Dynamo

Dist Sys

Theory
Academic

Practice

'00 → '10

Web services

early
key-value
storage

Goals
System Arch

(most)

Goals:

→ Scale

→ Efficiency/Latency

(=)

→ Availability

(=)

Simple API: get/put
keys/value

(=) rise of key/value stores

≈ "NoSQL movement" → "SQL"

two
religions
Background: "eventual consistency"

Availability: "always writeable"
- allow writes
- conflict resolution on reads

how to resolve conflicts?
- system
  - e.g. last writer wins (overly general)
  - app can be smarter

System Architecture:

API: get/put
- shopping cart
- is this interesting? (adv.)

Partitioning
- consistent hashing (but not full thing a la Chord paper)
  - key
(use virtual nodes)

Replication: \( N \) hosts

1 of \( N \): coordinator

\( \text{put} \rightarrow \) make sure key, properly replicated

"preference list": nodes resp.

\( N \)-way replication,

\( \text{pref list} > N \): why?

\( \rightarrow N \) distinct phys nodes

\( \rightarrow \) may be some down

"Eventual" Consistency: replicas not always in sync.

\( \text{put new} \)

\( c_i: \text{get} \rightarrow \text{new} \)

\( c_i: \text{get} \rightarrow \text{old} \)

Support for "shopping cart"

\( \rightarrow \) never lose "add to cart"

adds/deletes \( \rightarrow \) puts (writes)

how to track versions/conflicts:

version vectors
easy conflict detection can be a standard problem: large limit size to 10

Executing get/put:

1) Load balancing
   - generic load balancer (might need extra hops)
   - partition-aware client library

2) Coordinator:
   - first N healthy nodes
     (up failures → go further down path)

Quorum-based: R, W
   if R + W > N, intersection
   but: availability
     instead "sloppy" quorum:


= first N healthy nodes
  (skip nodes down) keep ping A, until up
  \[ \omega = ? \] "handoff"

Replica Synchronization

= Anti-entropy \{ Random pairwise \} sync

= Detect/fix differences:
  \{ Merkle trees \}
  problem: classic "consistent hashing"

= key ranges keep changing \& node addition/deletion

[Break]
Ending the class:

- W
- Th
- F
- BigTable
- Dynam
- (with) Blockchain
- 12
- 17
- 19-21
- Midterm #2
- take home
- final pres (+ it didn’t start very short write-up)
- celebration

Dynamo: last tech piece

- membership:
  - join/leave
- worried partitioned: solve by using “seed” nodes
- failure detection:
  - mostly to “local” failure detection
  - e.g. coordinator -> replicate write $x_1, \ldots, x_n$
historical: p2p  vs  not p2p: smaller scale

Experience/Lessons:

Performance vs. Durability:
(trade-off)  (latency)
e.g. in Unix file system
 disk/OS cache: write() \rightarrow mem

Dynamo:
write buffer: batch locally
\rightarrow can lose data
\rightarrow reduced tail latency by 5x

writes to N nodes:
\rightarrow memory
\rightarrow disk

Dynamo: hybrid
\rightarrow N node rep.
\rightarrow 1 durable mem.
\rightarrow rest (w)
memory: eventually persisted memory: perf but not as "reliable"

→ but: make more reliable by placement (across racks)
   (but bugs...)

Load Distribution

→ Partitioning:
   Consistent Hashing
   → really simplified
      (as compared to Chord)

Why?
1) → [Chord: variable size ranges]
   → hard to sync.

Background Tasks

→ scheduling of bg tasks
   (details: not shown here)
instead: fixed-size partitioning

2) knowledge of key-range responsibility

→ Chord: distributed
→ Dynamo: all nodes have this mapping info

Conclude:
→ key/value (NoSQL)