Dynamo: Amazon's Highly Available Key-Value Store

Guiseppe DeCandia (Amazon.com), Deniz Hastorun (Amazon.com), Madan Jampani (Amazon.com), Gunavardhan Kakulapati (Amazon.com), Avinash Lakshman (Amazon.com), Alex Pilchin (Amazon.com), Swami Sivasubramanian (Amazon.com), Peter Vosshall (Amazon.com), and Werner Vogels (Amazon.com)

- Sai Ram Kuchibhatla

Outline

- Introduction
- Assumptions
- Design Considerations
- Architecture
- Results
- Lessons

Dynamo

- Awarded an Audience Choice award - SOSP 2007 Program
- “Always-On” data store
- Why not relational database?
  - strong consistency => limits scale and availability
  - Cost associated with complex querying capability
- Strict performance, reliability, efficiency requirements
- Requirements must be met even in extreme situations (network, data center, component losses)
- Consistency vs. Availability
- Concepts used: DHTs, consistent hashing, vector clocks, quorum, anti-entropy based recovery, gossip etc

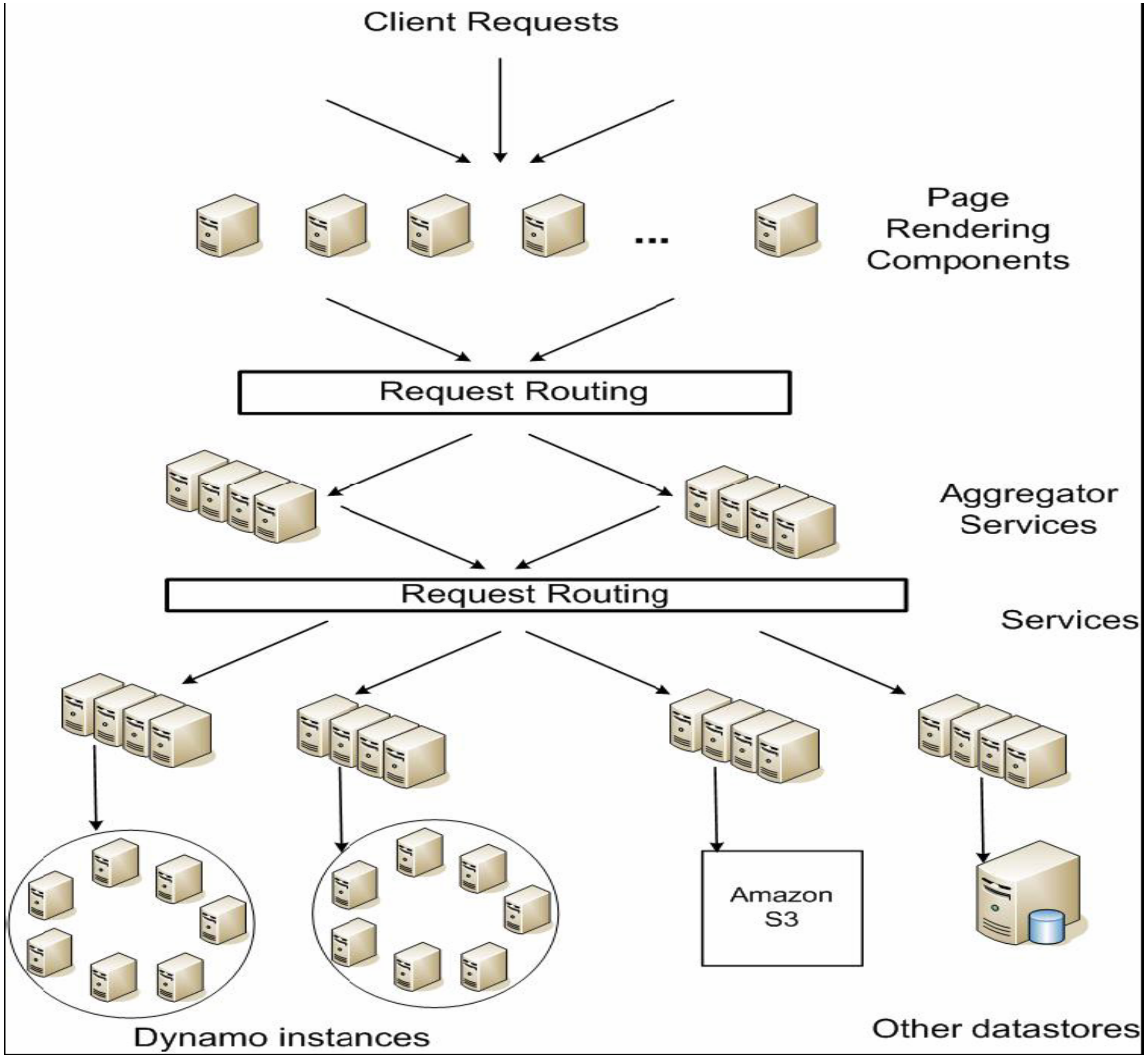
Characteristics

- Key-value store: no complex queries; single-object put/get
- Responsive: low 99.9th percentile latency
- Incrementally scalable: DHT-based
- App-tunable: tunable instance per app
- Always-writable: e.g. shopping cart
- Internal only: ignores authentication, security, integrity

Assumptions

- Query Model:
  - Simple reads and writes
  - Size of objects usually less than 1MB
- ACID properties:
  - Weaker Consistency
  - Permits only single key updates
- Efficiency:
  - Strict SLAs measured at 99.9th percentile
- Security and Trust
  - No authentication, authorization, integrity, DoS limits

Service Oriented Architecture



Design Considerations

- Optimistic Replication
  - Eventual Consistency
- Two problems:
  - When to resolve conflicts?
    - During read operations
  - Who should resolve?
    - Application.

Design Considerations
– Key Principles

- Incremental Scalability
  - Minimal impact on SLA
- Symmetry
  - All nodes have the same set of responsibilites
- Decentralization
  - Scalable, avoids failures as in Centralized design
- Heterogeneity
  - Work distribution proportional to the capacity of the node.

Related Work

- Ocean Store
  - Built on top of structured overlays
  - provides serializable updates
  - Well suited on untrusted platforms
- FAB
  - Distributed block storage system
  - high availability: Objects broken down into smaller blocks
- Bigtable
  - For structured data
  - Supports multiple attribute access

  - Dynamo differs: always writable, trusted environment, latency sensitive, reduced hops

