Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Dynamo
Amazon's Highly Available Key-value Store

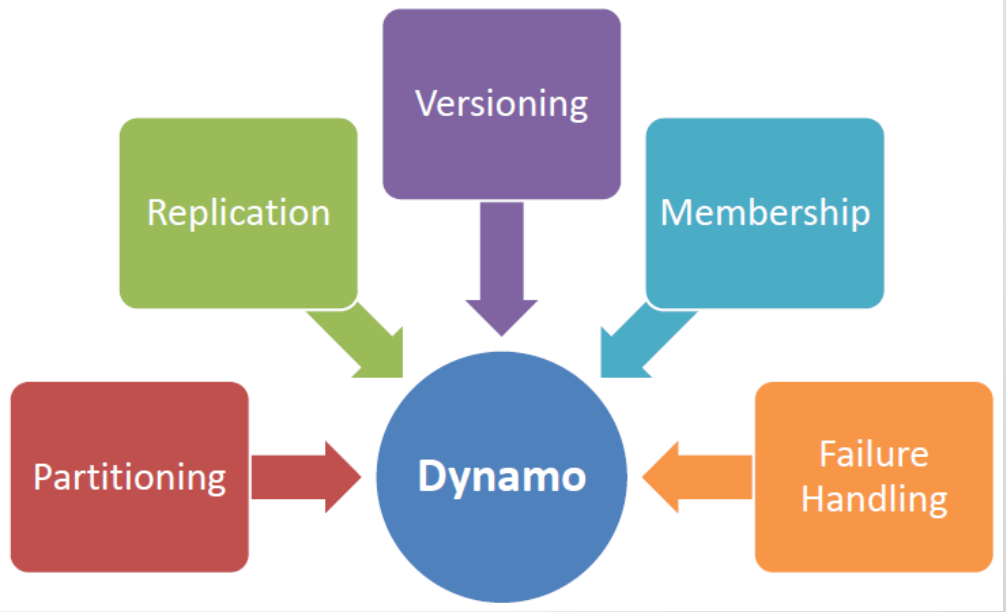
Miroslav Cupák

07/02/2013

Some slides borrowed from Steven van Beelen.

Outline

- What's the problem? What's Dynamo?
- Background
- How does Dynamo meet requirements?
- Evaluation
- Conclusions



Problem

- Amazon
  - major e-commerce provider
  - many customers that need to be satisfied
- requires a storage system
  - high availability
  - reliability at a massive scale
  - scalability
  - fault tolerance

Dynamo

- simple
  - datastore
- precisely
  - key-value storage system
  - primary-key interface
  - NoSQL (1 of ~150*)
  - zero-hop DHT
  - high availability over consistency
  - configurable for different services

