PNUTS: Yahoo!’s Hosted Data Serving Platform

Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver and Ramana Yerneni
What is needed from current DBMS?

- Web applications need:
  - Scalability
  - Geographic scope
  - High availability

- Web applications typically have:
  - Simplified query needs
    - No joins, aggregations
  - Relaxed consistency needs
    - Applications can tolerate stale or reordered data
PNUTS

- PNUTS, a massively parallel and geographically distributed database system for Yahoo!’s web applications
Outline

- Architecture
- Experiment
- Future work
- Critique
Architecture

Clients

REST API

Tablet controller

Routers

Yahoo! Message Broker

Data-path components

Storage units
### Ordered table

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>1.2</td>
</tr>
<tr>
<td>Avocado</td>
<td>2.0</td>
</tr>
<tr>
<td>Banana</td>
<td>1.0</td>
</tr>
<tr>
<td>Grape</td>
<td>2.5</td>
</tr>
<tr>
<td>Kiwi</td>
<td>2.0</td>
</tr>
<tr>
<td>Orange</td>
<td>0.8</td>
</tr>
<tr>
<td>Strawberry</td>
<td>1.2</td>
</tr>
<tr>
<td>Watermelon</td>
<td>1.0</td>
</tr>
</tbody>
</table>
## Hash table

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>1.2</td>
</tr>
<tr>
<td>Avocado</td>
<td></td>
</tr>
<tr>
<td>Banana</td>
<td>1.0</td>
</tr>
<tr>
<td>Grape</td>
<td>2.5</td>
</tr>
<tr>
<td>Kiwi</td>
<td>2.0</td>
</tr>
<tr>
<td>Orange</td>
<td>0.8</td>
</tr>
<tr>
<td>Strawberry</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Flexible schema

- Arbitrary structures are allowed
- New attributes can be added at any time
  - Records are not required to have values for all attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Country</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Architecture

- **Routers**
  - Which table -> which tablet -> which SU
  - Contains a cached copy of the interval mapping

- **Tablet controller**
  - Owns the mapping
  - Decides when to move tablet, split tablet
  - Not a bottleneck
Tablet Splitting & Balancing

Tables: horizontally partitioned -> tablets

Storage unit may become a hotspot

Tablets may grow over time

Shed load by moving tablets to other servers
Architecture

Clients

REST API

Tablet controller

Routers

Yahoo! Message Broker

Storage units

Data-path components
Architecture - YMB

- Yahoo! Message Broker (YMB)
  - Topic-based pub/sub system
  - Data is considered ‘committed’ when they have been published to YMB.
  - Only partial ordering of published messages
Consistency model

- Per-record timeline consistency

- Per-record mastering
  - Each record is assigned a “master region”
  - May differ between records
  - Updates to the record forwarded to the master region
  - Ensures consistent ordering of updates
Consistency model - API

- Read-any
- Read-critical(required_version)
- Read-latest
- Write
- Test-and-set-write(required_version)
Consistency model

In general, reads are served using a local copy
Consistency model

- Read-critical $\geq$ v.6
- Stale version
- Stale version
- Current version

- Generation 1
- Insertion
- Update

Time
Consistency model

But application can request and get current version
Consistency model

Time

- v. 1
- v. 2
- v. 3
- v. 4
- v. 5
- v. 6
- v. 7
- v. 8

Generation 1

Write

- Stale version
- Stale version
- Current version

Insertion
Update
Consistency model

Test-and-set writes facilitate per-record transactions

Write if = v. 7

ERROR

Stale version

Current version

Insertion  Update
Record Timeline Consistency

Transactions:

- Alice changes status from “Sleeping” to “Awake”
- Alice changes location from “Home” to “Work”

Region 1

(Alice, Home, Sleeping)  ➔  (Alice, Home, Awake)  ➔  (Alice, Work, Awake)

Region 2

(Alice, Home, Sleeping)  ➔  (Alice, Work, Awake)

No replica should see record as (Alice, Work, Sleeping)
Eventual Consistency

- *Timeline consistency* comes at a price
  - Writes not originating in record master region forward to master and have longer latency
  - When master region down, record is unavailable for write

- We added *eventual consistency* mode
  - On conflict, latest write per field wins
  - Target customers
    - Those that externally guarantee no conflicts
    - Those that understand/can cope
Experimental setup

- Performance metric
  - Average request latency

- Three PNUTS regions

<table>
<thead>
<tr>
<th>Region</th>
<th>Machine</th>
<th>Servers/region</th>
</tr>
</thead>
<tbody>
<tr>
<td>West 1, West 2</td>
<td>2.8 GHz Xeon, 4GB RAM</td>
<td>5 SU 2 YMB 1 Router 1 Tablet controller</td>
</tr>
<tr>
<td>East</td>
<td>Quad 2.13 GHz Xeon</td>
<td></td>
</tr>
</tbody>
</table>

- Workload
  - 1200-3600 requests/second
  - 0-50% writes
  - 80% locality
Inserts

- Inserts

<table>
<thead>
<tr>
<th>Region</th>
<th>Time (hash table)</th>
<th>Time (ordered table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>West 1 (master)</td>
<td>75.6 75.6 ms</td>
<td>33 ms</td>
</tr>
<tr>
<td>West 2 (non-master)</td>
<td>131.5 ms</td>
<td>105.8 ms</td>
</tr>
<tr>
<td>East (non-master)</td>
<td>315.5 ms</td>
<td>324.5 ms</td>
</tr>
</tbody>
</table>
Write

Average latency (ms)

Requests per second

Hash table

Ordered table
Scalability

Average latency (ms)

Storage units

Hash table

Ordered table
Request skew

![Graph showing average latency (ms) vs. Zipf parameter. The graph compares Hash table and Ordered table performance. The y-axis represents the average latency in milliseconds, ranging from 0 to 100. The x-axis represents the Zipf parameter, ranging from 0 to 1. The Hash table line is green and the ordered table line is yellow. The graph shows that the ordered table performs better than the hash table as the Zipf parameter increases.]
Future work

- Indexes
  - Efficient query processing
- Bundled updates
- Batch-query processing
Critique

- Yahoo! Message Broker
  - Multiple YMBs in one region, how to coordinate?
  - The mechanism is rather complicated, scalability?
  - All writes go through it, bottleneck?
Critique

- Limited scope of the experiment
  - 5 storage units for each region
  - Testing scalability: ranging SU from 2 to 5
  - Only test latency, how about throughput?
  - Comparison with other data storage systems, like Cassandra
  - Seems High latency
Thanks.

Questions?