In-Memory Databases

OLAP versus OLTP

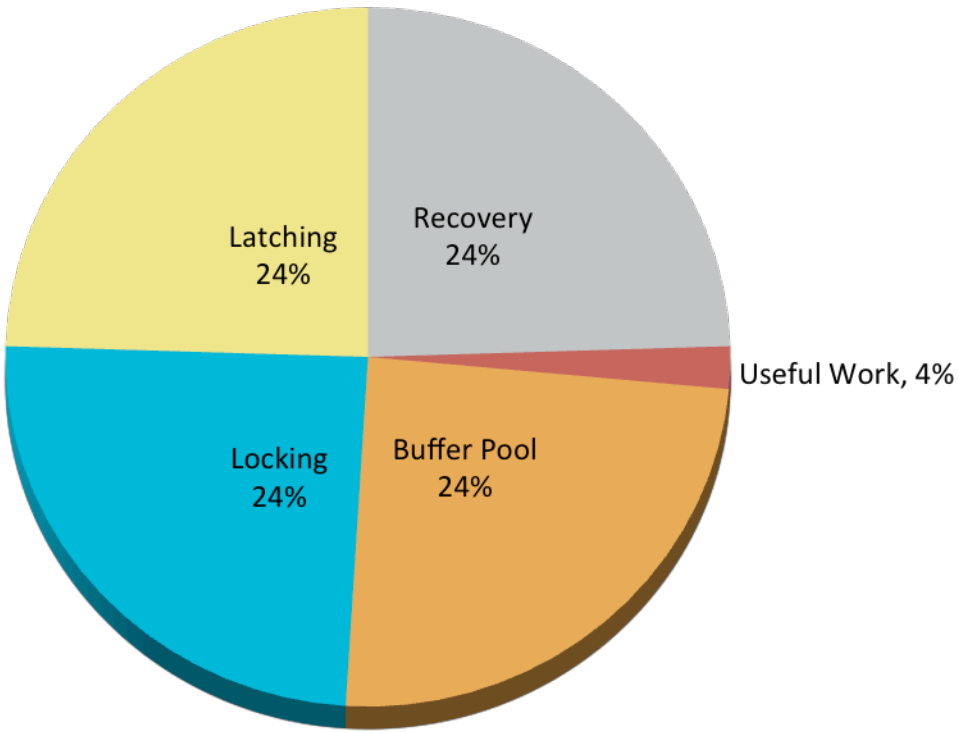
- OLAP (Business Intelligence):
  - Massive amounts of data
  - Complex queries
  - Large number of tables
  - Long running but still somewhat interactive
  - Mostly read only
- OLTP
  - Really only transactions, i.e., updates
  - Few tables touched
  - Typically generated queries

Reality Check - Size

- Transactional database size grows at the rate transactions increase
- 1 Tbyte is a really big TP database
- 1 Tbyte of main memory buyable for around $50K
  - (say) 64 Gbytes per server in 16 servers
- I.e. Moore's law has eclipsed TP data base size
- If your data doesn't fit in main memory now, then wait a couple of years and it will.....

Reality Check - Performance

- TPC-C CPU cycles
- On the Shore DBMS prototype
- OldSQL should be similar



To Go Faster…

- Focus on overhead
  - Better B-trees affects only 4% of the path length

- Get rid of ALL major sources of overhead
  - Main memory deployment – gets rid of buffer pool
    - Leaving other 75% of overhead intact
    - i.e., win is 25%

Solution Choices

- OldSQL
  - Legacy RDBMS vendors
- NoSQL
  - Give up SQL and ACID for performance
- NewSQL
  - Preserve SQL and ACID
  - Get performance from a new architecture

OldSQL

Traditional SQL vendors (the “elephants”)

- Code lines dating from the 1980’s
- “bloatware”
- Mediocre performance on New TP

NoSQL

- Give up SQL
- Give up ACID

Give Up SQL?

