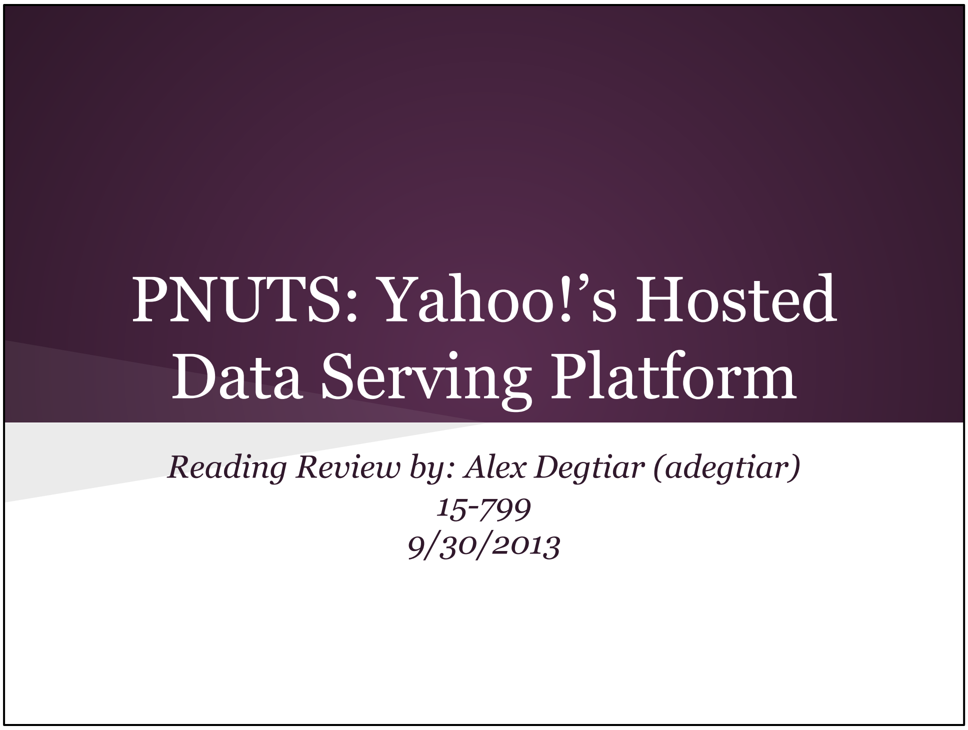
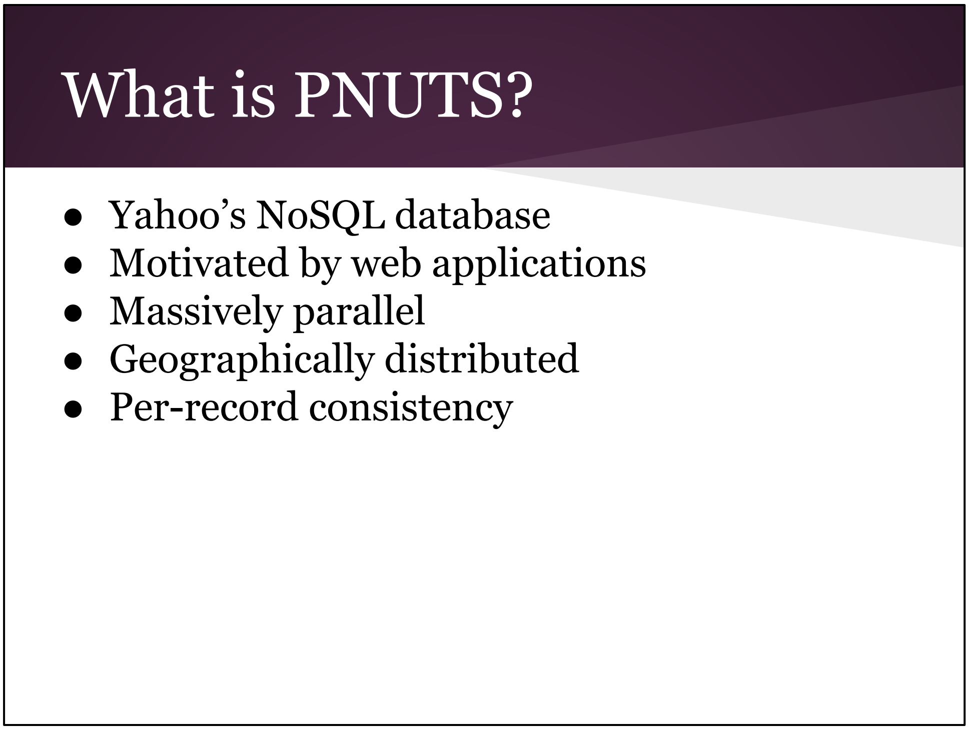


