Chain Replication for Supporting High Throughput and Availability

Robbert van Renesse  
rvr@cs.cornell.edu

Fred B. Schneider  
fbs@cs.cornell.edu
Abstract

“Chain replication is a new approach to coordinating clusters of fail-stop storage servers. The approach is intended for supporting large-scale storage services that exhibit high throughput and availability without sacrificing strong consistency guarantees.”
Abstract

● A Storage Service Interface
● Chain Replication
● Comparison to Primary/Backup
● Simulation Experiments
● Concluding Remarks
A Storage Service Interface
A Storage Service Interface

- persistent map from objId to value
- query(objId) -> value
  - retrieve current value of objId
- update(objId, newVal) -> value
  - update value of objId
  - value := f(oldVal, newVal)
  - not necessarily just PUT (ie CAS, ...)
  - but no cross-object transactions
A Storage Service Interface

State:

- Hist[$objId$]
  - History of all updates to $objId$
  - \( \text{query}(objId) = f(Hist[objId]) \)
- Pending[$objId$]
  - Set of pending requests for $objId$
A Storage Service Interface

Transitions:

● **T1**: Client request ‘r’ arrives  
  ○ Pending[objId] += r

● **T2**: Client request ‘r’ ignored  
  ○ Pending[objId] -= r

● **T3**: Client request processed  
  ○ Pending[objId] -= r  
  ○ if (update) Hist[objId] += r
A Storage Service Interface

Desirable Properties:

● High Availability
● High Throughput
● Strong Consistency
  ○ Operations are linearizable
  ○ Read-your-own-writes
Chain Replication
Chain Replication

Assumptions:

● Servers are fail-stop
  ○ More or less reasonable
  ○ (minus bugs, attackers, magnets, etc)

● Failures can be detected
  ○ Trickier than it sounds
Chain Replication

Figure 2: A chain.
Chain Replication

Node State:
- Hist[i] = list of updates processed by node ‘i’
- Sent[i] = updates seen by ‘i’ but not ACKed

System state:
- Pending = requests seen by any node but not yet processed by TAIL
- Hist = Hist[TAIL]
Chain Replication

Invariants:

- Hist[i] >= Hist[i+1]
  - (Update Invariant)
- Hist[i] = Hist[i+1] + Sent[i]
  - (In-process Requests Invariant)
Chain Replication

The Happy Case

- HEAD/TAIL receive a request
  - Added to Pending (T1)
- Query processed by TAIL
  - Removed from Pending (T3a)
- Update processed by TAIL
  - Removed from Pending, added to Hist (T3b)
Chain Replication

Dealing with Failure

- “Single” master
  - AKA Zookeeper
- Detects failed nodes
- Reconfigures the chain
- Points clients to HEAD and TAIL
Chain Replication - Failure of HEAD

H → H+ → [1,2]

updates

[1,2,3,4,5] → [1,2,3]
Chain Replication - Failure of HEAD

Master detects that H is dead, removes it from the chain
Chain Replication - Failure of HEAD

4 and 5 are lost - this is just (T2)
Chain Replication - Failure of TAIL

[1,2,3,4,5] [3,4,5]

queries

[1,2,3] [3]

replies

[1,2]
Master detects that T is dead, removes it from the chain
Chain Replication - Failure of TAIL

3 has now been processed by a tail - this is just (T3b)
Chain Replication - Failure of TAIL

Queries now go to T-
Chain Replication - Failure of Interior
Chain Replication - Failure of Interior

Master detects that S is dead
Chain Replication - Failure of Interior

Master asks S+ for its largest seqId, tells it to S-
Chain Replication - Failure of Interior

S- forwards missing updates to S+
Chain Replication - Failure of Interior

Then continues forwarding additional updates in order
Chain Replication - Extending

queries

[1,2,3,4,5]
[4,5]

T

replies

[1,2,3]
Chain Replication - Extending

New node added at T+, state initially empty
Chain Replication - Extending

T forwards Hist to T+, starts tracking Sent[T]
Chain Replication - Extending

Once done, In-process Updates Invariant holds
Chain Replication - Extending

T stops acting as tail, forwards $\text{Sent}[T]$ to $T^+$
Chain Replication - Extending

Once T+ has all of Hist, it’s the new TAIL
Comparison to Primary/Backup
Comparison to Primary/Backup

Primary/Backup:
Comparison to Primary/Backup

- CR splits P’s work between HEAD and TAIL
  - HEAD sequences/applies updates
  - TAIL interleaves queries
  - -> better overall throughput

- CR distributes updates serially
  - -> better throughput
  - -> higher latency
Comparison to Primary/Backup

Failure Recovery: CR

- **Head failure**
  - Updates unavailable for 2x message time
- **Middle failure**
  - Updates delayed for 4x message time
- **Tail failure**
  - Query unavailable for 2x message time
  - (updates delayed in the meantime)
Comparison to Primary/Backup

Failure Recovery: P/B

● Primary failure
  ○ Everything down for 5x message time

● Backup failure
  ○ Updates down for 1x message time
  ○ Queries for ‘dirty’ rows down 1x message time
  ○ Queries for ‘clean’ rows unaffected
Simulation Experiments
Simulation Experiments

- Infinite bandwidth
- Message latency = 1ms
- Query latency = 5ms
- Update latency = 50ms
- Applying a pre-calculated update = 20ms
- 25 clients, 1 concurrent request per client
Simulation Experiments

Single chain, no failures: better throughput than P/B
Simulation Experiments

Multiple chains:
- Each chain manages a subset of objects
- Consistent hashing from objId to chain
- Servers may participate in multiple chains
- 5000 chains, each with 3 servers
- Same 25-client load, randomly distributed
Simulation Experiments

Multiple chains: horizontal scalability
Simulation Experiments

Effects of failures:
- 24 servers
- 5000 chains (length 3)
- 150 GB/server
- 6.25 MB/s max bandwidth for recovery
- 10 minutes to reboot failed server
Simulation Experiments

Throughput with failures (at 00:30)
Concluding Remarks
Concluding Remarks

- CR supports strong consistency
- CR has better throughput than P/B
  - Sharing load between head and tail
  - More even bandwidth distribution
- CR has better availability than P/B
  - Mainly via faster recovery
  - Not partition tolerant though
- (worse latencies though)
Concluding Remarks

Subsequent work:

- **Object Storage on CRAQ**
  - Read from middle nodes to scale out
  - ‘Dirty’ reads require a check with the tail
- **ChainReaction**
  - Add additional nodes after the tail to scale out reads
  - Reads from post-tail are eventually consistent
Questions?