Database
Transaction Processing

Today’s scenario… “Big Data”

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

Traditional Transaction Processing

- 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!!!!
  - + High Performance Transaction System (1985)

Traditional Transaction Processing

- Workload was a mix of updates and queries
- To an ACID database 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 are using the web herself

+ Sends TP volume increases very fast
+ Serious need for scalability and performance

How has TP Changed in 25 Years?

PDAs

+ Your cell phone is a transaction originator

Need much higher performance!

TP is Now a Much Broader Problem (New TP)

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 coupons offering discounts

+ TP volumes are extremely large!!
+ 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
+ Airline reservation, shipping, manufacturing, payroll,….
+ 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 (user behavior over web)
+ 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, improve quality** **and** **respond** in real-time (i.e. update)

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

High velocity and you

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

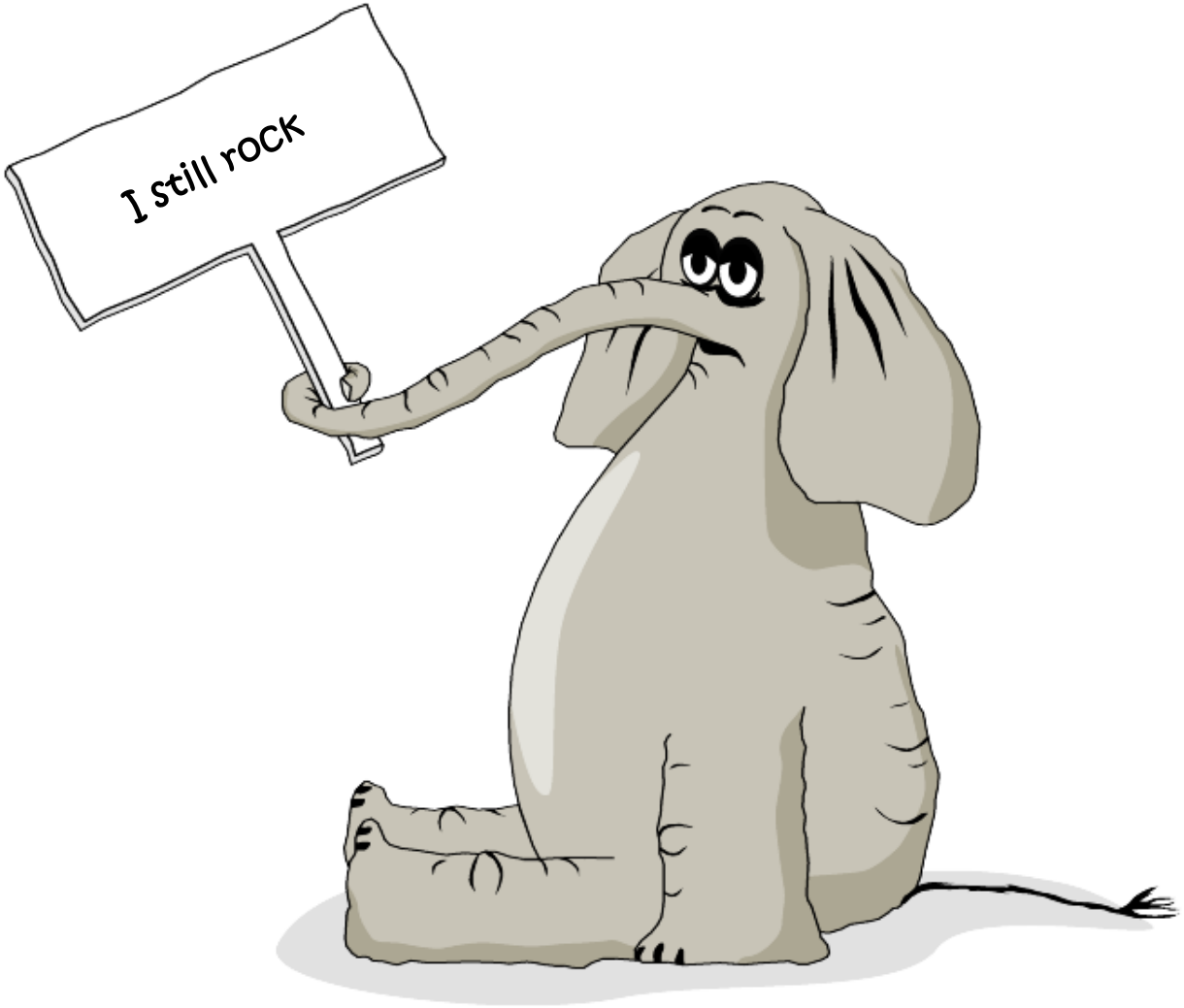
+ “software usefulness is reducing”

