-demand archive): http://learning.acm.org/webinar

ACM Learning Center: http://learning.acm.org

• ACM SIGMOD: http://www.sigmod.org/

• ACM Queue: http://queue.acm.org/