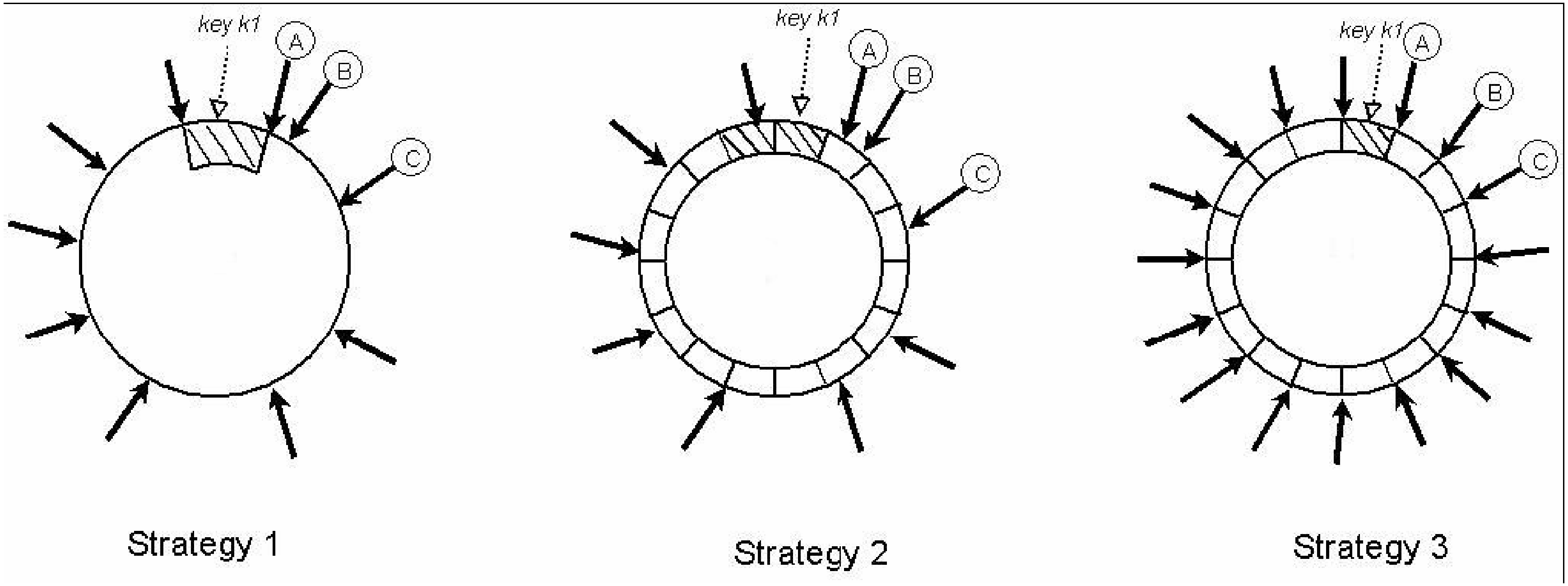
Architecture

- Node Partitioning
- Data Replication
- Object Versioning
- Failure Handling
- Node Joining

Partitioning Algorithm

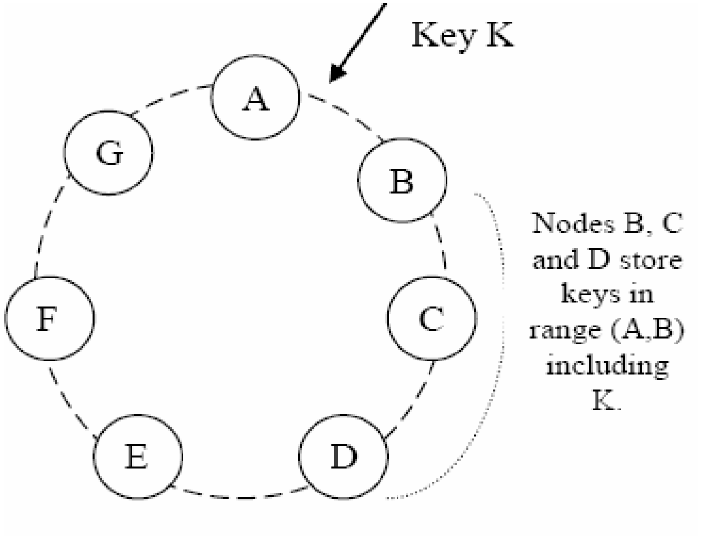
- Should scale incrementally
- Based on Consistent Hashing
- Advantages ?
  - Node joins/ leaves only affect immediate neighbors
  - Fast retrieval of objects
- Disadvantages?
  - Non- uniform load distribution
  - Unaware of the underlying nodes capacity
- Solution - Variant of Consistent Hashing
  - Virtual Node: Each node takes several positions on the ring

Load Balancing



Replication

- Preference List
  - Each node maintains a list of k nodes
- Each node stores keys between its Nth predecessor to itself
- Node D will store the keys that fall in the ranges (A, B], (B, C], and (C, D].