+ 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



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 program in your application

Give Up ACID

- Can you guarantee you won't need ACID tomorrow?

ACID = goodness, in spite of what these guys say

Who Needs ACID?

- Funds transfer
  - + 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

Who needs ACID in replication

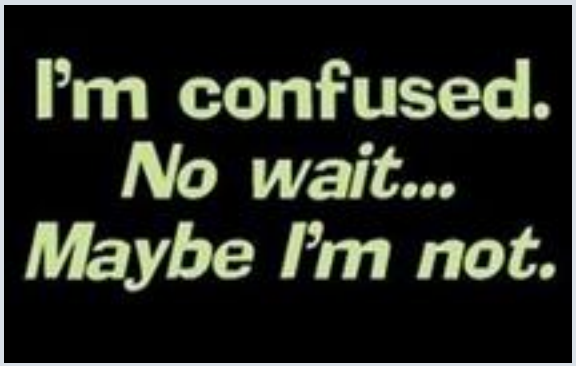
- Anybody with non-commutative updates
  + For example, + and * don't commute
- Anybody with integrity constraints
  + Can't sell the last item twice….
- Eventual consistency means "creates garbage"

NoSQL Summary

- Appropriate for non-transactional systems (read only)
- 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

NewSQL

- Needs something other than traditional record level locking (1st big source of overhead)
  - + timestamp order
  - + MVCC
  - + Your good idea goes here

NewSQL

- 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

NewSQL

- A problem with the shared database
  - + difficult to define and enforce clear boundaries between systems.
  - + difficult to define any kind of meaningful data schema that can be used by multiple applications.
- Needs a solution to issues for shared data structures (3rd big source of overhead)
  - + Some innovative use of B-trees
    - — Data file degradation problem is solved by using B+-Tree File Organization
  - + Single-threading- Oracle JVM is single-threaded at the execution level.
  - + Your good idea goes here

NewSQL

- Write-ahead logging: a standard way to ensure data integrity and reliability.
  - + Any changes made on the database are first logged in an append-only file called write-ahead Log or Commit Log.
  - + Then the actual blocks having the data (row, document) on the disk are updated.

- Needs a solution to write-ahead logging (4th big source of overhead)
  - + Obvious answer is built-in replication and failover
    - — Failover and failback operations help you ensure that your business will function even if a disaster strikes your production site.
    - — Failover is a process of switching from the VM on the source host to its VM replica on a host in the disaster recovery site.

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
- **Some details**
  - + **On-line failover?**
    - — **a backup operational mode that automatically switches to a standby database, server or network if primary fails**
  - + **On-line failback?**
    - — **process of returning production to its original location after a disaster or a scheduled maintenance period.**
  - + LAN network partitioning?
  - + WAN network partitioning?

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
- Some details
  - + On-line failover?
  - + On-line failback?
  - + **LAN network partitioning?**
    - — a network failure causes the members to split into multiple independent groups- a member in a group cannot communicate with members in other groups.
    - — In a partition scenario, all sides of the original cluster operate independently assuming members in other sides are failed.
  - + WAN network partitioning?

