Cloud Data Serving: From Key-Value Stores to DBMSs

Raghu Ramakrishnan
Chief Scientist, Audience and Cloud Computing

Brian Cooper
Adam Silberstein
Utkarsh Srivastava

Yahoo! Research

Joint work with the Sherpa team in Cloud Computing
Outline

• Introduction
• Clouds
• Scalable serving—the new landscape
  – Very Large Scale Distributed systems (VLSD)
• Yahoo!’s PNUTS/Sherpa
• Comparison of several systems
Motherhood-and-Apple-Pie

CLOUDS
Why Clouds @Y!?

- Abstraction & Innovation
  - Developers focus on apps, not infrastructure
- Scale & Availability
  - Cloud services should do the heavy lifting

Demands of cloud storage have led to simplified KV stores
Cloud Services @Y!: Use Cases

- Search Index
- Ads Optimization
- Image/Video Storage & Delivery
- Attachment Storage
- Machine Learning (e.g. Spam filters)
- Content Optimization
- Video Storage & Delivery
- Ads
- Optimization
- Search
- Index
- Cloud Services @Y!: Use Cases

My Yahoo! | May 12, 2009
Requirements for Cloud Services

- **Multitenant.** A cloud service must support multiple, organizationally distant customers.
- **Elasticity.** Tenants should be able to negotiate and receive resources/QoS *on-demand* up to a large scale.
- **Resource Sharing.** Ideally, spare cloud resources should be transparently applied when a tenant’s negotiated QoS is insufficient, e.g., due to spikes.
- **Horizontal scaling.** The cloud provider should be able to add cloud capacity in increments without affecting tenants of the service.
- **Metering.** A cloud service must support accounting that reasonably ascribes operational and capital expenditures to each of the tenants of the service.
- **Security.** A cloud service should be secure in that tenants are not made vulnerable because of loopholes in the cloud.
- **Availability.** A cloud service should be highly available.
- **Operability.** A cloud service should be easy to operate, with few operators. Operating costs should scale linearly or better with the capacity of the service.
Yahoo! Cloud Stack

EDGE
- YCS
- YCPI
- Brooklyn
- ...

WEB
- VM/OS
- yApache
- PHP
- App Engine

APP
- VM/OS
- Serving Grid
- ...

OPERATIONAL STORAGE
- PNUTS/Sherpa
- MOBStor
- ...

BATCH STORAGE
- Hadoop
- ...

Provisioning (Self-serve)
Monitoring/Metering/Security

Data Highway
Data Serving in the Y! Cloud

FredList.com application

1234323, transportation, For sale: one bicycle, barely used

5523442, childcare, Nanny available in San Jose

32138, camera, Nikon D40, USD 300

DECLARE DATASET Listings AS (ID String PRIMARY KEY, Category String, Description Text)

ALTER Listings MAKE CACHEABLE

Simple Web Service API’s

Hadoop
Compute

PNUTS / SHERPA
Database

MObStor
Storage

Vespa
Search

memcached
Caching

Batch export

Foreign key
photo → listing

Tribble
Messaging
New in 2010!

- SIGMOD and SIGOPS are starting a new annual conference, to be co-located alternately with SIGMOD and SOSP:
  
  **ACM Symposium on Cloud Computing (SoCC)**
  
  **PC Chairs:** Surajit Chaudhuri & Mendel Rosenblum

- Steering committee: Phil Bernstein, Ken Birman, Joe Hellerstein, John Ousterhout, Raghu Ramakrishnan, Doug Terry, John Wilkes
Renting vs. buying, and being DBA to the world …

DATA MANAGEMENT IN THE CLOUD
Help!

• I have a huge amount of data. What should I do with it?
What Are You Trying to Do?

