Not Your Father's Transaction Processing

Michael Stonebraker, CTO
VoltDB, Inc.

How Does This Fit into “Big Data”?

- Big volume
  - + I have too much data
- Big velocity
  - + Data is coming at me too fast
- Big variety
  - + I have too many data sources

High Velocity Applications

- Traditional transaction processing
- “New” transaction processing
- High velocity ingest

Traditional Transaction Processing

- Remember how we used to buy airplane tickets in the 1980s
  - + By telephone
  - + Through an intermediary (professional terminal operator)
- Commerce at the speed of the intermediary
- In 1985, 1,000 transactions per second was considered an incredible stretch goal!!!!
  - + HPTS (1985)

Traditional Transaction Processing

- Workload was a mix of updates and queries
- To an ACID data base system
  - + Make sure you never lose my data
  - + Make sure my data is correct
- At human speed
- Bread and butter of RDBMSs (OldSQL)

How has TP Changed in 25 Years?

The internet

- + Client is no longer a professional terminal operator
- + Instead Aunt Martha is using the web herself

- + Sends TP volume through the roof
- + Serious need for scalability and performance

How has TP Changed in 25 Years?

PDAs

+ Your cell phone is a transaction originator

+ Sends TP volume through the roof
+ Serious need for scalability and performance

Need in some traditional markets
for much higher performance!

And TP is Now a Much Broader Problem

The internet enables a green field of new TP applications

+ Massively multiplayer games (state of the game, leaderboards, selling virtual goods are all TP problems)
+ Social networking (social graph is a TP problem)
+ Real time ad placement
+ Real time couponing

+ And TP volumes are ginormous!!
+ Serious need for speed and scalability!

And TP is Now a Much Broader Problem

Sensor Tagging generates new TP applications

+ Marathon runners (fraud detection, leaderboards)
+ Taxicab (scheduling, fare collection)
+ Dynamic traffic routing
+ Car insurance “by the drink”
+ Mobile social networking

+ And TP volumes are ginormous!!
+ Serious need for speed and scalability!

And TP is Now a Much Broader Problem

Electronic commerce is here

+ Wall Street electronic trading
+ Real-time fraud detection
+ Micro transactions (through your PDA)

+ And TP volumes are ginormous!!
+ Serious need for speed and scalability!

Add in High Velocity Ingest

+ Real time click stream analysis
+ Most anything upstream from Hadoop
+ Or your data warehouse
+ Real time risk assessment on Wall Street

+ And TP volumes are ginormous!!
+ Serious need for speed and scalability!

In all cases…..

- Workload is a mix of updates and queries
- Coming at you like a firehose
- Still an ACID problem
  - + Don't lose my data
  - + Make sure it is correct
- Tends to break traditional solutions
  - + Scalability problems (volume)
  - + Response time problems (latency)

Put Differently

You need to ingest a firehose in real time

You need to process, validate, enrich and respond in real-time (i.e. update)

You often need real-time analytics (i.e. query)

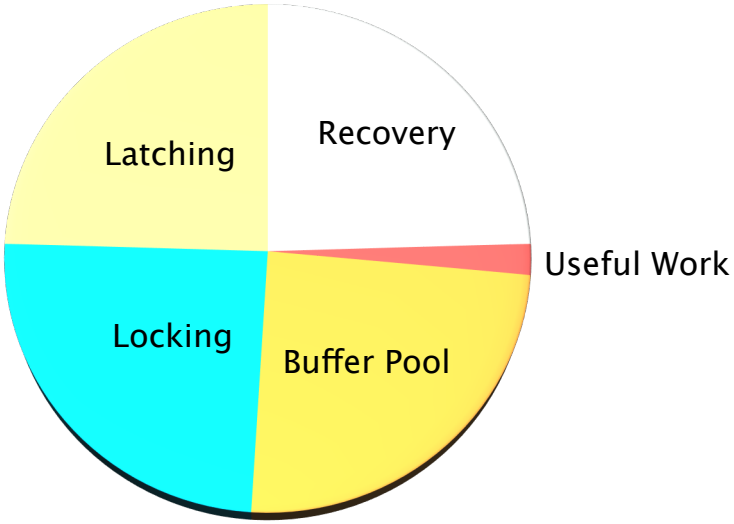
High velocity and you

Reality Check -- Size

- TP data base size grows at the rate transactions increase
- 1 Tbyte is a really big TP data base
- 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
- Elephants should be similar