PNUTS: Yahoo!'s Hosted Data Serving Platform

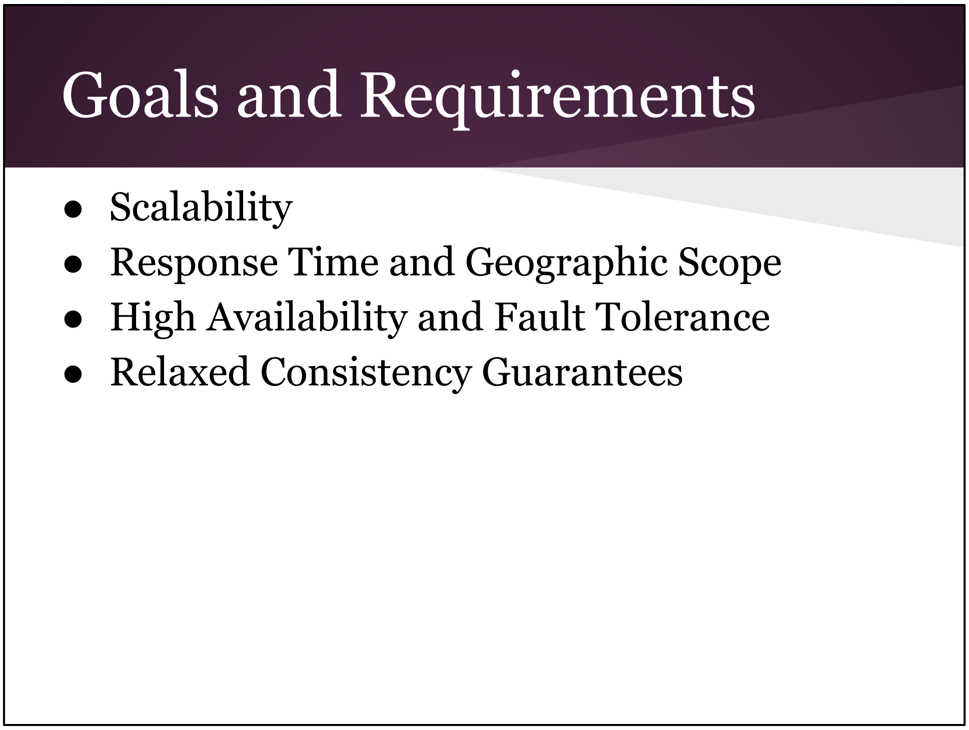
Reading Review by: Alex Degtiar (adegtiar)
15-799
9/30/2013



What is PNUTS?

- Yahoo’s NoSQL database
- Motivated by web applications
- Massively parallel
- Geographically distributed
- Per-record consistency

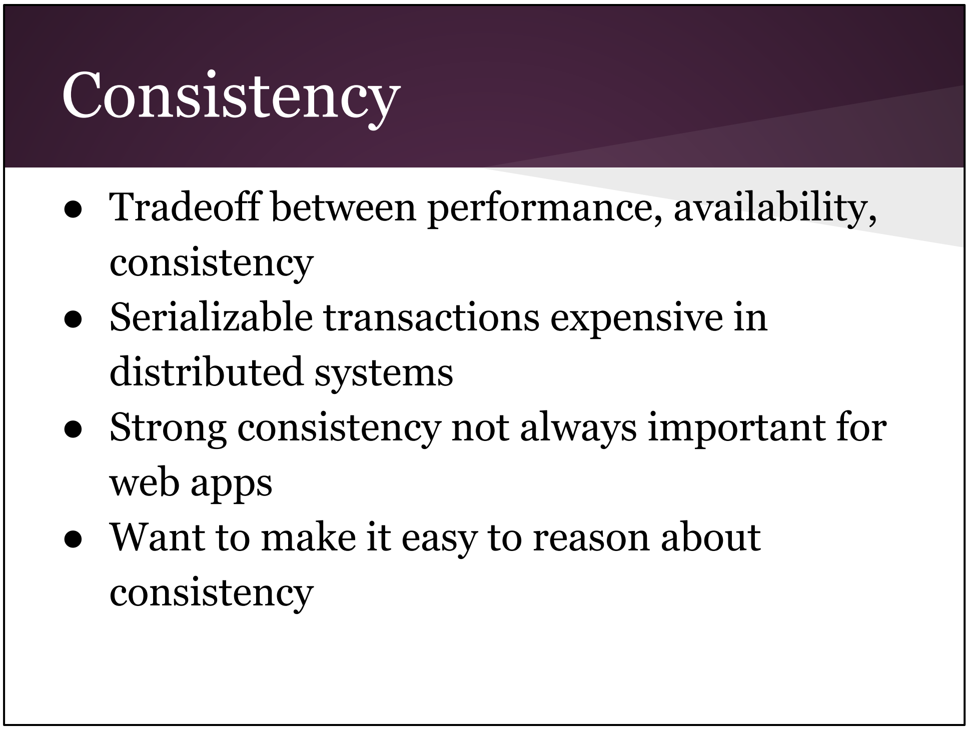
web apps, not complex queries



Goals and Requirements

- Scalability
- Response Time and Geographic Scope
- High Availability and Fault Tolerance
- Relaxed Consistency Guarantees

1. Scalability (architectural, handle periods of rapid growth)
2. Response Time and Geographic Scope (reads from nearby server -> low latency for users across the globe)
3. High Availability and Fault Tolerance (read & write availability, handle server failures, network partitions, power loss, etc))
4. Relaxed Consistency Guarantees