Data Workloads

OLTP
(Random access to a few records)
Read-heavy
Write-heavy
By rows

OLAP
(Scan access to a large number of records)
By columns
Unstructured

Combined
(Some OLTP and OLAP tasks)
Web Data Management

- Warehousing
- Scan oriented workloads
- Focus on sequential disk I/O
- $ per cpu cycle

**Large data analysis (Hadoop)**

- CRUD
- Point lookups and short scans
- Index organized table and random I/Os
- $ per latency

**Structured record storage (PNUTS/Sherpa)**

- Object retrieval and streaming
- Scalable file storage
- $ per GB storage & bandwidth

**Blob storage (MObStor)**

- Scan oriented workloads
- Focus on sequential disk I/O
- $ per cpu cycle
Ways of Using Hadoop

Data workloads

OLAP
(Scan access to a large number of records)

By rows

By columns

Unstructured

HadoopDB

SQL on Grid

Hive

Zebra
<table>
<thead>
<tr>
<th>Application</th>
<th>2008</th>
<th>2009</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Webmap</strong></td>
<td>~70 hours runtime</td>
<td>~73 hours runtime</td>
</tr>
<tr>
<td></td>
<td>~300 TB shuffling</td>
<td>~490 TB shuffling</td>
</tr>
<tr>
<td></td>
<td>~200 TB output</td>
<td>~280 TB output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+55% hardware</td>
</tr>
<tr>
<td><strong>Terasort</strong></td>
<td>209 seconds</td>
<td>62 seconds</td>
</tr>
<tr>
<td></td>
<td>1 Terabyte sorted</td>
<td>1 Tera byte, 1500 nodes</td>
</tr>
<tr>
<td></td>
<td>900 nodes</td>
<td>16.25 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Petabyte, 3700 nodes</td>
</tr>
<tr>
<td><strong>Largest cluster</strong></td>
<td>2000 nodes</td>
<td>4000 nodes</td>
</tr>
<tr>
<td></td>
<td>• 6PB raw disk</td>
<td>• 16PB raw disk</td>
</tr>
<tr>
<td></td>
<td>• 16TB of RAM</td>
<td>• 64TB of RAM</td>
</tr>
<tr>
<td></td>
<td>• 16K CPUs</td>
<td>• 32K CPUs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• (40% faster CPUs too)</td>
</tr>
</tbody>
</table>
ACID or BASE? Litmus tests are colorful, but the picture is cloudy

SCALABLE DATA SERVING
“I want a big, virtual database”

“What I want is a robust, high performance virtual relational database that runs transparently over a cluster, nodes dropping in and out of service at will, read-write replication and data migration all done automatically.

I want to be able to install a database on a server cloud and use it like it was all running on one machine.”

-- Greg Linden’s blog
The World Has Changed

• Web serving applications need:
  – Scalability!
    • Preferably elastic, commodity boxes
  – Flexible schemas
  – Geographic distribution
  – High availability
  – Low latency

• Web serving applications willing to do without:
  – Complex queries
  – ACID transactions
Yahoo! Serving Storage Problem

- Small records – 100KB or less
- Structured records – Lots of fields, evolving
- Extreme data scale - Tens of TB
- Extreme request scale - Tens of thousands of requests/sec
- Low latency globally - 20+ datacenters worldwide
- High Availability - Outages cost $millions
- Variable usage patterns - Applications and users change
Typical Y! Applications

• User logins and profiles
  – Including changes that must not be lost!
    • But single-record “transactions” suffice

• Events
  – Alerts (e.g., news, price drops)
  – Social network activity (e.g., user goes offline)
  – Ad clicks, article clicks

• Application-specific data
  – Postings in message board
  – Uploaded photos, tags
  – Shopping carts

500M+ unique users per month
Hundreds of petabytes of storage
Hundreds of billions of objects
Hundred of thousands of requests/sec
Global, rapidly evolving workloads
VLSD Data Serving Stores

- Must **partition** data across machines
  - How are partitions determined?
  - Can partitions be changed easily? (Affects **elasticity**)
  - How are read/update requests routed?
  - **Range selections**? Can requests span machines?
- **Availability**: What failures are handled?
  - With what semantic guarantees on data access?
- (How) Is data **replicated**?
  - Sync or async? Consistency model? Local or geo?
- How are updates made **durable**?
- How is data stored on a single machine?
The CAP Theorem

• You have to give up one of the following in a distributed system (Brewer, PODC 2000; Gilbert/Lynch, SIGACT News 2002):
  – Consistency of data
    • Think serializability
  – Availability
    • Pinging a live node should produce results
  – Partition tolerance
    • Live nodes should not be blocked by partitions
Approaches to CAP

• “BASE”
  – No ACID, use a single version of DB, reconcile later
• Defer transaction commit
  – Until partitions fixed and distr xact can run
• Eventual consistency (e.g., Amazon Dynamo)
  – Eventually, all copies of an object converge
• Restrict transactions (e.g., Sharded MySQL)
  – 1-M/c Xacts: Objects in xact are on the same machine
  – 1-Object Xacts: Xact can only read/write 1 object
• Object timelines (PNUTS)

http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
What is PNUTS/Sherpa?

