Big Data Technology
Real-Time NoSQL Key-Value Stores

Eshcar Hillel
Yahoo!

Ronny Lempel
Outbrain
Roadmap

- Previous class – NoSQL databases motivation

- This class – HBase: NoSQL Key-Value Stores
  - Data model and implementation
  - Log-Structured Merge stores
  - Concurrency control
  - I/O speedup
Apache HBase

http://hortonworks.com/hdp/
80’s-90’s: One-SQL-Fits-All

Online Transaction Processing (OLTP)

- Read-Write Workload
- Real-Time
- Latency-Sensitive
- Transaction-Oriented

Benchmark: TPC-C

SQL DBMS (Oracle, DB2, SQL Server, MySQL, ...)
- ACID transactions
- Moderate scale (TBs not PBs)

Analytics

BI Tools
- Latency-Oriented
- Data Cubes

Read-Only Workload
- Batch Processing
- Non-Transactional
- ETL

Benchmark: TPC-H

Online Transaction Processing

Analytics

BI Tools
- Latency-Oriented
- Data Cubes

Read-Only Workload
- Batch Processing
- Non-Transactional
- ETL

Benchmark: TPC-H

Online Transaction Processing

Analytics

BI Tools
- Latency-Oriented
- Data Cubes

Read-Only Workload
- Batch Processing
- Non-Transactional
- ETL

Benchmark: TPC-H

80’s-90’s: One-SQL-Fits-All

Online Transaction Processing (OLTP)

- Read-Write Workload
- Real-Time
- Latency-Sensitive
- Transaction-Oriented

Benchmark: TPC-C

SQL DBMS (Oracle, DB2, SQL Server, MySQL, ...)
- ACID transactions
- Moderate scale (TBs not PBs)

Analytics

BI Tools
- Latency-Oriented
- Data Cubes

Read-Only Workload
- Batch Processing
- Non-Transactional
- ETL

Benchmark: TPC-H

CS236620 Big Data Technology
Motivation – Real-Time Applications

- Communication
  - Instant messaging/ Microblogging/ Location check-in
- E-commerce
  - Product carts
- Content recommendations, online advertising
  - User profile aggregation and serving
- Incremental data analysis
  - Page ranking
Key-Value (KV-) Stores

- **API semantics**
  - Random access: lookup (**get**), insert/update/delete (**put**)
  - Sequential retrieval (**scan, range queries**)

- **Items:** unique key, sparse bag of attributes
  - No relational schema
  - Multiple applications share one table

- Very high volumes of small operations
  - Low-selectivity queries, lightweight updates

- Increasingly write-intensive workloads
- Latency is king
Data Model

- Versions
  - $v_0$
  - $v_1$
  - $v_2$
  - $v_3$

- Column family
  - Column key
  - Column key

- Cell

- Row key
Data Model Explained

- **Cell = unit of addressing**
  - Row key + column key + version

- **Why multi-versioning?**
  - Faster writes and reads
  - Efficient storage management

- **Why column families?**
  - Typically, column family = application
Model Implementation

- Tables on top of DFS
  - Scalability and fault tolerance come for free
  - Simple data sharing with MR jobs
- Table → multiple regions (tablets)
  - Unit of horizontal scalability
  - Range-partitioned – good for lookups, range queries
  - Easy to split (if overfills) and move
- Region → multiple stores, one per CF
Model Implementation

Region

A-H

File

I-P

Q-Z

CF1 (store)

CF2

CS236620 Big Data Technology
Region Management

- Region server (RS) controls multiple regions
  - RS is the I/O gatekeeper to the region
  - Simple concurrency control
  - Might become a bottleneck
- Not necessarily co-located with the data
  - Locality can help performance
- Control can move around
  - Recovery, load-balancing
Store: Log-Structured-Merge (LSM) Data

- Random I/O → Sequential I/O

- Random writes absorbed in RAM ($C_m$)
- Disk components ($C_d$) immutable
  - Write-once-read-many
- Periodic batch merge to disk
  - High-throughput I/O
- Random reads from $C_m$ or $C_d$ cache
Write Path

- Random writes to RAM ($C_m$)
  - Low-latency
  - Write-ahead-log for persistence

- When memory segment is full
  - Becomes immutable (snapshot)
  - A new mutable (active) segment serves writes
  - Flushed to disk, truncate WAL
Write Path - Compaction

- Compaction
  - Merge-sort files
  - Eliminates obsolete versions
- Minor compaction – partial
- Major compaction - full
Read Path

All components are write-once read-many.
Compaction helps maintain fewer (and smaller) files.
Read Path – Reading File

- Data file
- Index
- Data
- Block cache
- Immutable
- Accessible to MR
HBase Architecture

Concurrency Control – Write Synchronization

2 concurrent writes to \{company, role\} columns
key: Greg -> Cloudera, Engineer
key: Greg -> Restaurant, Waiter

Inconsistent state: key: Greg -> Restaurant, Engineer
Solution: row lock
Concurrent Control – RW Synchronization

Inconsistent state: key:Greg -> Restaurant, Engineer
Solution: multi-versioning (snapshot isolation semantics)
Multiversion Concurrency Control (MVCC)

Read is assigned a read point r_ts
- Highest integer s.t. all writes with write number $\leq x$ have completed

Returns the data cell with largest write number $\leq r_ts$
I/O Speedup

- **Block cache**
  - Reads served from RAM

- **Vertical partitioning** by column families
  - Per-application locality

- **Lookup speedup** with Bloom filters
  - Compact, probabilistic data structure
  - Tests whether an element is part of set
    - No false negatives, very few false positives
Bloom Filters Primer

- A bit array $B[m]$
- $H$ hash functions $h_1, \ldots, h_H$

- **Insert** ($k$)
  - For each $1 \leq i \leq H$, $B[h_i(k)] \leftarrow 1$

- **Lookup** ($k$)
  - Return $B[h_1(k)] \& \ldots \& B[h_H(k)]$

- False positive w/ low probability
- No false negatives
Summary

- Real-time, Latency-oriented databases
- Linear and modular scalability
  - Vertical partitioning, automatic vertical table sharding
- Strictly consistent reads and writes
  - Snapshot isolation per row
- Automatic failover support between RegionServers
- Block cache and Bloom Filters for real-time queries
Next Class

- Canonical stream mining problem
Further Reading

- Bigtable: A Distributed Storage System for Structured Data
- HBase