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
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
• 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

Ordered table with primary key of type STRING

Hash table with primary key of type STRING
Tablet boundaries defined by Hash(Primary Key)

Hash more efficient at load balancing
Architecture

- Regions with identical components
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

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

![Graph showing comparison between Hash table and Ordered table with average latency (ms) against number of storage units. The graph indicates that the Hash table has a lower average latency compared to the Ordered table as the number of storage units increases.]
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)
<table>
<thead>
<tr>
<th>Other Criticisms</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Range scans don’t scale</td>
</tr>
<tr>
<td>● Slow/expensive failure recovery</td>
</tr>
<tr>
<td>● Unclear how YMB works/scales</td>
</tr>
<tr>
<td>● On-record-at-a-time consistency not always enough</td>
</tr>
<tr>
<td>● Experiment not very large scale</td>
</tr>
<tr>
<td>○ Is scale tested at all?</td>
</tr>
<tr>
<td>○ Ordered table not tested at scale... hot keys?</td>
</tr>
</tbody>
</table>
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](http://www.slideshare.net/smilekg1220/pnuts-12502407).
- Relevant source overview to help understand the material: [http://the-paper-trail.org/blog/yahoos-pnuts/](http://the-paper-trail.org/blog/yahoos-pnuts/).