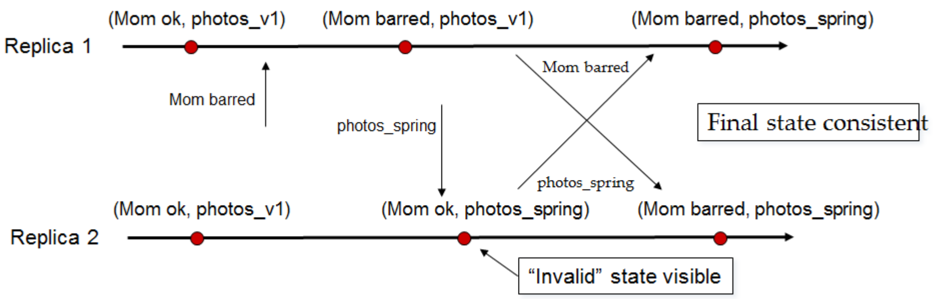


Consistency

- Tradeoff between performance, availability, consistency
- Serializable transactions expensive in distributed systems
- Strong consistency not always important for web apps
- Want to make it easy to reason about consistency

Eventual Consistency

- Updates to photo metadata on social site
  - U1: Remove his mother from the list of people who can view his photos
  - U2: Post spring-break photos



Replica 1

(Mom ok, photos_v1)

(Mom barred, photos_v1)

(Mom barred, photos_spring)

Mom barred

photos_spring

Mom barred

photos_spring

Final state consistent

Replica 2

(Mom ok, photos_v1)

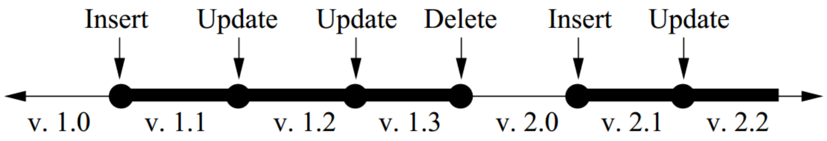
(Mom ok, photos_spring)

(Mom barred, photos_spring)

"Invalid" state visible

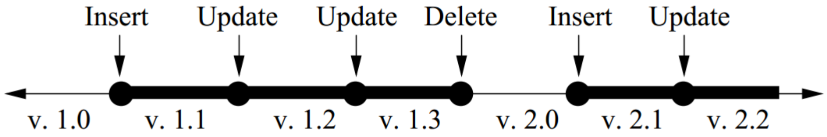
Per-record timeline consistency

- All replicas of a record apply record updates in same order



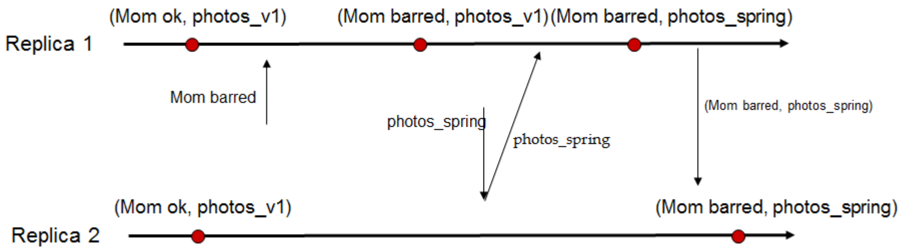
API and Specified Consistency

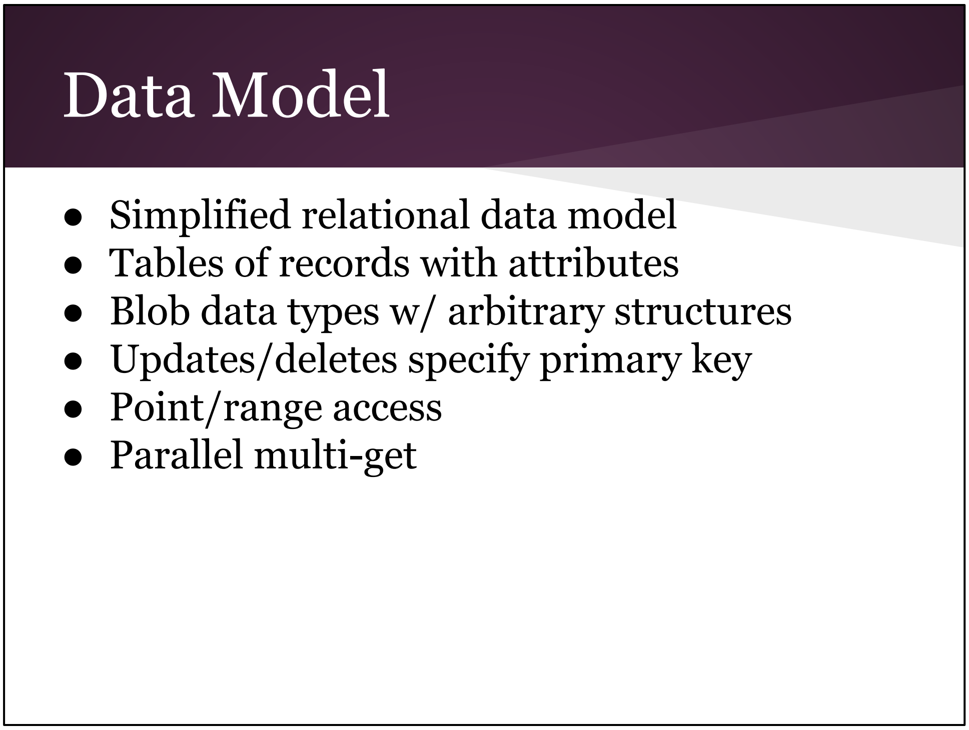
- Read-any
- Read-critical(>=version)
- Read-latest
- Write
- Test-and-set-write(version)



