Chain Replication in Theory and in Practice

Scott Lystig Fritchie
slfritchie@snookles.com

Gemini Mobile Technologies

September 30, 2010

Overview

- Introduction to Chain Replication
- Overview of Hibari's Implementation
- 30 seconds on the OSI Systems Fault Model
- OSI FCAPS: Fault management
- OSI FCAPS: Performance management
- Erlang-Specific Issues

Chain Replication

Papers

- "Chain Replication for Supporting High Throughput and Availability" by Robbert van Renesse and Fred B. Schneider, USENIX OSDI 2004.
- "Object Storage on CRAQ: High-throughput chain replication for read-mostly workloads" by Jeff Terrace and Michael J. Freedman, USENIX Tech, 2009

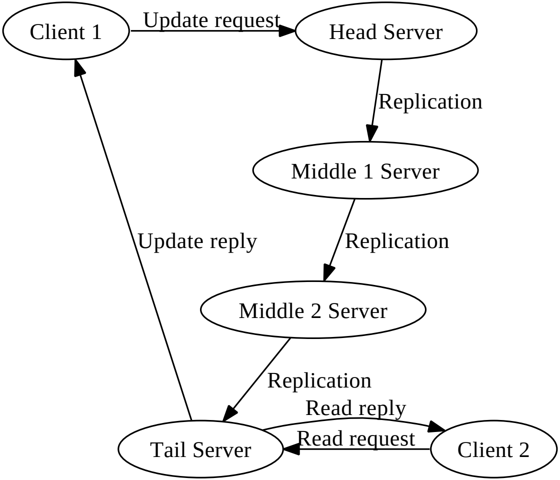
Chain Replication

Overview

- A variation of master/slave replication
- State machine replication
- In contrast, quorum replication is more popular in open source
    - Dynomite, Riak, Cassandra

Chain Replication Messaging Flows

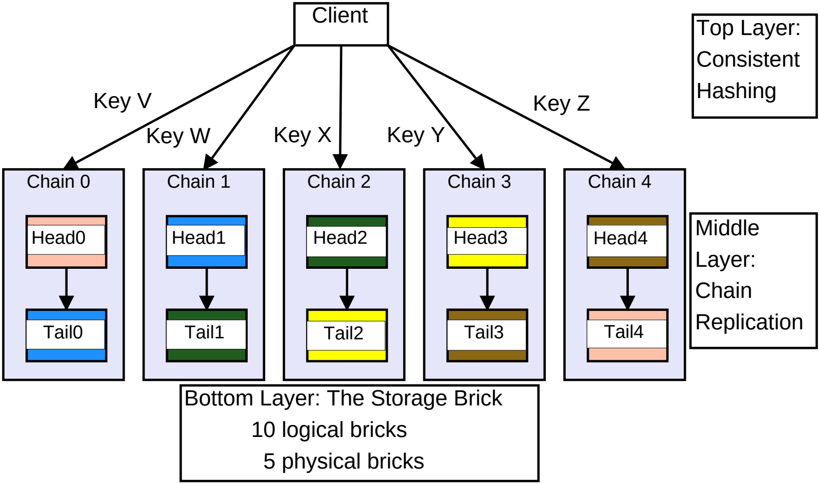
Strong Consistency: Read the Last Write



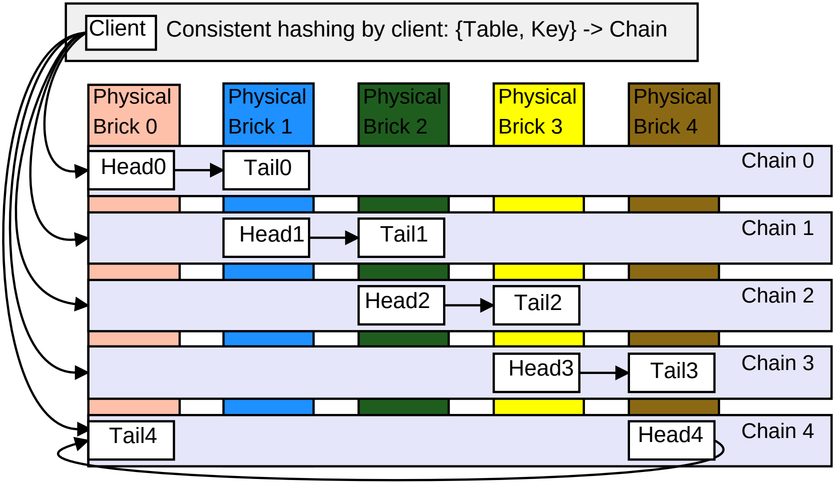
Hibari Overview

- Top layer: consistent hashing
  - Collections of billions of keys
- Middle layer: chain replication
  - Replicate a single key
  - Single collection of 0 – 50 million keys
- Bottom layer: storage brick
  - Store a single key
  - Single collection of 0 – 50 million keys

Hibari Logical Architecture: View 1



Hibari Logical Architecture: View 2



OSI/ISO Systems Fault Model: FCAPS

- Fault
  - Recognize, isolate, correct, and record faults
- Configuration
  - Gather, store, and modify device configuration state
- Accounting
  - Gather and store resource usage and billing data
- Performance
  - Gather, store, and analyze performance data
- Security
  - Gather, store, and modify device access control

Performance Management

Research Papers vs. Planned Use

- van Renesse and Schneider (2004):
  - Performance experiments are simulated
- Terrace and Freedman (2009):
  - Performance measured on Emulab *pc3000*-type machines
  - Focused on read-mostly workload:
    - Read:write ratio of 50:1 upwards to 150:1 (?)
- Gemini's planned use
  - Low-to-mid-range x86_64 servers + RAID disk
  - Write-heavy workload: read:write ratio of 3:1 or worse

Performance Management

I/O Latency

- Hard disks are no fun....
- Write latency
  - Append-only logs to minimize disk head seeks
    - Log files are the <u>only</u> data structure
  - `fsync(2)` latency + hard disks = slow
    - Hibari default: all updates are durable
  - All logical brick logs written to a common log
  - The common log aggregates `fsync(2)` requests
- Most Hibari bugs were in write/sync management code.