To Go a Lot Faster You Have to……

- 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

The Elephants

- Are slow because they spend all of their time on overhead!!!
  - + Not on useful work
- Would have to re-architect their legacy code to do better

Long Term Elephant Outlook

- Up against “The Innovators Dilemma”
  - + Steam shovel example
  - + Disk drive example
  - + See the book by Clayton Christenson for more details
- Long term drift into the sunset
  - + The most likely scenario
  - + Unless they can solve the dilemm

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 these 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

Who needs ACID in replication

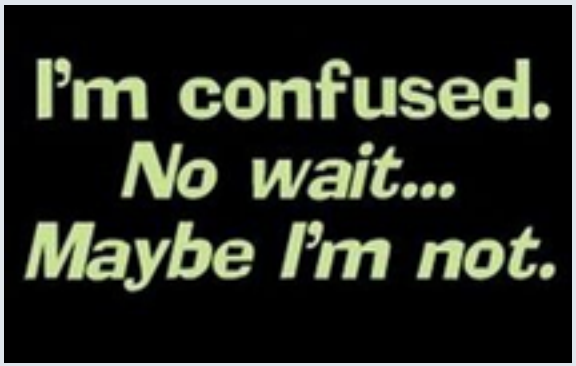
- Anybody with non-commutative updates
  - For example, + and * don't commute
- Anybody with integrity constraints
  - Can't sell the last item twice….
- Eventual consistency means "creates garbage"

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

Interesting

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

Table Stakes

- Scalability
  - + Run on a cluster of nodes
  - + One node obviously won't scale
- Automatic sharding
  - + Parallelism
- Focus on OLTP workload
  - + A few high volume transaction signatures (do as stored procedures)
  - + Occasional ad-hoc transactions

NewSQL Issue #1: Buffer Pool

- Obvious answer: main memory DBMS

Yabut: What if My Data Doesn't Fit?

- Main memory DBMSs can spill cold data to disk
  - + Without excessive overhead
- If your data is zipf-ian, you should be ok

NewSQL Issue #2: Write Ahead Log

- Obvious answer: replication and tandem-style failover (and fail back)
  - + Required for New TP anyway

Yabut: What if the Power Goes Out?

- You will die if you use conventional write-ahead logging (WAL)
- Ergo do something much cheaper
  - + Periodic checkpointing (costs next to nothing)
  - + Command log (stored procedure identifer plus parameters) with group commit
- Way better runtime performance; worse recovery time
  - + But total cluster failures are quite rare

NewSQL Issue #3: Multithreading

- Don’t do it

Yabut: What About Multicore?

- For A K-core CPU, divide memory into K (non overlapping) buckets
- i.e. convert multi-core to K single cores

NewSQL Issue #4: Record Level Locking

- Obvious answer: run to completion in timestamp order
- Ditto for replicas
  - + No locking!

The Details (1st of 2)

- Single shard transactions (the common case)
  - + Sequenced by shard controller
  - + With forwarding to replicas, who do transactions in sequence order (replicas are ACID)
- "One-shot" multi-shard transactions (the rare case)
  - + Sequenced by a single multi-shard controller
  - + Inserted into the single-shard stream at each shard independently (Everything still ACID)

The Details (2nd of 2)

- General transactions (multi-shard, multi-shot) (the very rare case)
  - + Sequenced by the single multi-shard controller
  - + Inserted into the single-shard stream at each shard independently
  - + BUT every affected shard must stall until all shots have been processed (Everything still ACID, but stalls are bad)
  - + To avoid the stall, shards must go into "speculative execution mode" (process xacts, without commit.

VoltDB Summary

- Main-memory storage
- Single threaded, run Xacts to completion
  - + No locking
  - + No latching
- Built-in HA and durability
  - + No log (in the traditional sense)

Current VoltDB Status

- Runs a subset of SQL (which is getting larger)
- On VoltDB clusters (in memory on commodity gear)
- With LAN and WAN replication
- 70X a popular OldSQL DBMS on TPC-C
- 5-7X Cassandra on VoltDB K-V layer
- Scales to 384 cores (biggest iron we could get our hands on)
- Clearly note this is an open source system!

Summary

Old TP

New TP

OldSQL for New OLTP		▪ Too slow ▪ Does not scale
NoSQL for New OLTP		▪ Lacks consistency guarantees
NewSQL for New OLTP		▪ Fast, scalable and consistent

Beware of Any Vendor

- Who is multi-threaded
- Who implements a traditional write-ahead log
- Who uses ODBC or JDBC for high volume transactions
- Who implements record level locking
- Who runs a disk-based system

Thank You