Per-Record Timeline Consistency example

- U1: Remove his mother from the list of people who can view his photos
- U2: Post spring-break photos

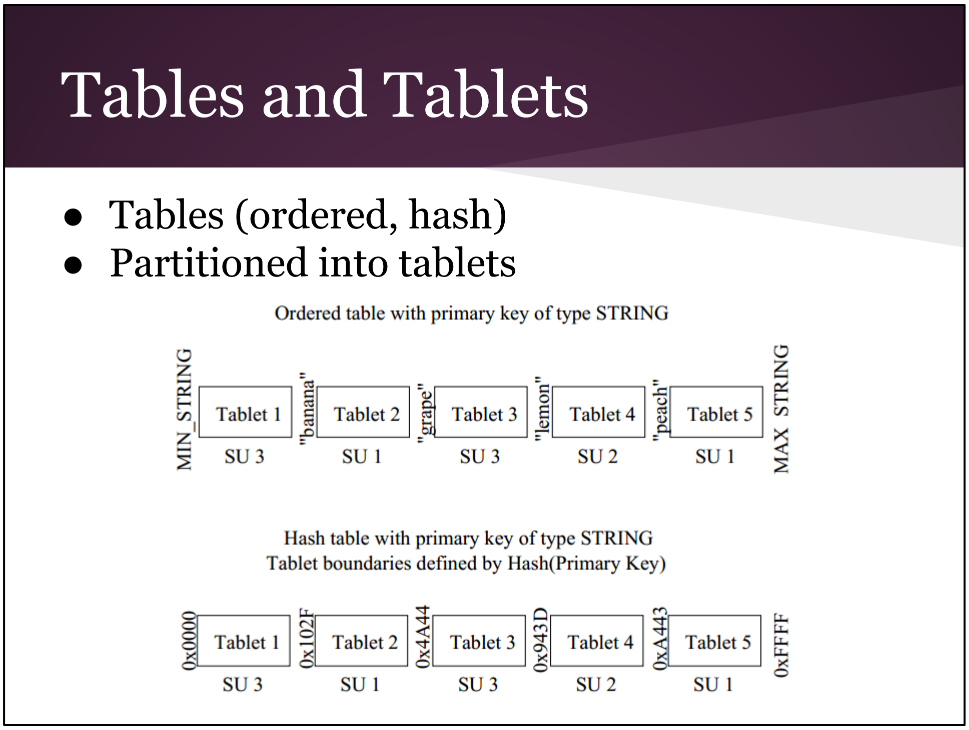




Data Model

- Simplified relational data model
- Tables of records with attributes
- Blob data types w/ arbitrary structures
- Updates/deletes specify primary key
- Point/range access
- Parallel multi-get

