Dynamo: Amazon’s Highly Available Key-Value Store

DeCandia et al.
Amazon.com

Presented by
Sushil
CS 5204
Motivation

• A storage system that attains high availability, performance and durability

• Decentralized techniques
  – Data partitioning
  – Replica synchronization
  – Membership
Agenda

• Introduction
• System Architecture
• Implementation
• Experiments
• Conclusion
Amazon.com

• One of the largest e-business platform
  – 3 million checkouts on a peak day
• Also a major cloud hosting service
  – Customer trust
• Hundreds of thousands of machines
• Network failures/ disk failures is a norm

Availablility?
Service Oriented Architecture

- Loosely coupled replicated services
- Stateless
- Persistent store

- Services e.g
  - Recommendation, top selling, catalog, etc
Service Requirements

• Query Model – key, value
  – Code versioning systems
• Must be able to make tradeoffs between availability, consistency, durability
• 99.99 percentile SLA
• Example – Shopping Cart service

• why not relational database?
Design Considerations

• Highly available (writes)
  – Eventually consistent
  – Merge during read
  –Handled by applications

• Less manual interaction

• Incrementally scalable

• Completely decentralized
  – Contrast Bigtable?
Challenges

- Partitioning
- Availability (writes)
- Handling Failures
  - Temporary
  - Permanent
- Membership
Partitioning

- Consistent Hashing
  - Contrast linear hashing?
  - MD5 hash
- Replication – N nodes
  - Preference list
    - Multiple data centers
  - Coordinator
- Is this a global view?
  - Hierarchical namespace?
Ring partitioning

• Problems
  – Non uniform data
  – Heterogeneity

• Use virtual nodes
  – Multiple tokens per node
  – Add/remove node keeps system load steady
  – Incorrect buckets are bounded.
Data Versioning

• Asynchronous updates
• Merging – maintain new immutable version
  – Objects as blobs
  – Syntactic and semantic
  – Associative and commutative?
• Multiple branches
• Reconcile versions – during read
  – Last write wins
• Vector clocks
Vector clocks

- list (node, counter)
- Partial ordering
- Merged during read
Get and Put

- **API**
  - `get(key)` returns `list<object>`
  - `put(key, context, object)`
- **Context** – contains metadata & version
- **Load balancer or client library**
- **Request forwarding to coordinator**
- **Quorum** – durability and anti-entropy
  - `R` nodes for read
  - `W` nodes for write
Hinted Handoff

• Sloppy quorum
  – Use first N healthy nodes
  – N=3, B unresponsive
  – Sent to E, metadata hints B
Replica Synchronization

- Anti-entropy
- Merkle trees
  - Leaf – hash value of key
  - Parents - hash of childs
  - One tree per key range/virtual node
  - Peers compare merkle trees
- Advantages
  - Less reads
Gossip

• Admin issue command to join/remove node
• Serving node records in its local membership history
• Gossip based protocol used to agree on the memberships
• Partition and Placement information sent during gossip
Failure detection

Integer pr; /* Local period number */

Every \( T' \) time units at \( M_i \):

0. \( pr := pr + 1 \)
1. Select random member \( M_j \) from view
   Send a ping\((M_i, M_j, pr)\) message to \( M_j \)
   Wait for the worst-case message round-trip time for
   an ack\((M_i, M_j, pr)\) message
2. If have **not** received an ack\((M_i, M_j, pr)\) message yet
   Select \( k \) members randomly from view
   Send each of them a ping-req\((M_i, M_j, pr)\) message
   Wait for an ack\((M_i, M_j, pr)\) message until
   the end of period \( pr \)
3. If have **not** received an ack\((M_i, M_j, pr)\) message yet
   Declare \( M_j \) as failed

Anytime at \( M_i \):

4. On receipt of a ping-req\((M_m, M_j, pr)\) (\( M_j \neq M_i \))
   Send a ping\((M_i, M_j, M_m, pr)\) message to \( M_j \)
   On receipt of an ack\((M_i, M_j, M_m, pr)\) message from \( M_j \)
   Send an ack\((M_m, M_j, pr)\) message to \( M_m \)

Anytime at \( M_i \):

5. On receipt of a ping\((M_m, M_i, M_l, pr)\) message from
   member \( M_m \)
   Reply with an ack\((M_m, M_i, M_l, pr)\) message to \( M_m \)

Anytime at \( M_i \):

6. On receipt of a ping\((M_m, M_i, pr)\) message from member \( M_m \)
   Reply with an ack\((M_m, M_i, pr)\) message to \( M_m \)
Implementation

• Local persistence
  – BerkleyDataBase Transactional Data Store

• Request Coordination
  – SEDA architecture
Read Operation

• Send read requests to nodes
• Wait for minimum no of responses (R)
• Too few replies fail within time bound
• Gather and find conflicting versions
• Create context (opaque to caller)
• Read repair
Values of N, R and W

• N represents durability
  – Typical value 3
• W and R affect durability, availability, consistency
  – What if W is low?
• Durability and Availability go hand-in-hand?
Results

(hourly plot of latencies during our peak season in Dec. 2006)
Out of balance nodes

Figure 6: Fraction of nodes that are out-of-balance (i.e., nodes whose request load is above a certain threshold from the average system load) and their corresponding request load. The interval between ticks in x-axis corresponds to a time period of 30 minutes.
Partition Strategies

• T random tokens per node and partition by token value
  – Scan a range
  – Updating merkle trees

• T random tokens per node and equal partitions
  – Decoupling partition and placement
  – Changing the placement scheme at runtime

• Q/S tokens per node, equal partitions
Conclusion

• Dynamo has provided high availability and fault tolerance
• Provides owners to customize according to their SLA requirements
• Decentralized techniques can provide highly available system
Current State

- Some of the principles used by S3
- Open source implementation
  - Cassandra
  - Voldemort