NewSQL

Andy Pavlo
February 6, 2012
Outline

• The Last Decade of Databases
• NewSQL Introduction
• H-Store
Early-2000s

• All the big players were heavyweight and expensive.
  – *Oracle, DB2, Sybase, SQL Server, etc.*

• Open-source databases were missing important features.
  – *Postgres, mSQL, and MySQL.*
- Horizontal scalability through database splits
- Items split by category
- SPOF elimination

December, 2002

Randy Shoup - “The eBay Architecture”
http://highscalability.com/ebay-architecture
• Push functionality to application:
  • **Joins**
  • **Referential integrity**
  • **Sorting done**
• No distributed transactions.

December, 2002
Mid-2000s

- MySQL + InnoDB is widely adopted by new web companies:
  - Supported transactions, replication, recovery.
  - Still must use custom middleware to scale out across multiple machines.
  - Memcache for caching queries.
Facebook Architecture

Facebook Architecture

- Scale out using custom middleware.
- Store ~75% of database in Memcache.
- No distributed transactions.

Source: Jay Thadeshwar - “Technology Used by Facebook”
Late-2000s

- NoSQL systems are able to scale horizontally right out of the box:
  - *Schemaless.*
  - *Using custom APIs instead of SQL.*
  - *Not ACID (i.e., eventual consistency)*
  - *Many are based on Google’s BigTable or Amazon’s Dynamo systems.*
MongoDB Architecture

Nathan Tippy - “MongoDB”
MongoDB Architecture

- Easy to use.
- Becoming more like a DBMS over time.
- No transactions.
Early-2010s

• New DBMSs that can scale across multiple machines natively and provide ACID guarantees.
  – MySQL Middleware
  – Brand New Architectures
New SQL
451 Group’s Definition

• A DBMS that delivers the scalability and flexibility promised by NoSQL while retaining the support for SQL queries and/or ACID, or to improve performance for appropriate workloads.
Stonebraker’s Definition

• SQL as the primary interface.
• ACID support for transactions
• Non-locking concurrency control.
• High per-node performance.
• Parallel, shared-nothing architecture.

Michael Stonebraker - “New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps”
http://cacm.acm.org/blogs/blog-cacm/109710
On-Line Transaction Processing
OLTP Transactions

Fast

Repetitive

Small
Workload Characterization

- **Operation Complexity**
  - Simple
  - Complex

- **Workload Focus**
  - OLTP
  - Social Networks
  - Data Warehouses

Michael Stonebraker – “Ten Rules For Scalable Performance In Simple Operation’ Datastores”
http://cacm.acm.org/magazines/2011/6/108651
Transaction Bottlenecks

• Disk Reads/Writes
  – Persistent Data, Undo/Redo Logs

• Network Communication
  – Intra-Node, Client-Server

• Concurrency Control
  – Locking, Latching
An Ideal OLTP System

- Main Memory Only
- No Multi-processor Overhead
- High Scalability
- High Availability
- Autonomic Configuration
```java
class NewOrder extends StoredProcedure {
    Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W_ID = ?";
    Query CheckStock = "SELECT S_QTY FROM STOCK
                        WHERE S_W_ID = ? AND S_I_ID = ?";
    Query InsertOrder = "INSERT INTO ORDERS VALUES (?,?)";
    Query InsertOrdLine = "INSERT INTO ORDER_LINE VALUES (?, ?, ?, ?,?)";
    Query UpdateStock = "UPDATE STOCK SET S_QTY = S_QTY - ?
                        WHERE S_W_ID = ? AND S_I_ID = ?";

    run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[])
    queueSQL(GetWarehouse, w_id);
    for (int i = 0; i < i_ids.length; i++)
        queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
    Result r[] = executeBatch();

    int o_id = r[0].get("W_NEXT_O_ID") + 1;
    queueSQL(InsertOrder, w_id, o_id);
    for (int i = 0; i < r.length; i++) {
        if (r[i+1].get("S_QTY") < i_qtys[i]) abort();
        queueSQL(InsertOrdLine, w_id, o_id, i_ids[i], i_qtys[i]);
        queueSQL(UpdateStock, i_qtys[i], i_w_ids[i], i_ids[i]);
    }
    return (executeBatch() != null);
}
```
Database Partitioning

TPC-C Schema

WAREHOUSE
  ↓
DISTRICT
  ↓
CUSTOMER
  ↓
ORDERS
  ↓
ORDER_ITEM

ITEM
  ↓
STOCK
  ↓
ORDERS

Schema Tree

WAREHOUSE

DISTRICT

CUSTOMER

ORDERS

ORDER_ITEM

ITEM

Replicated
Database Partitioning

Schema Tree

- WAREHOUSE
- DISTRICT
- CUSTOMER
- ORDERS
- ORDER_ITEM

Partitions

- P1
- P2
- P3
- P4
- P5

ITEM

Replicated
Distributed Transaction Protocol

Procedure Name
Input Parameters

<Timestamp, Counter, SiteId>

#2084922509960152064

P1
#208... #216... #229... #231...

P2
#208... #229... #231...
Distributed Transaction Protocol

Two-Phase Commit

- TransactionPrepare Request
- TransactionPrepare Response
- TransactionFinish Request
- TransactionFinish Response
H-Store vs. VoltDB

• An incestuous past
  – H-Store merged with Horizontica (Spring 2008)
  – VoltDB forked from H-Store (Fall 2008)
  – H-Store forked back from VoltDB (Winter 2009)

• Major differences:
  – Support for arbitrary transactions.
  – Google Protocol Buffer Network Communication