Cloud Data Serving: From Key-Value Stores to DBMSs

Raghu Ramakrishnan
Chief Scientist, Audience and Cloud Computing

Brian Cooper
Adam Silberstein

Yahoo! Research

Joint work with the Sherpa team in Cloud Computing and Utkarsh Srivastava
Cloud Services @Y!: Use Cases

- Ads Optimization
- Content Optimization
- Search Index
- Image/Video Storage & Delivery
- Machine Learning (e.g. Spam filters)
- Attachment Storage

Yahool Mail

Stay connected. Preview and keep up-to-date with your mail.

What's Your Credit Score? A bad credit score is 600 or below. See yours in 2 easy steps for $9. By Experian®
Web Data Management

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

- Large data analysis (Hadoop)
- Structured record storage (PNUTS/Sherpa)
- Blob storage (MObStor)

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

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

*Will discuss today*
ACID or BASE? Litmus tests are colorful, but the picture is cloudy

SCALABLE DATA SERVING
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, resource sharing, operability** …
  – 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
PNUTS/SHERPA
What is PNUTS/Sherpa?

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

Structured, flexible schema

Parallel database

Geographic replication

Hosted, managed infrastructure
PNUTS: Key Components

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

- Graapes are good to eat
- Limes are green
- Apple is wisdom
- Strawberry shortcake
- Arrgh! Don’t get scurvy!
- But at what price?
- How much did you pay for this lemon?
- Is this a vegetable?
- The perfect fruit
- New Zealand
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>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
</tr>
<tr>
<td>Fair</td>
</tr>
</tbody>
</table>
PROCESSING READS & UPDATES
Updates

1. Write key k
2. Write key k
3. Write key k
4. SUCCESS
5. Write key k
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
Range Queries in YDOT

- Clustered, ordered retrieval of records
Bulk Load in YDOT

• YDOT bulk inserts can cause performance hotspots

• Solution: preallocate tablets
ELASTICITY, OPERABILITY, HORIZONTAL SCALING
## 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
Each storage unit has many tablets (horizontal partitions of the table)

Tablet Splitting and Balancing

Storage unit may become a hotspot

Overfull tablets split

Shed load by moving tablets to other servers

Tablets may grow over time
ASYNCHRONOUS REPLICATION AND CONSISTENCY
Asynchronous Replication
• ACID consistency
  – **Transaction:** Add Brian as Toby’s friend, and Toby as Brian’s friend

**States you will never see:**
- Brian is Toby’s friend but Toby is not Brian’s friend
- Toby is Brian’s friend but Brian is not Toby’s friend
Network disruption: Alice redirected to East

<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>
• 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.
PNUTS Consistency Model

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

The image shows a map of the United States with a person using a laptop. The map is used to represent the Record Master data with records for A, B, C, D, E, and F.
Tablet Master

Region W

Key1: 42342 E
Key3: 66354 W
Key4: 12352 E
Key5: 75656 C
Key6: 15677 E

Region C

Tablet master

Key1: 42342 E
Key3: 66354 W
Key4: 12352 E
Key5: 75656 C
Key6: 15677 E

Region E

Key1: 42342 E
Key3: 66354 W
Key4: 12352 E
Key5: 75656 C
Key6: 15677 E
Tablet Mastership

**Step 1: Forward Req to Tablet Master**

Region W
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region C
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region E
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

**Step 2: Apply Insert to Tablet Master**

Region W
- Key1: 42342 E
- Key2: 42521 W
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region C
- Key1: 42342 E
- Key2: 42521 W
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region E
- Key1: 42342 E
- Key2: 42521 W
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

**Step 3: Replicate Insert at Rec Master**

Region W
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region C
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region E
- Key1: 42342 E
- Key2: 42521 W
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

**Step 4: Apply Insert to Other Sites**

Region W
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region C
- Key1: 42342 E
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E

Region E
- Key1: 42342 E
- Key2: 42521 W
- Key3: 66354 W
- Key4: 12352 E
- Key5: 75656 C
- Key6: 15677 E
Write Interaction

• Consistency on (default)
  – Insert: primary keys guaranteed to be unique
    • Across all data replicas
  – Update: updates are applied to a record in a given order
    • Same order for all replicas
  – Delete: deleted records will not re-appear, and records will not disappear without a delete

• Consistency off
  – Above guarantees may not be enforced

• This decision is at the table level
  – Once you do some “consistency off” stuff, cannot guarantee any consistency
Summary: Consistency

- Sherpa offers timeline and eventual consistency
  - Can get per-record ACID using test-and-set
- Other systems offer other choices

<table>
<thead>
<tr>
<th></th>
<th>ACID</th>
<th>Per-record ACID</th>
<th>Eventual</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORACLE</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Weaker serializable levels</td>
</tr>
<tr>
<td>MySQL</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Weaker serializable levels</td>
</tr>
<tr>
<td>cassandra</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>Supports quorum read/writes</td>
</tr>
<tr>
<td>HDFS</td>
<td></td>
<td></td>
<td></td>
<td>Best effort</td>
</tr>
<tr>
<td>VERTICA</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Does not allow updates</td>
</tr>
<tr>
<td>UDB</td>
<td></td>
<td></td>
<td></td>
<td>Best-effort only</td>
</tr>
</tbody>
</table>
AVAILABILITY
Coping With Failures
Possible Failure Modes

**Failure type**

**Storage unit**

**Consistency impact**

None

**Availability impact**

Degraded service (forwards) for some data.