- Compiler translates SQL at compile time into a sequence of low level operations
- Similar to what the NoSQL products make you program in your application
- 30 years of RDBMS experience
  - Hard to beat the compiler
  - High level languages are good (data independence, less code)
  - Stored procedures are good!
    - One round trip from app to DBMS rather than one one round trip per record
    - Move the code to the data, not the other way around

Give Up ACID

- If you need data consistency, giving up ACID is a decision to tear your hair out by doing database "heavy lifting" in user code
- Can you guarantee you won't need ACID tomorrow?

ACID = goodness, in spite of what noSQL guys say

Who Needs ACID?

- Funds transfer
  - Or anybody moving something from X to Y
- Anybody with integrity constraints
  - Back out if fails
  - Anybody for whom "usually ships in 24 hours" is not an acceptable outcome
- Anybody with a multi-record state
  - E.g. move and shoot

NoSQL Summary

- Appropriate for non-transactional systems
- Appropriate for single record transactions that are commutative
- Not a good fit for New TP
- Use the right tool for the job
- But: Two recently-proposed NoSQL language standards – CQL and UnQL – are amazingly similar to (you guessed it!) SQL

NewSQL

- SQL
- ACID
- Performance and scalability through modern innovative software architecture

NewSQL

- Needs something other than traditional record level locking (1st big source of overhead)
  - timestamp order
  - MVCC
- Needs a solution to buffer pool overhead (2nd big source of overhead)
  - Main memory (at least for data that is not cold)
  - Some other way to reduce buffer pool cost
- Needs a solution to latching for shared data structures (3rd big source of overhead)
  - Some innovative use of B-trees
  - Single-threading

NewSQL

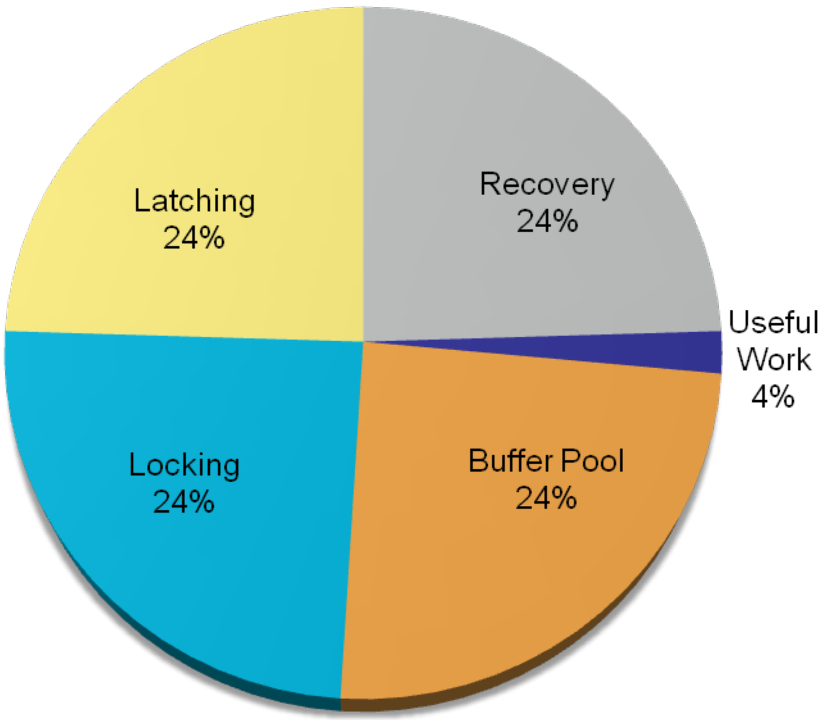
- Needs a solution to write-ahead logging (4th big source of overhead)
  - Obvious answer is built-in replication and failover
  - New TP views this as a requirement anyway