Object Versioning

- Eventual Consistency
- Asynchronous update of replicas
- Client must specify which version of replica it is updating
- Dynamo uses Vector clocks to capture causality between different versions of the same object
- Vector clock = <node, counter> pairs
- Vector Clocks issues
  - Size of vector colcks grows very large if many servers coordinate the writes to an object

Object Versioning (cont…)

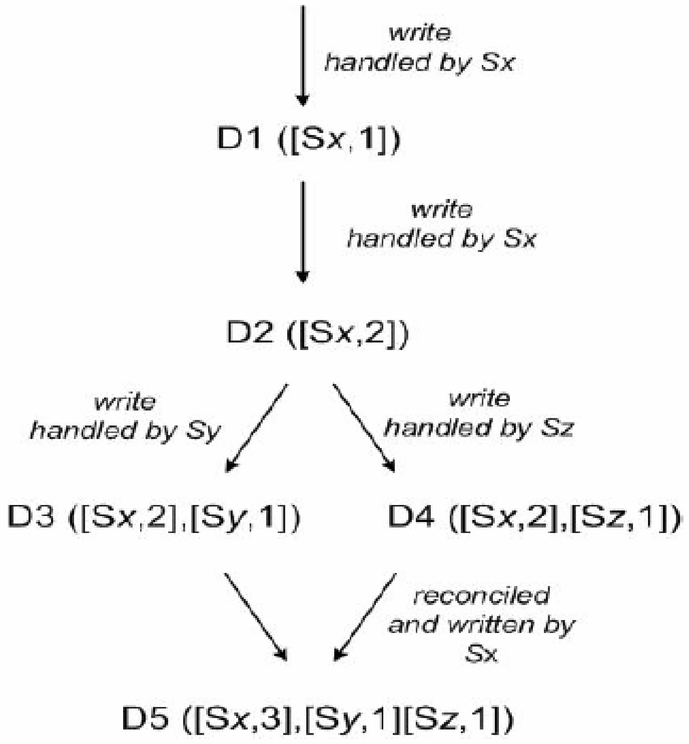


Figure 3: Version evolution of an object over time.

- Operations
  - Read: get()
  - Update: put()
- Consistency among replicas
  - R = number of nodes needed for successful read
  - W = number of nodes needed for successful write
- Adjust R,W => R + W > N
- Solution: R + W < N
- Why ?

  Outliers – latencies are dependent on slow replicas

Failure Handling

- Sloppy Quorum
  - Does not enforce strict quorum membership
- Hinted Handoff
  - Hides temporary node or network failure.
  - Replica temporarily stored with one of the K nodes
  - Returned to original node on recovery
- Problems with Hinted Handoff
  - Works well only when the churn in low
- IF hinted replicas are lost …!
- Solution: Anti-Entropy => Merkel Trees