A NewSQL Example – VoltDB

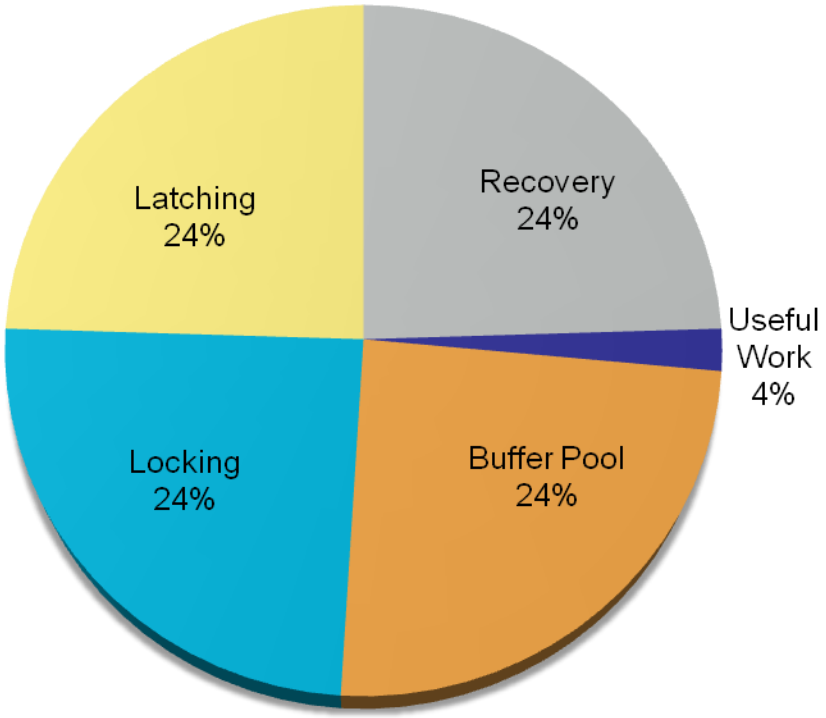
- Main-memory storage
- Single threaded, run Xacts to completion
  - + No locking
  - + No latching
- Built-in High Availability and durability
  - + No log (in the traditional sense)

What About Multicore?

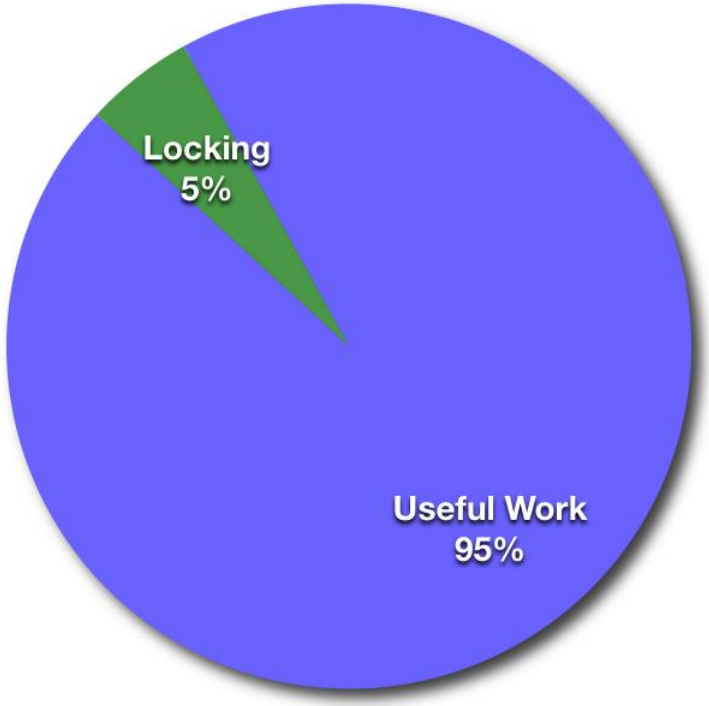
- For A K-core CPU, divide memory into K (non overlapping) buckets
- i.e. convert multi-core to K single cores

Where all the time goes… revisited

Before



VoltDB



Current VoltDB Status

- Runs a subset of SQL (which is getting larger)
- On VoltDB clusters (in memory)
- With LAN and WAN replication
- 70X a popular OldSQL DBMS on TPC-C
  - TPC-C-Trans. Processing Performance Council Benchmark C
    - measured in transactions per minute (tpmC) (batch of 5 trans..)
- 5-7X Cassandra on VoltDB Key-Value layer
- Scales to 384 cores
- 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 ▪ Low-level interface/manipulation
NewSQL for New OLTP		▪ Fast, scalable and consistent ▪ Supports SQL

OLTP- Online Transaction Processing

Thank You