YCSB++ Benchmarking Tool

Performance Debugging Advanced Features of Scalable Table Stores

Swapnil Patil

Milo Polte, Wittawat Tantisiriroj, Kai Ren, Lin Xiao, Julio Lopez, Garth Gibson, Adam Fuchs *, Billie Rinaldi *

Carnegie Mellon University, * National Security Agency

Open Cirrus Summit, October 2011, Atlanta GA
Scalable table stores are critical systems

- For data processing & analysis (e.g. Pregel, Hive)
- For systems services (e.g., Google Colossus metadata)
Evolution of scalable table stores

**Growing set of HBase features**

- **2008**: HBase release
- **2009**: RangeRowFilters, Batch updates
- **2010**: Bulk load tools, RegEx filtering, Scan optimizations
- **2011+**: Co-processors, Access Control

Simple, lightweight → complex, feature-rich stores

- 🔴 Supports a broader range of applications and services
- 🔴 Hard to debug and understand performance problems
- 🔴 Complex behavior and interaction of various components
Need richer tools for understanding advanced features in table stores …

**YCSB++ Functionality**

ZooKeeper-based distributed and coordinated testing API and extensions the new Apache Accumulo DB
Fine-grained, correlated monitoring using OTUS

**Features Tested Using YCSB++**

Batch writing • Table pre-splitting • Bulk loading
Weak consistency • Server-side filtering • Fine-grained security

Tool released at [http://www.pdl.cmu.edu/ycsb++]
Outline

• Problem
• YCSB++ design
• Illustrative examples
• Ongoing work and summary
Yahoo Cloud Serving Benchmark [Cooper2010]

- For CRUD (create-read-update-delete) benchmarking
- Single-node system with an extensible API
YCSB++: New extensions

- Added support for the new Apache Accumulo DB
  - New parameters and workload executors
YCSB++: Distributed & parallel tests

Multi-client, multi-phase coordination using ZooKeeper
  — Enables testing at large scales and testing asymmetric features
YCSB++: Collective monitoring

**OTUS monitor built on Ganglia** [Ren2011]

— Collects information from YCSB, table stores, HDFS and OS
Example of YCSB++ debugging

**OTUS** collects fine-grained information

— Both HDFS process and TabletServer process on same node
Outline

• Problem
• YCSB++ design
• Illustrative examples
  — YCSB++ on HBase and Accumulo (Bigtable-like stores)
• Ongoing work and summary
Recap of Bigtable-like table stores

Write-path: in-memory buffering & async FS writes
1) Mutations logged in memory tables (unsorted order)
2) Minor compaction: Memtables -> sorted, indexed files in HDFS
3) Major compaction: LSM-tree based file merging in background

Read-path: lookup both memtables and on-disk files
Apache ACCUMULO

Started at NSA; now an Apache Incubator project
  — Designed for high-speed ingest and scan workloads

New features in ACCUMULO
  — Iterator framework for user-specified programs placed in between different stages of the DB pipeline
    ▪ E.g., Support joins and stream processing using iterators
  — Also supports fine-grained cell-level access control
ILLUSTRATIVE EXAMPLE #1

Analyzing the fast inserts vs. weak consistency tradeoff using YCSB++
Client-side batch writing

Feature: clients batch inserts, delay writes to server

- Improves insert throughput and latency
- Newly inserted data may not be immediately visible to other clients
Batch writing improves throughput

6 clients creating 9 million 1-Kbyte records on 6 servers

- Small batches - high client CPU utilization, limits throughput
- Large batches - saturate servers, limited benefit from batching
**Batch writing causes weak consistency**

**Test setup: ZooKeeper-based client coordination**

- Share producer-consumer queue between readers/writers
- R-W lag = delay before C2 can read C1’s most recent write
Batch writing causes weak consistency

Deferred write wins, but lag can be ~100 seconds

\[(N\%) = \text{fraction of requests that needed multiple read()s}\]