range has predicate
no complex queries, no constraint enforcement



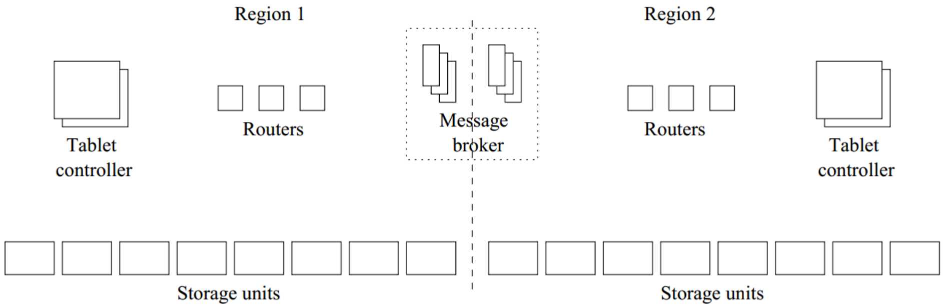
Tables and Tablets

- Tables (ordered, hash)
- Partitioned into tablets

Hash more efficient at load balancing

Architecture

- Regions with identical components



Region 1

Tablet controller

Routers

Message broker

Region 2

Routers

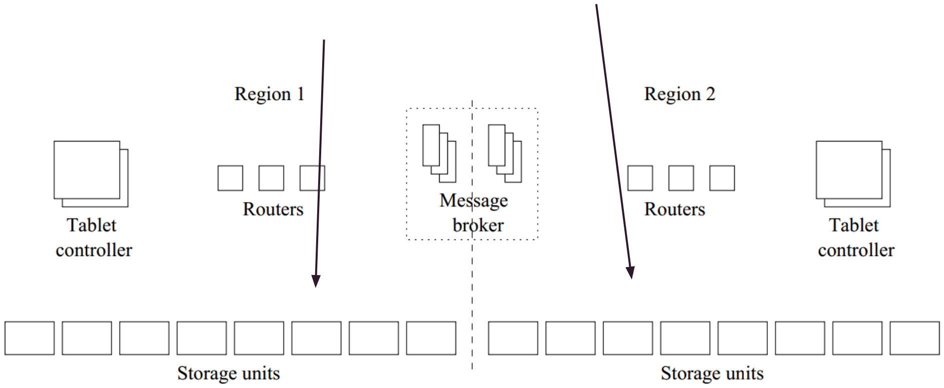
Tablet controller

Storage units

Storage units

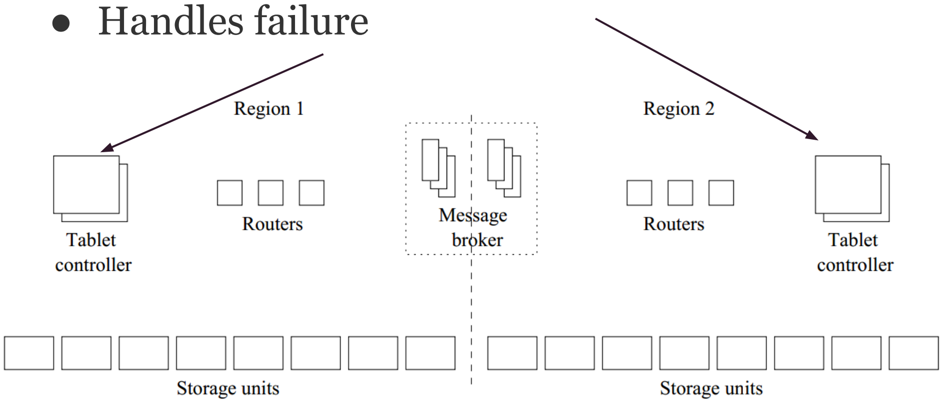
Storage Units

- Physical data storage nodes
- API: GET/SET/SCAN



Tablet Controller

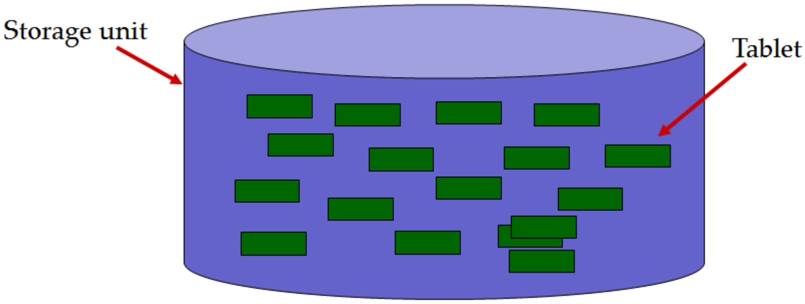
- Holds interval -> tablet mappings
- Remaps under load imbalance
- Handles failure



Tablet splitting and balancing

Each storage unit has many tablets (horizontal partitions of the table)

