A Scalable, Available Storage Tier for RDBMS

Ashraf Aboulnaga
Rui Liu
Ken Salem

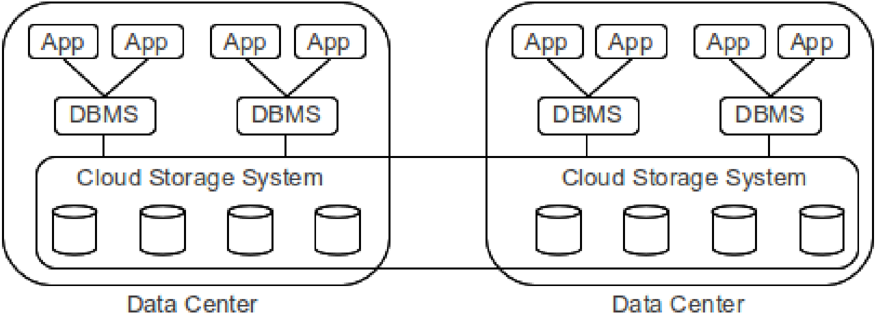
David R. Cheriton School of Computer Science
University of Waterloo

2 Nov 2011

Objective

- multi-tenant relational database management service with
  - elastic scalability of storage capacity, performance, tenancy
  - no down time
  - transactions
  - SQL
- starting points
  - established relational DBMS, e.g., MySQL
  - “NoSQL” systems, e.g., HBase, Cassandra

Our Approach



Benefits

- what we get:
  - scalable, elastic storage capacity and bandwidth
  - scalable, elastic tenancy
  - highly available storage tier, including disaster tolerance
  - transactions
  - SQL
- what we don't:
  - scaling of individual hosted DBMS tenants
    - but existing techniques can be applied
  - always-up hosted DBMS
    - but always-up storage tier might simplify DBMS high-availability

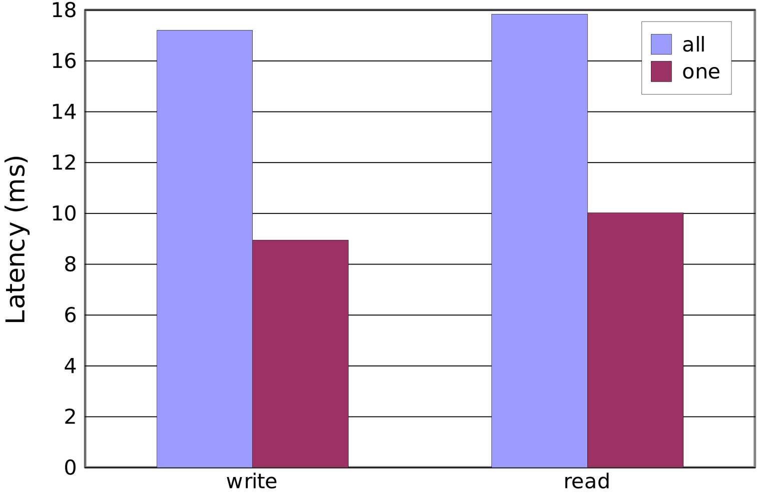
DBECS

- we use
  - MySQL as the hosted DBMS (but most will do)
  - Cassandra, an eventually consistent storage tier
- why Cassandra?
  - multi-master replication
    - multiple data centers
    - partition tolerance
  - fine-grained (per-operation) control of consistency/performance tradeoff
  - client-controlled update serialization