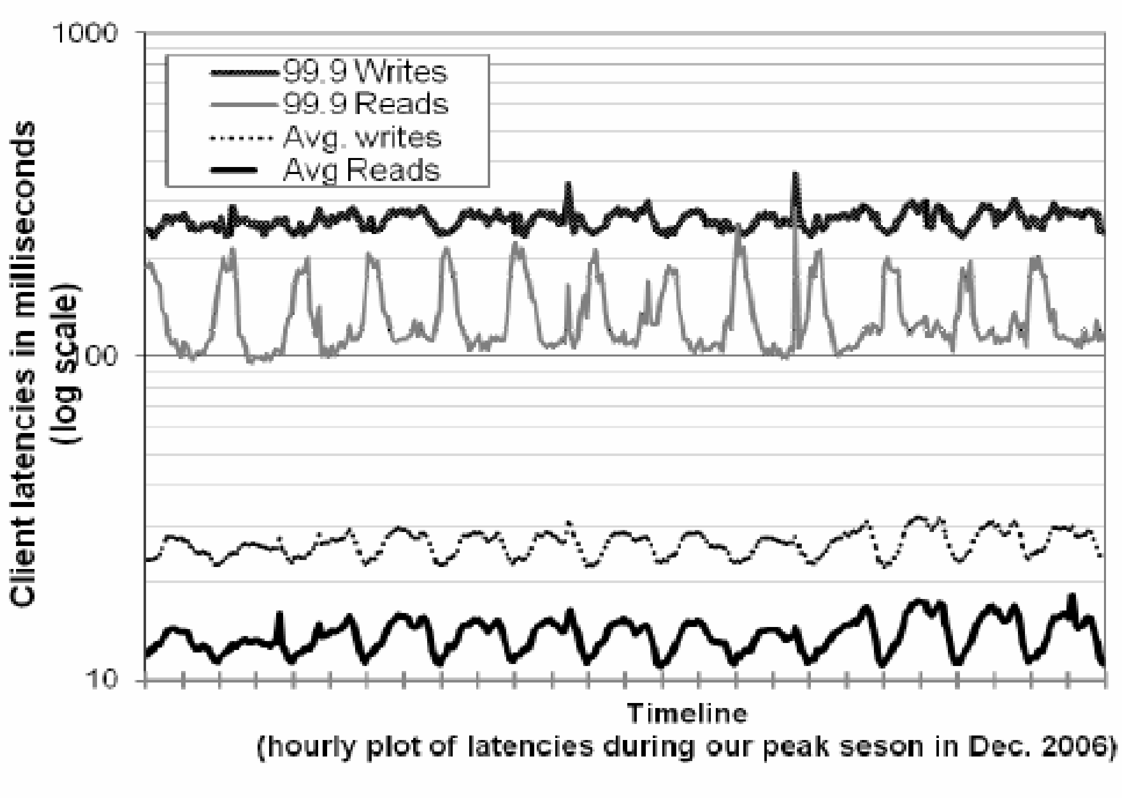
Failure Handling

- Merkel Trees

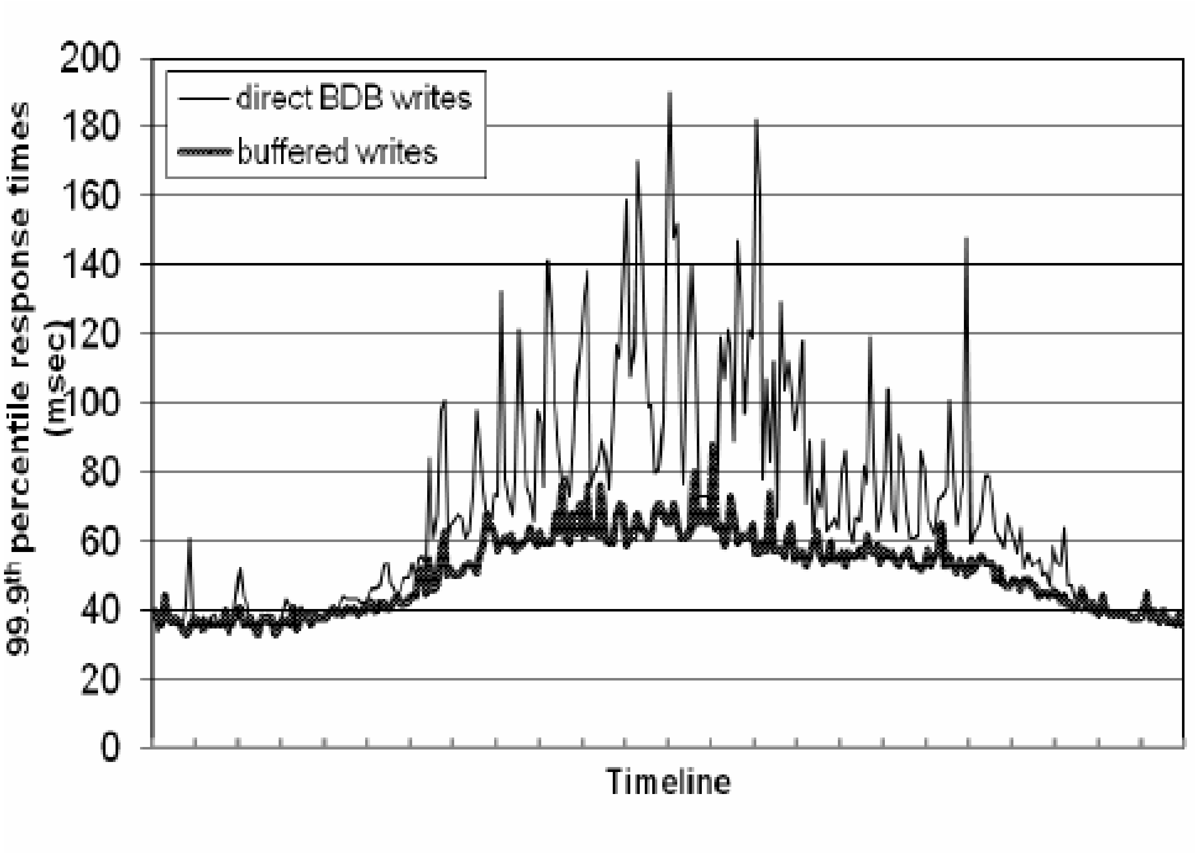
  Adv : minimizes the amount of data transfer between nodes for synchronization