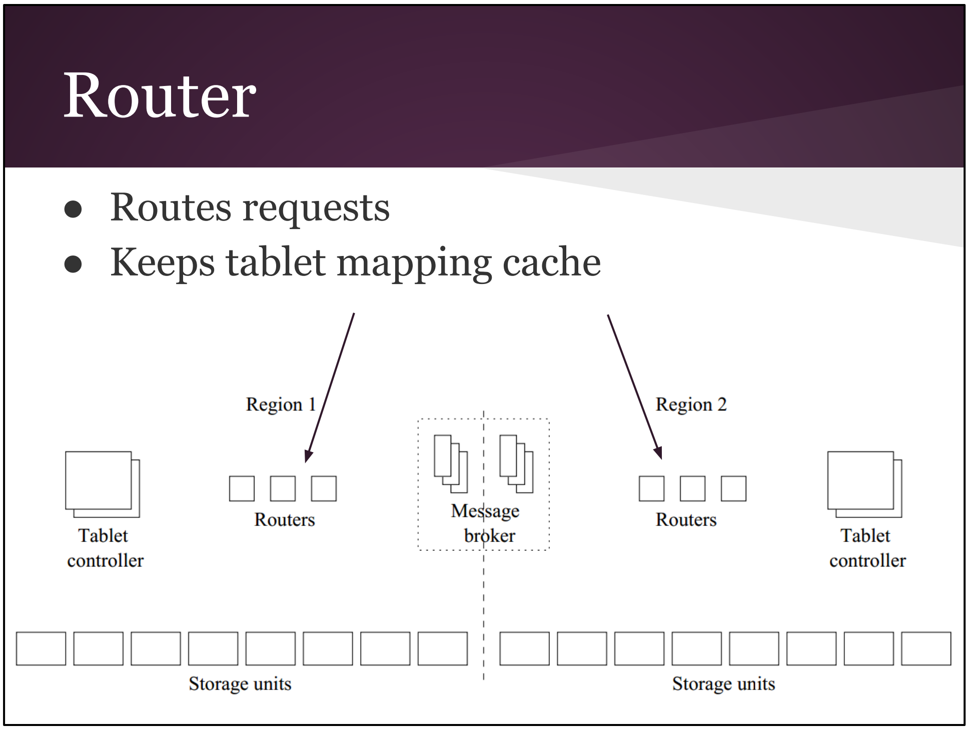
Storage unit may become a hotspot



Overfull tablets split

Tablets may grow over time

Shed load by moving tablets to other servers



Router

- Routes requests
- Keeps tablet mapping cache

Region 1

Region 2

Tablet controller

Routers

Message broker

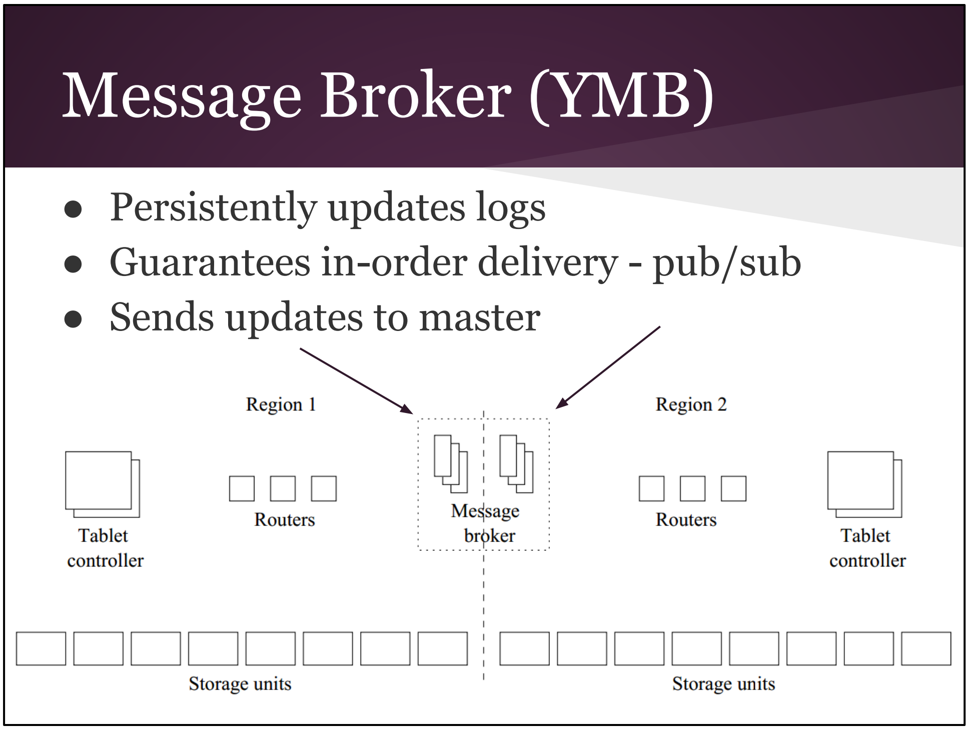
Routers

Tablet controller

Storage units

Storage units

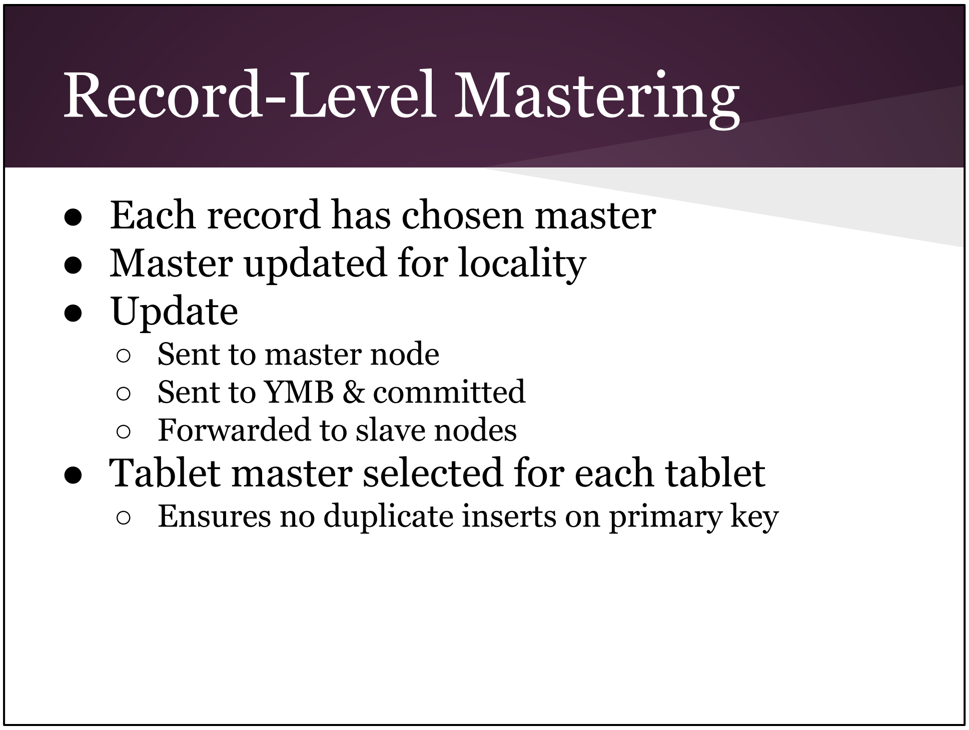
on error from SU, updates cache



Message Broker (YMB)

- Persistently updates logs
- Guarantees in-order delivery - pub/sub
- Sends updates to master