A Cassandra Primer

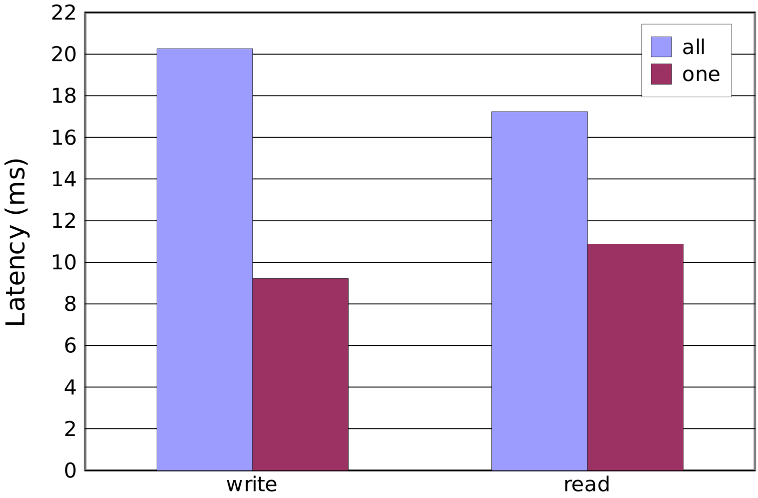
- stores “column families”, tables of semi-structured records, accessed by key
- records replicated and distributed by hashing keys
- primitive operations are reading a field from a record, update a field in a record
- per-operation consistency specification:
  - `write(1)` vs. `write(ALL)`
  - `read(1)` vs. `read(ALL)`
- scalable and available

Latency vs. Consistency in Cassandra



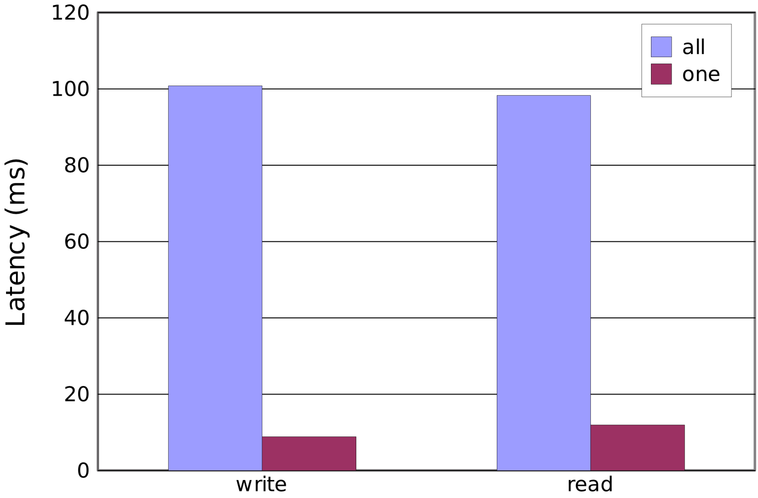
one EC2 availability zone

Latency vs. Consistency in Cassandra



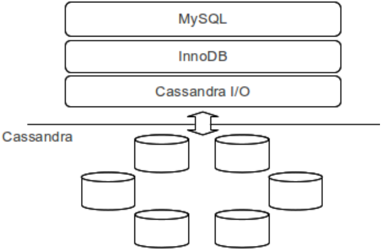
two availability zones, one region

Latency vs. Consistency in Cassandra



two EC2 regions (US East, US West)

Cassandra as a DBMS Storage Tier



- DBMS block per Cassandra record
- keyed by block ID
- CassandraI/O layer maps DBMS block requests to Cassandra read and write

Reading and Writing Data

- which consistency level should CassandraI/O use for each Cassandra read and write?

  `read(1),write(1)`:
  : fastest, but stale reads make DBMS very unhappy

  `read(ALL),write(1)`:
  : no stale reads, but slow reads and potential availability threat

  `read(1), write(ALL)`:
  : no stale reads, but slow writes

- can we approach the performance of `read(1),write(1)` while avoiding stale reads?

Optimisitic I/O

- observation: though Cassandra only guarantees eventual consistency, most reads see current data (why?)
- we can exploit this using an optimistic read/write protocol:
  - DBMS block write $\rightarrow$ Cassandra `write(1)`
  - DBMS block read $\rightarrow$ Cassandra `read(1)`, but check for stale data and recover if necessary
- how to check for stale data?
  - CassandraI/O stores a version number with each page, and remembers current version
  - on read, check version number of retrieved page against known current version
- how to recover from stale read?
  - aggressive: retry `read(1)`
  - conservative: `read(ALL)`
- optimization: remember version numbers for frequently read pages only, use `read(ALL)` to read others