CREATE TABLE Parts (ID VARCHAR, StockNumber INT, Status VARCHAR, ...)

Structured, flexible schema

Parallel database

Geographic replication

Hosted, managed infrastructure
What Will It Become?

Indexes and views

A 42342 E
B 42521 W
C 66354 W
D 12352 E
E 75656 C
F 15677 E

A 42342 E
B 42521 W
C 66354 W
D 12352 E
E 75656 C
F 15677 E
Technology Elements

PNUTS
- Query planning and execution
- Index maintenance

Distributed infrastructure for tabular data
- Data partitioning
- Update consistency
- Replication

YDOT FS
- Ordered tables

YDHT FS
- Hash tables

Tribble
- Pub/sub messaging

Zookeeper
- Consistency service
PNUTS: Key Components

- Caches the maps from the TC
- Routes client requests to correct SU
- Stores records
- Services get/set/delete requests
- Maintains map from database.table.key-to-tablet-to-SU
- Provides load balancing
Detailed Architecture

**Local region**
- Clients
- Tablet Controller
- Routers
- REST API
- Storage units

**Remote regions**
- Tribble
- Storage units
DATA MODEL
Data Manipulation

• Per-record operations
  – Get
  – Set
  – Delete

• Multi-record operations
  – Multiget
  – Scan
  – Getrange

• Web service (RESTful) API
## Tablets—Hash Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grape</td>
<td>Grapes are good to eat</td>
<td>$12</td>
</tr>
<tr>
<td>Lime</td>
<td>Limes are green</td>
<td>$9</td>
</tr>
<tr>
<td>Apple</td>
<td>Apple is wisdom</td>
<td>$1</td>
</tr>
<tr>
<td>Strawberry</td>
<td>Strawberry shortcake</td>
<td>$900</td>
</tr>
<tr>
<td>Orange</td>
<td>Arrgh! Don’t get scurvy!</td>
<td>$2</td>
</tr>
<tr>
<td>Avocado</td>
<td>But at what price?</td>
<td>$3</td>
</tr>
<tr>
<td>Lemon</td>
<td>How much did you pay for this lemon?</td>
<td>$1</td>
</tr>
<tr>
<td>Tomato</td>
<td>Is this a vegetable?</td>
<td>$14</td>
</tr>
<tr>
<td>Banana</td>
<td>The perfect fruit</td>
<td>$2</td>
</tr>
<tr>
<td>Kiwi</td>
<td>New Zealand</td>
<td>$8</td>
</tr>
</tbody>
</table>
# Tablets—Ordered Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>Apple is wisdom</td>
<td>$1</td>
</tr>
<tr>
<td>Avocado</td>
<td>But at what price?</td>
<td>$3</td>
</tr>
<tr>
<td>Banana</td>
<td>The perfect fruit</td>
<td>$2</td>
</tr>
<tr>
<td>Grape</td>
<td>Grapes are good to eat</td>
<td>$12</td>
</tr>
<tr>
<td>Kiwi</td>
<td>New Zealand</td>
<td>$8</td>
</tr>
<tr>
<td>Lemon</td>
<td>How much did you pay for this lemon?</td>
<td>$1</td>
</tr>
<tr>
<td>Lime</td>
<td>Limes are green</td>
<td>$9</td>
</tr>
<tr>
<td>Orange</td>
<td>Arrgh! Don’t get scurvy!</td>
<td>$2</td>
</tr>
<tr>
<td>Strawberry</td>
<td>Strawberry shortcake</td>
<td>$900</td>
</tr>
<tr>
<td>Tomato</td>
<td>Is this a vegetable?</td>
<td>$14</td>
</tr>
</tbody>
</table>
### Flexible Schema