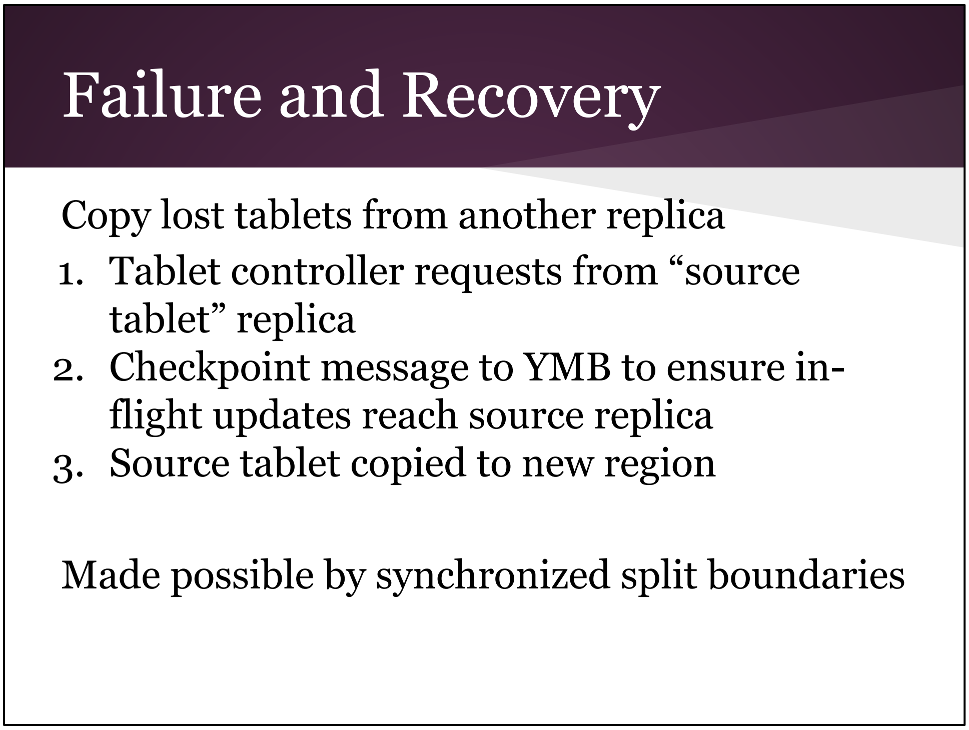
on error from SU, updates cache



Record-Level Mastering

- Each record has chosen master
- Master updated for locality
- Update
    - Sent to master node
    - Sent to YMB & committed
    - Forwarded to slave nodes
- Tablet master selected for each tablet
    - Ensures no duplicate inserts on primary key

~85% of reads/writes are with good locality/latency
history of 3 masters kept - if changing, relocate master.



Failure and Recovery

Copy lost tablets from another replica

1. Tablet controller requests from “source tablet” replica
2. Checkpoint message to YMB to ensure in-flight updates reach source replica
3. Source tablet copied to new region

Made possible by synchronized split boundaries

Other Features

- Scatter-gather engine
  - Part of router
  - Can support Top-K in range query
- Notifications
  - Pub/sub support via YMB
- Hosted database service
  - Balances capacity among added servers
  - Automatic recovery
  - Isolation between different workloads/applications (via different SU)

Experimental Results

- 1 router, 2 message brokers, 5 storage units
- High cost for inserts in non-master region

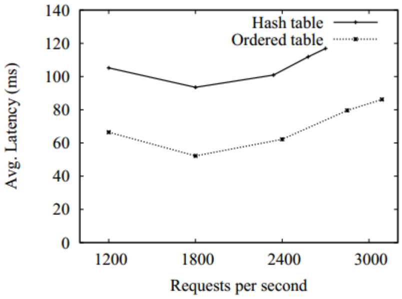
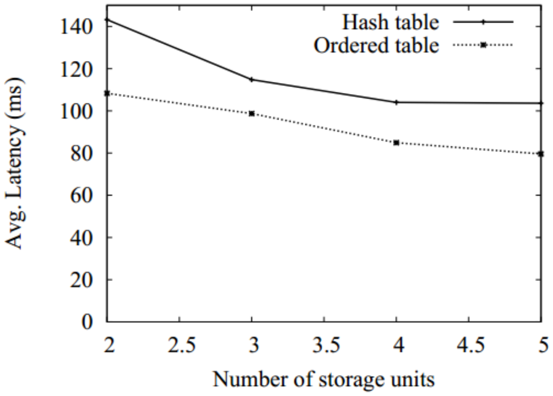


Figure 3: Impact of varying request rate on the average request latency.