Implementation of batching affects the median latency
ILLUSTRATIVE EXAMPLE #2

Benchmarking high-speed ingest features using YCSB++
Features for high-speed insertions

Most table stores have high-speed ingest features

- Periodically insert large amounts of data or migrate old data in bulk
- Classic relational DB techniques applied to new stores

Two features: **bulk loading and table pre-splitting**

- Less data migration during inserts
- Engages more tablet servers immediately
- Need careful tuning and configuration [Sasha2002]
8-phase test setup: table bulk loading

### Phases

1. **Pre-Load (re-formatting)**
2. **Pre-Load (importing)**
3. **Read/Update workload**
4. **Load (re-formatting)**
5. **Load (importing)**
6. **Read/Update workload**
7. **Sleep**
8. **Read/Update workload**

### Bulk loading involves two steps
- Hadoop-based data formatting
- Importing store files into table store

#### Pre-load phase (1 and 2)
- Bulk load 6M rows in an empty table
- *Goal:* parallelism by engaging all servers

#### Load phase (4 and 5)
- Load 48M new rows
- *Goal:* study rebalancing during ingest

#### R/U measurements (3, 6 and 7)
- Correlate latency with rebalancing work

Swapnil Patil, CMU
Read latency affected by rebalancing work

**Phases**

1. Pre-Load (re-formatting)
2. Pre-Load (importing)
3. Read/Update workload
4. Load (re-formatting)
5. Load (importing)
6. Read/Update workload
7. Sleep
8. Read/Update workload

**Graph:**

- **R/U 1 (Phase 3)**
- **R/U 2 (Phase 6)**
- **R/U 3 (Phase 8)**

**Notes:**

- High latency after high insertion periods that cause servers to rebalance (compactions)
- Latency drops after store is in a steady state
Rebalancing on ACCUMULO servers

**Phases**

1. Pre-Load (re-formatting)
2. Pre-Load (importing)
3. Read/Update workload
4. Load (re-formatting)
5. Load (importing)
6. Read/Update workload
7. Sleep
8. Read/Update workload

**Graph**

- **StoreFiles**
- **Tablets**
- **Compactions**

- **OTUS monitor shows the server-side compactions during post-ingest measurement phases**
HBASE is slower: Different compaction policies

Swapnil Patil, CMU
Extending to table pre-splitting

Pre-split a key range into N partitions to avoid splitting during insertion

Pre-Load (re-formatting)

Pre-Load (importing)

Read/Update workload

Load (re-formatting)

Load (importing)

Read/Update workload

Sleep

Read/Update workload

Load

Pre-split into N ranges

Pre-load

Table pre-splitting test

Read/Update workload

Sleep

Read/Update workload

Bulk loading test
Outline

• Problem
• YCSB++ design
• Illustrative examples
• Ongoing work and summary
Things not covered in this talk

More features: function shipping to servers

— Data filtering at the servers
— Fine-grained, cell-level access control

More details in the ACM SOCC 2011 paper

Ongoing work

— Analyze more table stores: Cassandra, CouchDB, MongoDB
— Continue research through the new Intel Science and Technology Center for Cloud Computing at CMU (with GaTech)
Summary: YCSB++ tool

- Tool for performance debugging and benchmarking advanced features using new extensions to YCSB

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak consistency semantics</td>
<td>Distributed clients using ZooKeeper</td>
</tr>
<tr>
<td>Fast insertions (pre-splits &amp; bulk loads)</td>
<td>Multi-phase testing (with Hadoop)</td>
</tr>
<tr>
<td>Server-side filtering</td>
<td>New workload generators and database client API extensions</td>
</tr>
<tr>
<td>Fine-grained access control</td>
<td></td>
</tr>
</tbody>
</table>

- Two case-studies: Apache HBASE and ACCUMULO
- Tool available at [http://www.pdl.cmu.edu/ycsb++]