<table>
<thead>
<tr>
<th>Posted date</th>
<th>Listing id</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/1/07</td>
<td>424252</td>
<td>Couch</td>
<td>$570</td>
</tr>
<tr>
<td>6/1/07</td>
<td>763245</td>
<td>Bike</td>
<td>$86</td>
</tr>
<tr>
<td>6/3/07</td>
<td>211242</td>
<td>Car</td>
<td>$1123</td>
</tr>
<tr>
<td>6/5/07</td>
<td>421133</td>
<td>Lamp</td>
<td>$15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Color</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Good</td>
</tr>
<tr>
<td>Red</td>
<td>Fair</td>
</tr>
</tbody>
</table>
Primary vs. Secondary Access

### Primary table

<table>
<thead>
<tr>
<th>Posted date</th>
<th>Listing id</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/1/07</td>
<td>424252</td>
<td>Couch</td>
<td>$570</td>
</tr>
<tr>
<td>6/1/07</td>
<td>763245</td>
<td>Bike</td>
<td>$86</td>
</tr>
<tr>
<td>6/3/07</td>
<td>211242</td>
<td>Car</td>
<td>$1123</td>
</tr>
<tr>
<td>6/5/07</td>
<td>421133</td>
<td>Lamp</td>
<td>$15</td>
</tr>
</tbody>
</table>

### Secondary index

<table>
<thead>
<tr>
<th>Price</th>
<th>Posted date</th>
<th>Listing id</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>6/5/07</td>
<td>421133</td>
</tr>
<tr>
<td>86</td>
<td>6/1/07</td>
<td>763245</td>
</tr>
<tr>
<td>570</td>
<td>6/1/07</td>
<td>424252</td>
</tr>
<tr>
<td>1123</td>
<td>6/3/07</td>
<td>211242</td>
</tr>
</tbody>
</table>

Planned functionality
Index Maintenance

• How to have lots of interesting indexes and views, without killing performance?

• Solution: Asynchrony!
  – Indexes/views updated asynchronously when base table updated
PROCESSING READS & UPDATES
Updates

1. Write key k
2. Write key k
3. Write key k
4. Write key k
5. SUCCESS
6. Write key k
7. Sequence # for key k
8. Sequence # for key k

Routers

Message brokers
Accessing Data

1. Get key k
2. Get key k
3. Record for key k
4. Record for key k

SU

SU

SU
Range Queries in YDOT

• Clustered, ordered retrieval of records
Bulk Load in YDOT

• YDOT bulk inserts can cause performance hotspots

• Solution: preallocate tablets
ASYNCHRONOUS REPLICATION AND CONSISTENCY
Asynchronous Replication
Consistency Model

- If copies are asynchronously updated, what can we say about stale copies?
  - ACID guarantees require synchronous updates
  - Eventual consistency: Copies can drift apart, but will eventually converge if the system is allowed to quiesce
    - To what value will copies converge?
    - Do systems ever “quiesce”?
  - Is there any middle ground?
### Example: Social Alice

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Busy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Busy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Free</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>___</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Free</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>???</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>???</td>
</tr>
</tbody>
</table>

#### Record Timeline
- Busy
- Free
- Free
• Goal: Make it easier for applications to reason about updates and cope with asynchrony

• What happens to a record with primary key “Alice”?

As the record is updated, copies may get out of sync.
In general, reads are served using a local copy
PNUTS Consistency Model

But application can request and get current version
Or variations such as “read forward”—while copies may lag the master record, every copy goes through the same sequence of changes
Achieved via per-record primary copy protocol
(To maximize availability, record masterships automatically transferred if site fails)
Can be selectively weakened to eventual consistency
(local writes that are reconciled using version vectors)
PNUTS Consistency Model

Test-and-set writes facilitate per-record transactions
OPERABILITY
# Distribution

<table>
<thead>
<tr>
<th>Date</th>
<th>ID</th>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/1/07</td>
<td>256623</td>
<td>Car</td>
<td>$1123</td>
</tr>
<tr>
<td>6/2/07</td>
<td>636353</td>
<td>Bike</td>
<td>$86</td>
</tr>
<tr>
<td>6/5/07</td>
<td>662113</td>
<td>Chair</td>
<td>$10</td>
</tr>
<tr>
<td>6/7/07</td>
<td>121113</td>
<td>Lamp</td>
<td>$19</td>
</tr>
<tr>
<td>6/9/07</td>
<td>887734</td>
<td>Bike</td>
<td>$56</td>
</tr>
<tr>
<td>6/11/07</td>
<td>252111</td>
<td>Scooter</td>
<td>$18</td>
</tr>
<tr>
<td>6/11/07</td>
<td>116458</td>
<td>Hammer</td>
<td>$8000</td>
</tr>
</tbody>
</table>