A NewSQL Example – VoltDB

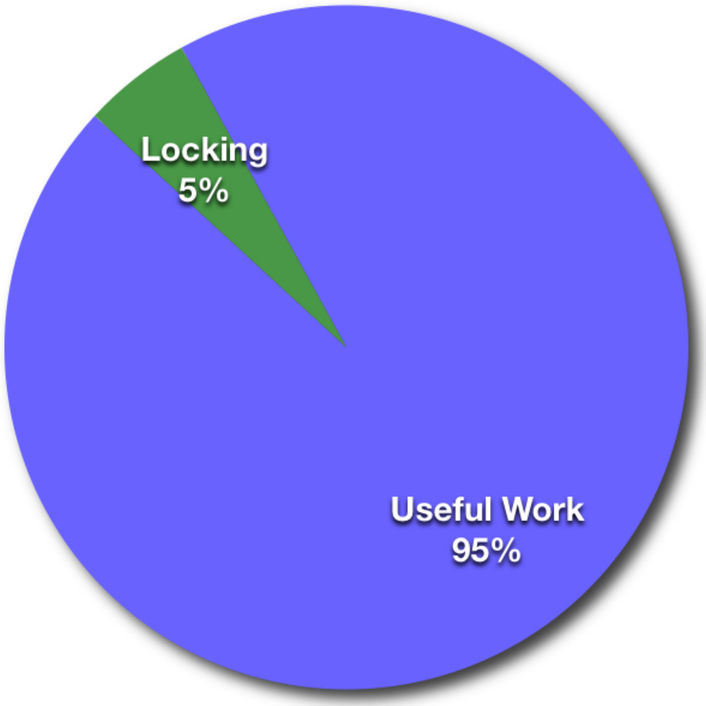
- Main-memory storage
- Single threaded, run transaction to completion
  - No locking
  - No latching
- Built-in high availability and durability
  - No log (in the traditional sense)

Where all the time goes... revisited

Before



VoltDB



Current VoltDB Status

- Runs a subset of SQL
- On VoltDB clusters (in memory on commodity hardware)
- With LAN and WAN replication
- 70x a popular OldSQL DBMS on TPC-C
- 5-7x Cassandra on VoltDB key-value layer
- Scales to 384 cores

Summary

Old TP

New TP

OldSQL for New OLTP		▪ Too slow ▪ Does not scale
NoSQL for New OLTP		▪ Lacks consistency guarantees ▪ Low-level interface
NewSQL for New OLTP		▪ Fast, scalable and consistent ▪ Supports SQL

Technical Overview

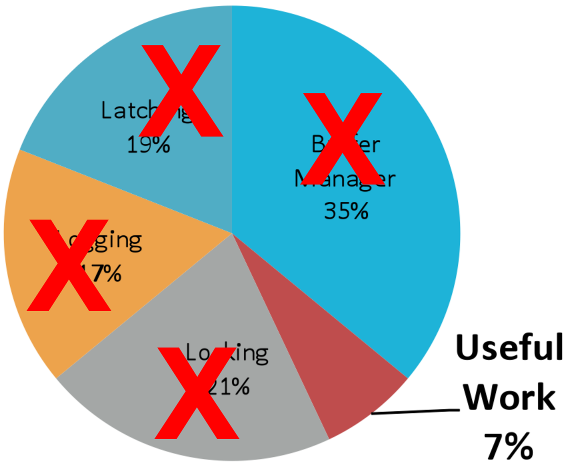
"OLTP Through the Looking Glass"

http://cs-www.cs.yale.edu/homes/dna/papers/oltpperf-sigmod08.pdf

VoltDB avoids the overhead of traditional databases

- K-safety for fault tolerance
- no logging
- In memory operation for maximum throughput
- no buffer management
- Partitions operate autonomously and single-threaded
- no latching or locking