  Updates and inserts fail for some records

**Resolution**

If data not lost: Reboot machine

If data lost: Copy lost tablets from a remote replica

Issue overrides to restore availability

**Time to resolve**

If data lost, hours or less (depending on tablet size and colo location). If no data lost, minutes.
## Possible Failure Modes

<table>
<thead>
<tr>
<th>Failure type</th>
<th>Consistency impact</th>
<th>Availability impact</th>
<th>Resolution</th>
<th>Time to resolve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router</td>
<td>None</td>
<td>None</td>
<td>Boot router</td>
<td>Minutes</td>
</tr>
</tbody>
</table>
Possible Failure Modes

**Failure**

Tablet controller

**Consistency impact**
None

**Availability impact**
Some actions (e.g., tablet copy) will be blocked

**Resolution**
Start secondary controller

**Time to resolve**
Minutes
Possible Failure Modes

**Failure**
One msg hub node

**Consistency impact**
None

**Availability impact**
Writes fail for some records until a new secondary node takes over

**Resolution**
Create new primary or secondary for lost topics

**Time to resolve**
Minutes
Possible Failure Modes

Failure

Colo power outage or partition

Consistency impact

Option to allow “relaxed consistency” to improve availability

Availability impact

Some inserts, updates and deletes cannot succeed

Some critical reads fail

Option to allow updates to proceed in “relaxed consistency mode”

Resolution

Major overrides to force mastership transfer; discard conflicting updates

Time to resolve

Hours
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 PNUThis Shell
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
Green Apples and Red Apples

COMPARING SOME CLOUD SERVING STORES
Help!

- I have a huge amount of data. What should I do with it?
Serving Stores

• 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, MongoDB, 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
Architecture

• 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
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
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

![Architecture Diagram]

- **Client**
- **HBaseMaster**
- **HRegionServer**
- **Disk**
- **REST API**
- **Java Client**
HRegion Server

- Records partitioned into MemStores
  - Each MemStore contains many HFiles
- All writes to MemStore applied to single memcache
- Reads consult HFiles and memcache
- Memcaches flushed as HFiles (HDFS files) when full
- Compactions limit number of HFiles
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
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

Keep in mind that these systems are in active development, and these numbers represent just a current snapshot!
Benchmark Tool

- Java application
  - Many systems have Java APIs
  - Other systems via HTTP/REST, JNI or some other solution

---

Workload parameter file
- R/W mix
- Record size
- Data set
- ...

Command-line parameters
- DB to use
- Target throughput
- Number of threads
- ...

YCSB client
- Workload executor
- DB client
  - Client threads
  - Stats

Extensible: define new workloads

Extensible: plug in new clients
Test setup

• 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
  – Reads retrieve whole record; updates write a single field
  – 100 or more client threads

• Caveats
  – Write performance would be improved for Sherpa, sharded MySQL and Cassandra with a dedicated log disk
  – We tuned each system as well as we knew how, with assistance from the teams of developers
Workload A – Update Heavy

- 50/50 Read/update

Comment: Cassandra is optimized for writes, and has better write latency. However, Sherpa has pretty good write latency, comparable read latency, and comparable peak throughput. HBase has good write latency because it does not sync updates to disk, at the cost of lower durability; but read latency is bad.
Workload B – Read Heavy

- 95/5 Read/update

Comment: Sherpa does very well here, with better read and write latency and peak throughput than Cassandra, and better read latency and peak throughput than HBase. Again HBase write latency is low because of no disk syncs. Buffer pool architecture is good for random reads.
Workload E – Short Scans

- Scans of 1-100 records of size 1KB

Comment: HBase and Sherpa are roughly equivalent for latency and peak throughput, even though HBase is “meant” for scans. Still to do: increase range size and see if HBase becomes better at large ranges. Cassandra range support will be much better in next version (0.5) so we didn’t benchmark it on our 0.4 version.
Scale-up

- Read heavy workload with varying hardware

Comment: Sherpa scales well, with flat latency as system size increases. Cassandra scales less well, with more P2P communication. HBase is unstable; 3 servers or less performs poorly. More experiments are needed to get more data points on these curves.
Ongoing and Future Work

• ACID xacts over “entity groups”, coupled with entity group timelines across replicas
  – Entity groups limited to a single machine, e.g., all records belonging to a user
• Auto tuning
• Multitenancy with SLAs
• Materialized views
• Alternative storage unit architectures
  – LSM files (Stasis)
  – Hybrid OLAP/OLTP workloads?
• Leveraging large (aggregate) main memories
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
QUESTIONS?