- **Data shuffling for load balancing**
Tablet Splitting and Balancing

Each storage unit has many tablets (horizontal partitions of the table) and overfull tablets split, which may cause the storage unit to become a hotspot. Tablets may grow over time, leading to the need to shed load by moving tablets to other servers.
Consistency Techniques

- **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

- **Tablet-level mastering**
  - Each tablet is assigned a “master region”
  - Inserts and deletes of records forwarded to the master region
  - Master region decides tablet splits

- **These details are hidden from the application**
  - Except for the latency impact!
Record versus tablet master

Record master serializes updates

Tablet master serializes inserts
Coping With Failures

<table>
<thead>
<tr>
<th></th>
<th>42342</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>42521</td>
<td>W</td>
</tr>
<tr>
<td>C</td>
<td>66354</td>
<td>W</td>
</tr>
<tr>
<td>D</td>
<td>12352</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>75656</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>15677</td>
<td>E</td>
</tr>
</tbody>
</table>

OVERRIDE W → E

<table>
<thead>
<tr>
<th></th>
<th>42342</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>42521</td>
<td>W</td>
</tr>
<tr>
<td>C</td>
<td>66354</td>
<td>W</td>
</tr>
<tr>
<td>D</td>
<td>12352</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>75656</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>15677</td>
<td>E</td>
</tr>
</tbody>
</table>
Further PNutty Reading

Efficient Bulk Insertion into a Distributed Ordered Table (SIGMOD 2008)
Adam Silberstein, Brian Cooper, Utkarsh Srivastava, Erik Vee,
Ramana Yerneni, Raghu Ramakrishnan

PNUTS: Yahoo!'s Hosted Data Serving Platform (VLDB 2008)
Brian Cooper, Raghu Ramakrishnan, Utkarsh Srivastava,
Adam Silberstein, Phil Bohannon, Hans-Arno Jacobsen,
Nick Puz, Daniel Weaver, Ramana Yerneni

Asynchronous View Maintenance for VLSD Databases (SIGMOD 2009)
Parag Agrawal, Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava and
Raghu Ramakrishnan

Cloud Storage Design in a PNUTShell
Brian F. Cooper, Raghu Ramakrishnan, and Utkarsh Srivastava
Beautiful Data, O'Reilly Media, 2009

Adaptively Parallelizing Distributed Range Queries (VLDB 2009)
Ymir Vigfusson, Adam Silberstein, Brian Cooper, Rodrigo Fonseca
COMPARING SOME CLOUD SERVING STORES

Green Apples and Red Apples
Motivation

• Many “cloud DB” and “nosql” systems out there
  – PNUTS
  – BigTable
    • HBase, Hypertable, HTable
  – Azure
  – Cassandra
  – Megastore
  – Amazon Web Services
    • S3, SimpleDB, EBS
  – And more: CouchDB, Voldemort, etc.

• How do they compare?
  – Feature tradeoffs
  – Performance tradeoffs
  – Not clear!
The Contestants

- **Baseline**: Sharded MySQL
  - Horizontally partition data among MySQL servers

- **PNUTS/Sherpa**
  - Yahoo!’s cloud database

- **Cassandra**
  - BigTable + Dynamo

- **HBase**
  - BigTable + Hadoop
SHARDED MYSQL
• Our own implementation of sharding
Pros and Cons

• Pros
  – Simple
  – “Infinitely” scalable
  – Low latency
  – Geo-replication

• Cons
  – Not elastic (Resharding is hard)
  – Poor support for load balancing
  – Failover? (Adds complexity)
  – Replication unreliable (Async log shipping)
Azure SDS

- Cloud of SQL Server instances
- App partitions data into instance-sized pieces
  - Transactions and queries within an instance

[Diagram showing data flow from Data to SDS Instance, with Storage and Per-field indexing layers]
Google MegaStore

• Transactions across entity groups
  – Entity-group: hierarchically linked records
    • Ramakris
    • Ramakris.preferences
    • Ramakris.posts
    • Ramakris.posts.aug-24-09
  – Can transactionally update multiple records within an entity group
    • Records may be on different servers
    • Use Paxos to ensure ACID, deal with server failures
  – Can join records within an entity group