Built to horizontally scale

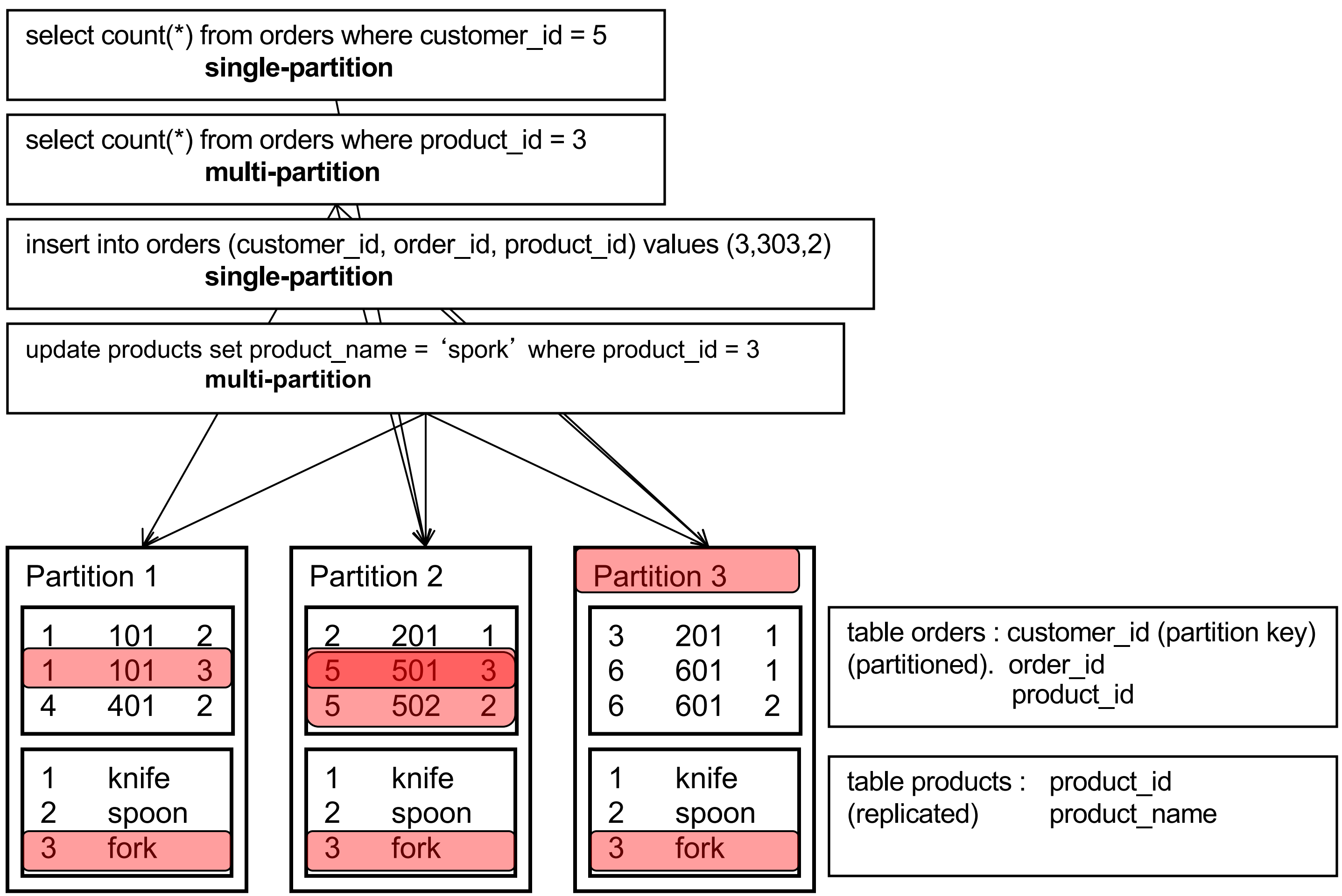


Technical Overview – Partitions (1/3)