Performance Management

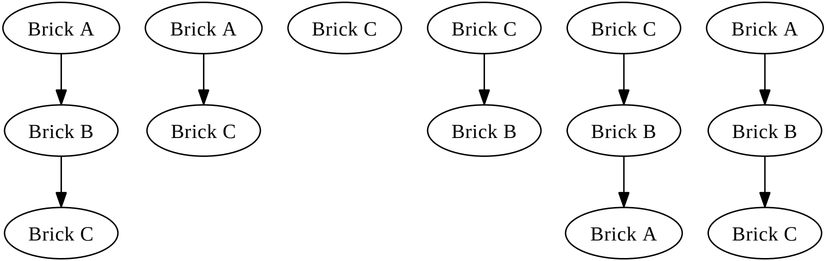
I/O latency

- Read latency
  - Read I/O generators: normal workload, brick repair, key migration, data "scavenger"
  - Avoid blocking `gen_server` processes with I/O
    - "primer": spawn proc to pre-read value data
    - Borrowed from Squid HTTP cache, Flash HTTP server
  - Access by lexicographic vs. temporal orders
    - Repair and migration: lexicographic order (by key)
    - Optimal read I/O pattern: temporal order (by update time)
    - Full repair of 2-3 TByte brick can take <u>several days</u>
  - Rate control

Performance Management

I/O latency

- Other workload factors:
  - Chain reordering
  - OS-based read-ahead



Configuration Management

The "master"

- van Renesse and Schneider (2004) call for a "master":
- Detects failures
- Reorders chains
- Informs clients about new chain state
- "In what follows, we assume the master is a single process that never fails."
- In prototype, multiple master processes + Paxos replication

Configuration Management

Hibari Admin Server

- Admin Server is a single running entity
  - Active/standby OTP application
  - Static configuration: only 2 or 3 machines in cluster
- Monitors health of each logical brick
- Reconfigures chains when brick health changes
- Stores brick health history in Hibari logical bricks
  - Uses quorum replication
  - Avoid "chicken and the egg" bootstrap problem
- But ... OTP app controller is vulnerable to network partition
  - Segue to the next FCAPS topic...

Fault Management

- Detect Admin Server failures
- Detect brick failures
- Detect network partitions
- Detect brick failures during network partitions?
- Repair out-of-sync replicas

Fault Management

- van Renesse and Schneider (2004): "Servers are assumed to be fail-stop"
  - Terrace and Freedman (2009): same
- Fail stop means . . . stop?
  - Except when network partitions heal
    - Kill any brick that makes illegal health state transition
  - Except in arbitrary message delays
    - `net_kernel` deadlock
    - `busy_dist_port` throttling

Fault Management

Replica repair/re-sync

- van Renesse and Schneider: add repairing/re-syncing server at end of chain
    - Also: Replay all updates in same order
- But ... non-trivial task with concurrent updates!
    - Update key K vs. repair K
    - Manage `write(2)` and `fsync(2)` delays on both bricks
        - 99^{th} percentile latency of 350+ milliseconds not uncommon

Fault Management

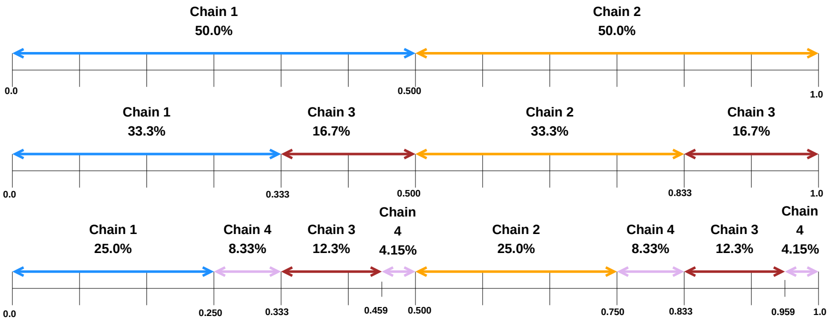
Other items

- Partition detector app
  - Use two physical networks to detect partition of one network
- Key timestamps . . . strictly increasing with each update
  - Clients: enable compare-and-swap atomic operations
  - Servers: quick way to determine key sync status
- File checksums
  - Crash brick when corruption is detected
- Replica placement
  - Very flexible, but no automatic “rack awareness”

Fault/Configuration Management

Automatic Key Migration

- Change chain length, i.e. change replication factor
- Add/remove/reweight chains
- Automatically rebalance keys across chains



Erlang-Specific Issues

- Messaging reliability
  - "Send and pray" — Joe Armstrong
  - . . . too easy to forget when buried in code
- Murphy's law
  - A process rate-limited at 40 msgs/sec sends 85,000 messages in 60 seconds?
  - Impossible . . . but it really happened
- Be extra-conscious of code with side-effects
- How many nodes can Erlang network distribution support?

Thank You!

- ACM anonymous reviewers, Louise Lystig Fritchie, Olaf Hall-Holt, Satoshi Kinoshita, James Larson, Romain Lenglet, Jay Nelson, Joseph Wayne Norton, Gary Ogasawara, Mark Raugas, Justin Sheehy, Ville Tuulos, and Jordan Wilberding

- `http://hibari.sourceforge.net/`
- `http://www.snookles.com/slf-blog/tag/hibari/`