• Other details
  – Built on top of BigTable
  – Supports schemas, column partitioning, some indexing
PNUTS
Architecture

Clients

REST API

Tablet controller

Log servers

Routers

Storage units
Pros and Cons

• Pros
  – Reliable geo-replication
  – Scalable consistency model
  – Elastic scaling
  – Easy load balancing

• Cons
  – System complexity relative to sharded MySQL to support geo-replication, consistency, etc.
  – Latency added by router
HBASE
Architecture
HRegion Server

- Records partitioned into HStores
  - Each HStore contains many MapFiles
- All writes to HStore applied to single memcache
- Reads consult MapFiles and memcache
- Memcaches flushed as MapFiles (HDFS files) when full
- Compactions limit number of MapFiles
Pros and Cons

• Pros
  – High write throughput
  – Elastic scaling
  – Easy load balancing
  – Column storage for OLAP workloads

• Cons
  – Writes not immediately persisted to disk
  – Reads cross multiple disk, memory locations
  – No geo-replication
  – Latency/bottleneck of HBaseMaster when using REST
CASSANDRA
• Facebook’s storage system
  - BigTable data model
  - Dynamo partitioning and consistency model
  - Peer-to-peer architecture
• Consistent hashing, like Dynamo or Chord
  – Server position = hash(serverid)
  – Content position = hash(contentid)
  – Server responsible for all content in a hash interval
Cassandra Server

- Writes go to log and memory table
- Periodically memory table merged with disk table
Pros and Cons

• Pros
  – Elastic scalability
  – Easy management
    • Peer-to-peer configuration
  – BigTable model is nice
    • Flexible schema, column groups for partitioning, versioning, etc.
  – Eventual consistency is scalable

• Cons
  – Eventual consistency is hard to program against
  – No built-in support for geo-replication
    • Gossip can work, but is not really optimized for, cross-datacenter
  – Load balancing?
    • Consistent hashing limits options
  – System complexity
    • P2P systems are complex; have complex corner cases
NUMBERS
Overview

• Setup
  – Six server-class machines
    • 8 cores (2 x quadcore) 2.5 GHz CPUs
    • 8 GB RAM
    • 6 x 146GB 15K RPM SAS drives in RAID 1+0
    • Gigabit ethernet
    • RHEL 4
  – Plus extra machines for clients, routers, controllers, etc.

• Workloads
  – 120 million 1 KB records = 20 GB per server
  – Read heavy workload: 95/5 read/update
  – Updates write a single field
  – 100 or more client threads
  – Obviously many variations are possible; these are just two points in the space

• Metrics
  – Latency versus throughput curves

• Caveats
  – Write performance would be improved for Sherpa, Sharded and Cassandra with a dedicated log disk
  – We tuned each system as well as we knew how, with assistance from the team of developers
Update Latency

![Update Latency Graph](image-url)

The graph shows the average update latency (ms) against throughput (ops/sec) for different databases.

- **HBase**
- **Cassandra**
- **PNUTS**
- **Sharded MySQL**

The x-axis represents throughput (ops/sec) ranging from 0 to 9000, while the y-axis shows average update latency (ms) ranging from 0 to 40 ms.

The graph indicates that HBase has the lowest latency, followed by Cassandra, PNUTS, and Sharded MySQL, which has the highest latency.

As throughput increases, the latency for all databases increases as well, but at different rates.
Write-Heavy Workloads

Read performance

Average read latency (ms)

Percentage of operations that are updates

- Cassandra
- Hbase
- PNUTS
- Sharded
Qualitative Comparison

• Storage Layer
  – File Based: HBase, Cassandra
  – MySQL: PNUTS, Sharded MySQL

• Write Persistence
  – Writes committed synchronously to disk: PNUTS, Cassandra, Sharded MySQL
  – Writes flushed asynchronously to disk: HBase (current version)

• Read Pattern
  – Find record in MySQL (disk or buffer pool): PNUTS, Sharded MySQL
  – Find record and deltas in memory and on disk: HBase, Cassandra
Qualitative Comparison

• Replication (not yet utilized in benchmarks)
  – Intra-region: HBase, Cassandra
  – Inter- and intra-region: PNUTS
  – Inter- and intra-region: MySQL (but not guaranteed)