- One partition per physical CPU core
  - Each physical server has multiple VoltDB partitions
- Data - Two types of tables
  - Partitioned
    - Single column serves as partitioning key
    - Rows are spread across all VoltDB partitions by partition column
    - Transactional data (high frequency of modification)
  - Replicated
    - All rows exist within all VoltDB partitions
    - Relatively static data (low frequency of modification)
- Code - Two types of work – both ACID
  - Single-Partition
    - All insert/update/delete operations within single partition
    - Majority of transactional workload
  - Multi-Partition
    - CRUD against partitioned tables across multiple partitions
    - Insert/update/delete on replicated tables

Technical Overview – Partitions (2/3)

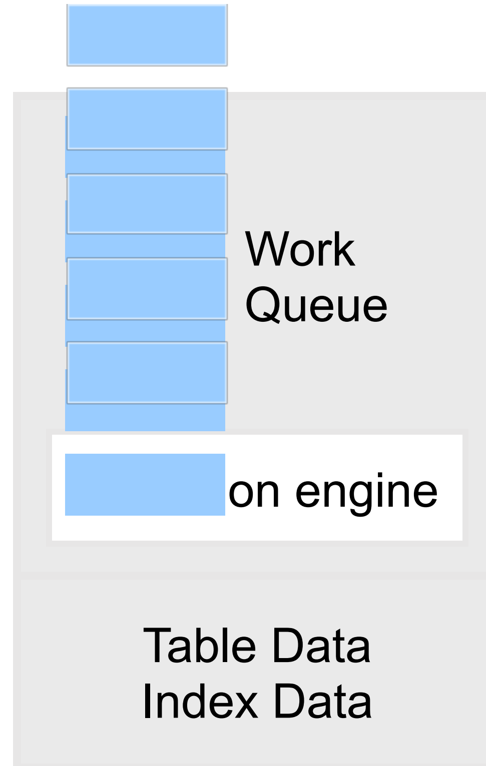
Single-partition vs. Multi-partition



Technical Overview – Partitions (3/3)

Inside a VoltDB partition…

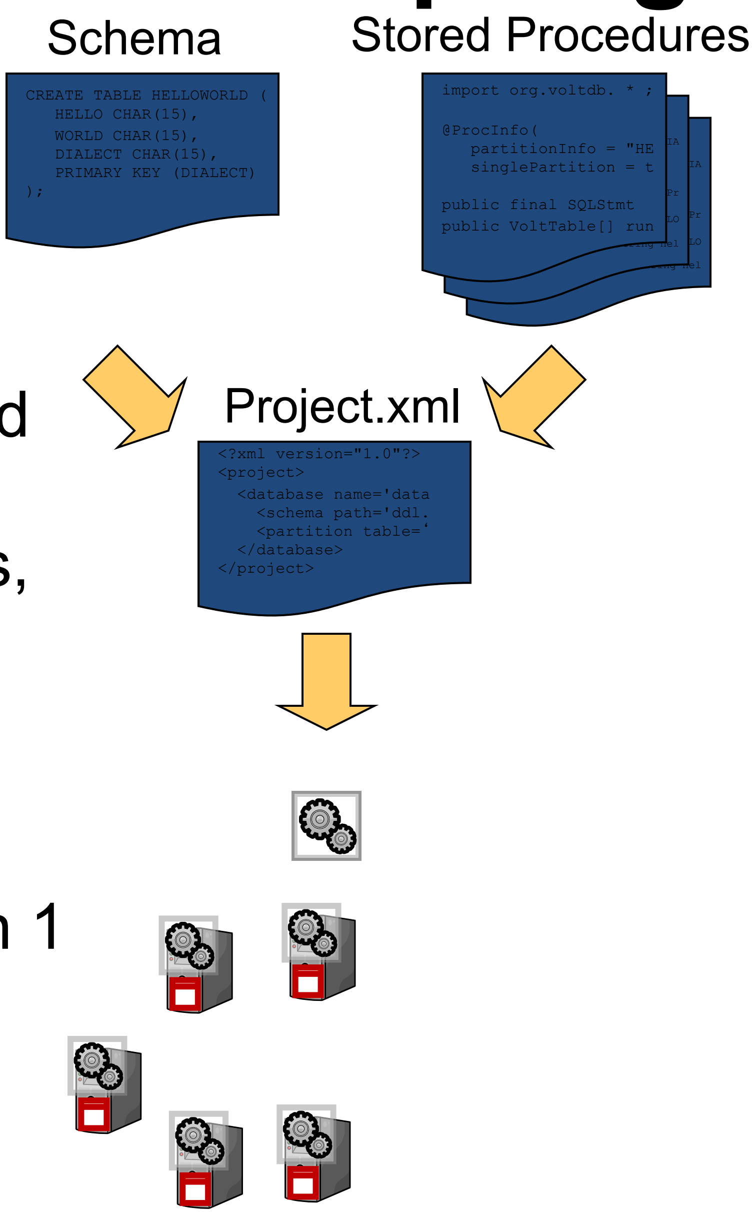
- Each partition contains data and an execution engine.
- The execution engine contains a queue for transaction requests.
- Requests are executed sequentially (single threaded).