More Experimental results

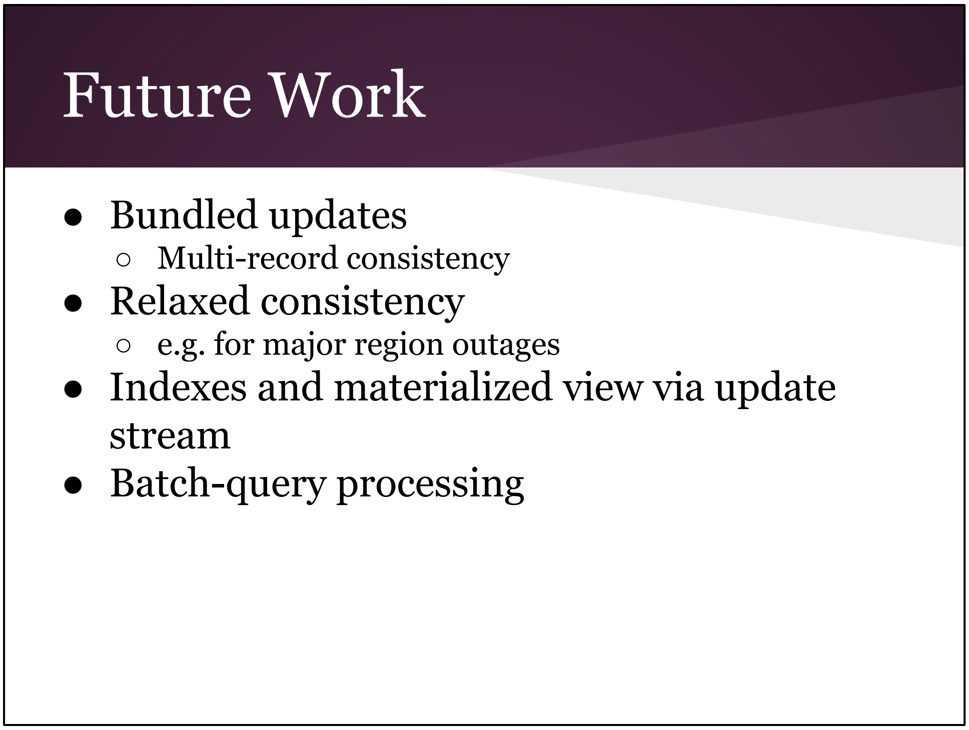


Limitations

- No multi-record transactions
- Record-level consistency forces use of same model for in-order updates
- Poor latency guarantees
  - Writes & consistent reads go to (possibly remote) master
- Optimized for read/write single records and small scans (tens or hundreds of records)

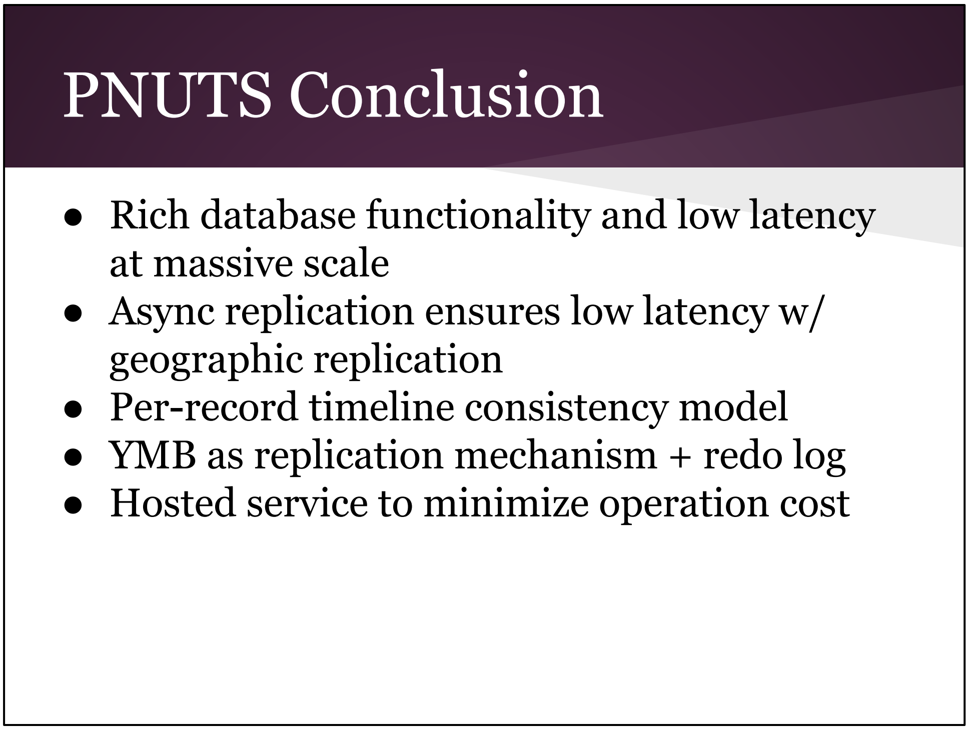
Other Criticisms

- Range scans don’t scale
- Slow/expensive failure recovery
- Unclear how YMB works/scales
- On-record-at-a-time consistency not always enough
- Experiment not very large scale
  - Is scale tested at all?
  - Ordered table not tested at scale… hot keys?



Future Work

- Bundled updates
  - Multi-record consistency
- Relaxed consistency
  - e.g. for major region outages
- Indexes and materialized view via update stream
- Batch-query processing



PNUTS Conclusion

- Rich database functionality and low latency at massive scale
- Async replication ensures low latency w/ geographic replication
- Per-record timeline consistency model
- YMB as replication mechanism + redo log
- Hosted service to minimize operation cost

Acknowledgements

- Information, figures, etc. **PNUTS: Yahoo!'s Hosted Data Serving Platform**, B. Cooper, et al.
- Consistency and tablet diagrams adapted/taken from Yahoo talk. http://www.slideshare.net/smilekg1220/pnuts-12502407.
- Relevant source overview to help understand the material: http://the-paper-trail.org/blog/yahoos-pnuts/.