• Mapping record to server
  – Router: PNUTS, HBase (with REST API)
  – Client holds mapping: HBase (java library), Sharded MySQL
  – P2P: Cassandra
SYSTEMS IN CONTEXT
Types of Record Stores

• Query expressiveness

Simple

S3

Object retrieval

PNUTS

Retrieval from single table of objects/records

Oracle

SQL

Feature rich
Types of Record Stores

- Consistency model

- S3 (Best effort) - Eventual consistency
- PNUTS - Timeline consistency
- Oracle - ACID guarantees
- Object-centric consistency
- Program centric consistency

Strong guarantees
Types of Record Stores

- Data model

- **PNUTS**
  - Flexibility,
  - Schema evolution

- **CouchDB**
  - Object-centric consistency

- **Oracle**
  - Consistency spans objects
  - Optimized for
  - Fixed schemas
Types of Record Stores

- Elasticity (ability to add resources on demand)

<table>
<thead>
<tr>
<th>Inelastic</th>
<th>Elastic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limited (via data distribution)</td>
<td>VLSD (Very Large Scale Distribution/Replication)</td>
</tr>
</tbody>
</table>

- Oracle

- PNUTS

- S3

- Elasticity Range
<table>
<thead>
<tr>
<th>Data Stores Comparison</th>
<th>Versus PNUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User-partitioned SQL stores</strong></td>
<td><strong>More expressive queries</strong></td>
</tr>
<tr>
<td>– Microsoft Azure SDS</td>
<td>– Users must control partitioning</td>
</tr>
<tr>
<td>– Amazon SimpleDB</td>
<td>– Limited elasticity</td>
</tr>
<tr>
<td><strong>Multi-tenant application databases</strong></td>
<td><strong>Highly optimized for complex workloads</strong></td>
</tr>
<tr>
<td>– Salesforce.com</td>
<td>– Limited flexibility to evolving applications</td>
</tr>
<tr>
<td>– Oracle on Demand</td>
<td>– Inherit limitations of underlying data management system</td>
</tr>
<tr>
<td><strong>Mutable object stores</strong></td>
<td><strong>Object storage versus record management</strong></td>
</tr>
<tr>
<td>– Amazon S3</td>
<td></td>
</tr>
</tbody>
</table>
Application Design Space

Get a few things

Scan everything

Records

Sherpa
MySQL
BigTable
Everest

Files

MObStor
YMDB
Filer
Hadoop
## Comparison Matrix

<table>
<thead>
<tr>
<th></th>
<th>Partitioning</th>
<th>Availability</th>
<th>Replication</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hash/sort</td>
<td>Dynamic</td>
<td>Routing</td>
<td>Failures handled</td>
</tr>
<tr>
<td>PNUTS</td>
<td>H+S</td>
<td>Y</td>
<td>Rtr</td>
<td>Coloserver</td>
</tr>
<tr>
<td>MySQL</td>
<td>H+S</td>
<td>N</td>
<td>Cli</td>
<td>Coloserver</td>
</tr>
<tr>
<td>HDFS</td>
<td>Other</td>
<td>Y</td>
<td>Rtr</td>
<td>Coloserver</td>
</tr>
<tr>
<td>BigTable</td>
<td>S</td>
<td>Y</td>
<td>Rtr</td>
<td>Coloserver</td>
</tr>
<tr>
<td>Dynamo</td>
<td>H</td>
<td>Y</td>
<td>P2P</td>
<td>Coloserver</td>
</tr>
<tr>
<td>Cassandra</td>
<td>H+S</td>
<td>Y</td>
<td>P2P</td>
<td>Coloserver</td>
</tr>
<tr>
<td>Megastore</td>
<td>S</td>
<td>Y</td>
<td>Rtr</td>
<td>Coloserver</td>
</tr>
<tr>
<td>Azure</td>
<td>S</td>
<td>N</td>
<td>Cli</td>
<td>Server</td>
</tr>
</tbody>
</table>
## Comparison Matrix

<table>
<thead>
<tr>
<th></th>
<th>Elastic</th>
<th>Operability</th>
<th>Availability</th>
<th>Global low latency</th>
<th>Structured access</th>
<th>Updates</th>
<th>Consistency model</th>
<th>SQL/ACID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sherpa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y! UDB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BigTable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cassandra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: Green circles indicate availability, red circles indicate unavailability.*
QUESTIONS?