- Complete copy of all replicated tables
- Portion of rows (about 1/partitions) of all partitioned tables

Technical Overview – Compiling

- The database is constructed from
  - The schema (DDL)
  - The work load (Java stored procedures)
  - The Project (users, groups, partitioning)
- VoltCompiler creates application catalog
  - Copy to servers along with 1 .jar and 1 .so
  - Start servers



Technical Overview - Transactions

- All access to VoltDB is via Java stored procedures (Java + SQL)
- A single invocation of a stored procedure is a transaction (committed on success)
- Limits round trips between DBMS and application
- High performance client applications communicate asynchronously with VoltDB

Technical Overview – Clusters/Durability

- Scalability
  - Increase RAM in servers to add capacity
  - Add servers to increase performance / capacity
  - Consistently measuring 90% of single-node performance increase per additional node
- High availability
  - K-safety for redundancy
- Snapshots
  - Scheduled, continuous, on demand
- Spooling to data warehouse
- Disaster Recovery/WAN replication (Future)
  - Asynchronous replication

VoltDB and OLTP

Asynchronous Communications

Client applications communicate asynchronously with VoltDB

- Stored procedure invocations are placed "on the wire"
- Responses are pulled from the server
- Allows a single client application to generate > 100K TPS
- Client library will simulate synchronous if needed

Traditional

```
salary := get_salary(employee_id);
```

VoltDB

```
callProcedure(asyncCallback, "get_salary", employee_id);
```

Transaction Control

VoltDB does not support client-side transaction control

- Client applications cannot:
  - insert into t_colors (color_name) values (‘purple’);
  - rollback;
- Stored procedures commit if successful, rollback if failed
- Client code in stored procedure can call for rollback

Lack of concurrency

- Single-threaded execution within partitions (single-partition) or across partitions (multi-partition)
- No need to worry about locking/dead-locks
  - great for "inventory" type applications
    - checking inventory levels
    - creating line items for customers
- Because of this, transactions execute in microseconds
- However, single-threaded comes at a price
  - Other transactions wait for running transaction to complete
  - Don't do anything crazy in a SP (request web page, send email)
  - Useful for OLTP, not OLAP

Throughput vs. Latency

- VoltDB is built for throughput over latency
- Latency measured in mid single-digits in a properly sized cluster
- Do not estimate latency as (1 / TPS)

SQL Support

- SELECT, INSERT (using values), UPDATE, and DELETE
- Aggregate SQL supports AVG, COUNT, MAX, MIN, SUM
- Materialized views using COUNT and SUM
- Hash and Tree Indexes
- SQL functions and functionality will be added over time, for now done in Java
- Execution plan for all SQL is created at compile time and available for analysis

Schema Changes

Traditional OLTP

- add table…
- alter table…

VoltDB

- modify schema and stored procedures
- build catalog
- deploy catalog