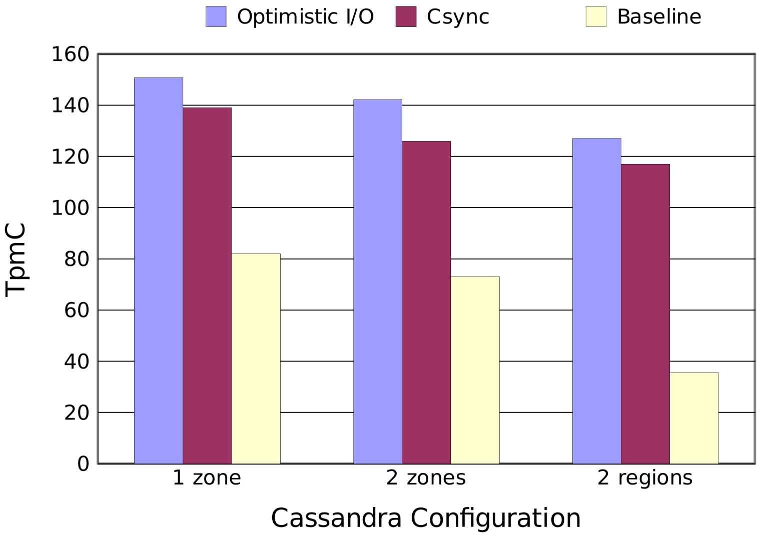
Cassandra Failures

- Cassandra will detect and recover from node failures
- are Cassandra failures transparent to hosted DBMS?
  - Optimistic I/O uses `write(1)`. Is the update really safe?
  - Optimistic I/O sometimes uses `read(ALL)`. This will block if any replica is down.
- we use client-controlled synchronization for better tolerance of Cassandra failures

Client-Controlled Synchronization

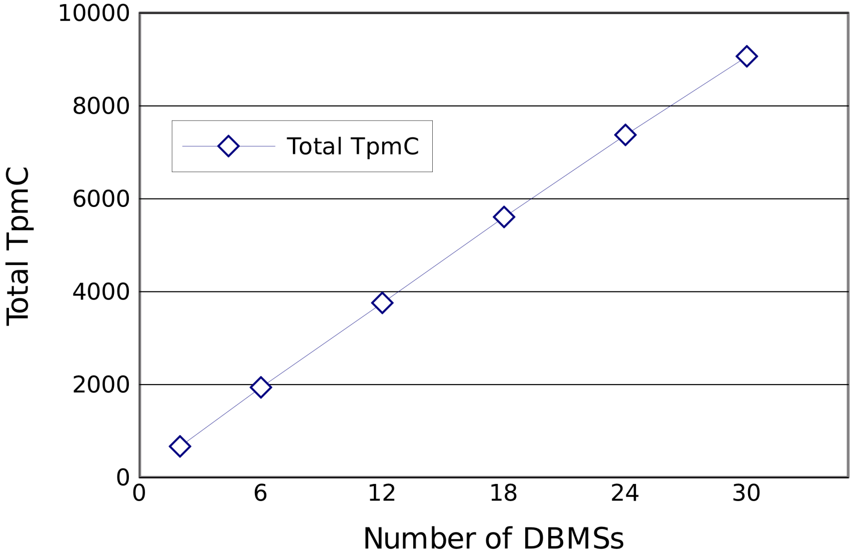
- DBMS (via CassandraI/O) uses `write(1)` plus new Cassandra `CSync()` operation
- `CSync()` ack means previous unsynchronized writes are performed on at least a quorum of replicas
- any delay between `write(1)` and `CSync()` allows synchronization latency hiding
- we can use `read(QUORUM)` instead of `read(ALL)` to read synchronized writes (better availability)
- DBMS is used to explicit synchronization (the file system made me do it!)

Does it Work?



Optimistic I/O
Csync
Baseline
160
140
120
100
80
60
40
20
0
TpmC
1 zone
2 zones
2 regions
Cassandra Configuration

Scalability



Cassandra Node Failure