  Disadv: When nodes join/leave, the tree has to be recalculated
- Gossip based protocol propagates membership changes
- .

Results


1000
100
10
99.9 Writes
99.9 Reads
Avg. writes
Avg Reads
Client latencies in milliseconds (log scale)
Timeline
(hourly plot of latencies during our peak seson in Dec. 2006)

Results


200
180
160
140
120
100
80
60
40
20
0
direct BDB writes
buffered writes
99.9th percentile response times (msec)
Timeline

Results

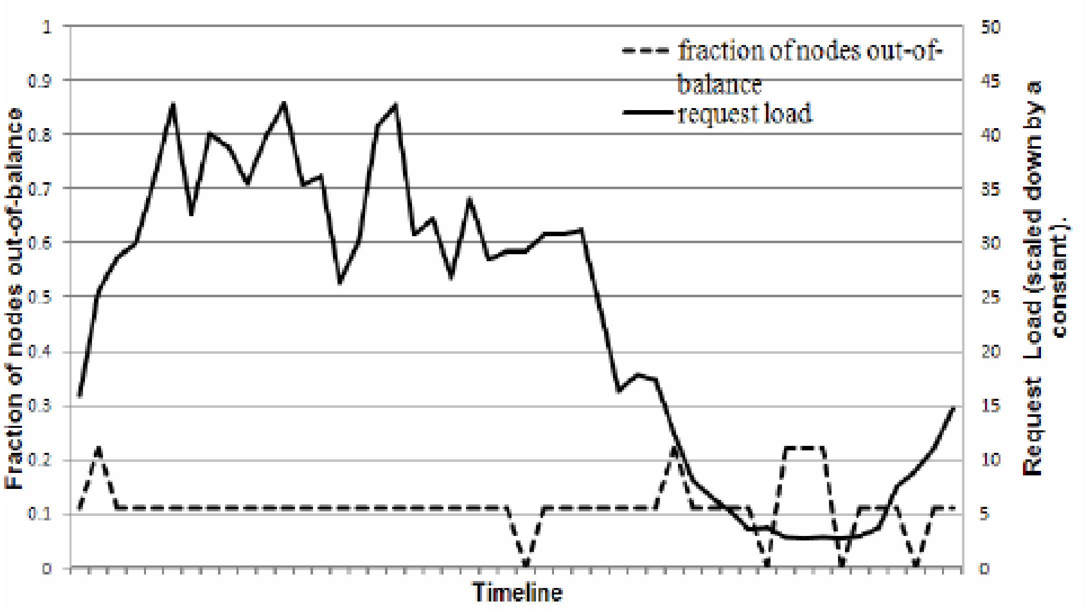
Number of divergent versions :

One version = 99.94%

2 versions = 0.00057%

3 versions = 0.00047%

4 versions = 0.00009%



Final Thoughts

- Tuning knobs for various uses

  Tune quorum and replication parameters (*N,R,W*)

  Can trade durability for performance (mem buffer, *W* = 1)
- How to read/write multiple objects?
- Burden on Application – application logic becomes more complex
- How do they determine capacity
- If a node goes down - all virtual nodes corresponding to it are lost.- merkel trees reconstruction, R+W factor.