Sweet Storage SLOs with Frosting

Andrew Wang, Shivaram Venkataraman, Sara Alspaugh, Ion Stoica, Randy Katz
Exploratory drill-down

Batch analytics

Interactive web-serving

HBase

HDFS

MySQL
Exploratory drill-down

Batch analytics

Interactive web-serving

HBase

HDFS

MySQL

Copy
Exploratory drill-down

Batch analytics

Interactive web-serving

HBase

HDFS

MySQL
Exploratory drill-down

Batch analytics

Interactive web-serving

HBase

HDFS

MySQL
Today’s Clusters

• Segregated storage systems

• Downsides
  – Delayed reaction time from analytics
  – Increased provisioning costs ($$$)
  – Reduced cluster utilization
  – Duplication of data

• Ideally, all apps share a single storage system!
Exploratory drill-down

Batch analytics

Interactive web-serving

Unified Storage Layer

Storage layer is not aware of each application’s SLO!
Mixing Front-end and Batch

![Graph showing 99th percentile latency over time for Baseline (FIFO, frontend only) and FIFO types.](image)
High-percentile Latency SLOs

• Metric of merit for latency
  – 95th or 99th percentile

• Important with request fan-out
  – Stragglers affect overall latency

• Growing importance for complex pages
  – Status updates, profile pics, friend requests, etc.
Problem

• Want to multiplex front-end and batch workloads

• Also need 99th percentile latency guarantees for front-end storage system operations
Existing solutions

• Mismatch between apps and storage systems
  – Apps think about key-value or row operations
  – Storage systems think about disks

• Need to manually tune low-level parameters
  – MB/s, IOPS, etc.

• Use average latency, not 99th percentile
Goals of Frosting

• Enable a single, shared storage layer
• High-level *service-level objectives* (SLOs) specified directly to the storage system
  – “my gets will finish in 200 ms, 99% of the time”
• No manual tuning by the app programmer
Deep Software Stacks

- **HBase**
  - BigTable-like
  - Distributed column store
  - Get, put, scans on rows

- **HDFS**
  - GFS-like
  - Distributed filesystem

- **OS**
  - Interfaces with hardware
Deep Software Stacks

- Clean layered architecture
- Request processing traverses software stack
  - Hard to debug latency!
  - Lots of code
  - Complex interactions
Frosting Architecture

- Try the simple approach

Client

HBase

HDFS

OS
Frosting Architecture

- Try the simple approach
- Insert scheduling at the top layer (HBase)
- Proportional share among HBase clients
- Dynamically adjust shares to enforce client SLOs
High-level SLO Enforcement

- Feedback loop
- Measure each client’s performance
- Compare with SLO
- Increase or decrease allocation accordingly
Evaluation

• HBase cluster on c1.xlarge EC2 nodes
  – 8 CPU cores
  – 4 local disks

• Yahoo! Cloud Serving Benchmark clients
  – **Frontend**: 1-row gets, high priority
  – **Batch**: 500-row scans, low priority
Evaluation

- FIFO
- Frosting, 99:1
- Frosting, 90:10
- Baseline (frontend only)

99th Percentile Latency (ms)

Time Elapsed (s)
Evaluation

Average Throughput (req/s)

Schedulers

FIFO
Frosting 90:10
Frosting 99:1
Evaluation

- 99th percentile latency
- Frontend share
Conclusion

• Fundamental latency vs. throughput tradeoff
• High-level SLOs can be enforced directly and automatically by the storage system
• Ideas can be applied to existing systems