* http://nosql-database.org/

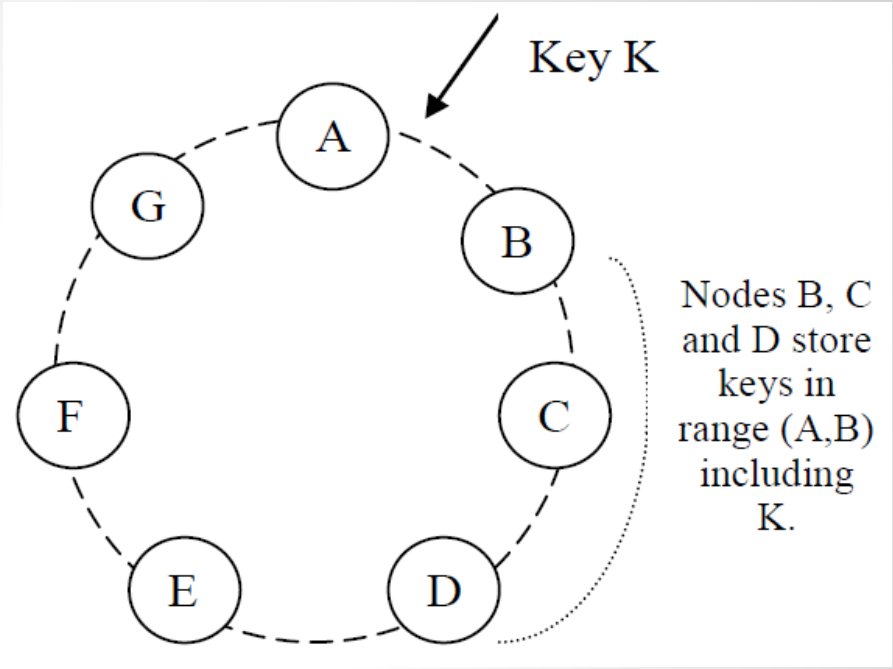
Dynamo

- principles
  - incremental scalability
  - symmetry
  - decentralization
  - heterogeneity
  - application dependence
  - always writable
  - *O(1)* routing (latency)
  - simple API: *get(key), put(key, context, object)*

Background

- similar to P2P systems & distributed DBs
- why not RDBMS?
  - focus on consistency
  - structured data
- assumptions
  - simple query model with ~~A~~"C"~~I~~D
  - security not important
- SLAs
  - response <300ms for 99.9+% of requests

Partitioning

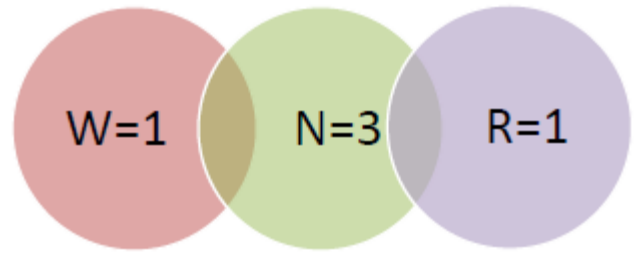


- dynamic
- consistent hashing
- "ring" space
  - node gets a position
  - data assigned to a node based on a key hash

    (walk clockwise to find the nearest greater node)
  - easy arrival/removal of nodes
  - each node has multiple positions (virtual nodes) due to heterogeneity in performance

Consistency

- eventual consistency model
- node handling a request = coordinator
- quorum-based consistency protocol
  - min. no. of replicas needed for read (R)/write (W)
  - coordinator only waits for R(W) responses before replying to the client



Replication

- data replicated to N hosts
- N per-instance
- coordinator replicates keys to N-1 successors
- node responsibility: <me, N-th predecessor>
- key has a preference list of physical hosts

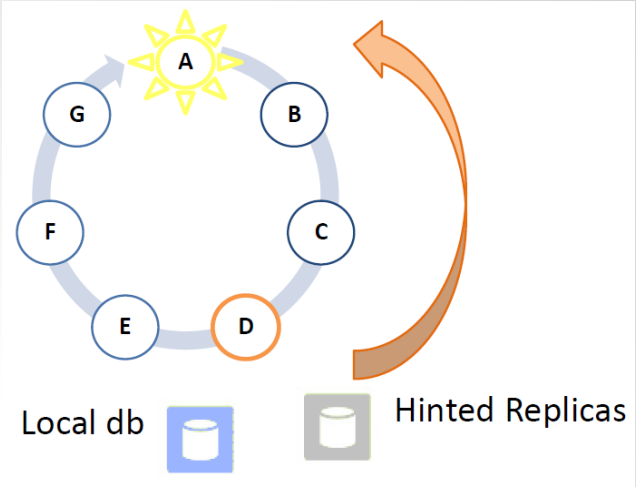
Versioning

- asynchronous updates
- concurrent updates -> conflicting versions
- metadata contains vector clocks
- versions reconciled by system/client
- application must tolerate inconsistencies
- clock truncation to avoid large vector clocks

Membership

- full membership model
- manual addition/removal of nodes in a ring
- change propagation: gossip-based protocol
- mappings reconciled with changes
- externally discovered seeds
  - known to all nodes
  - used to avoid logical partitions
- reallocation of keys upon addition/removal

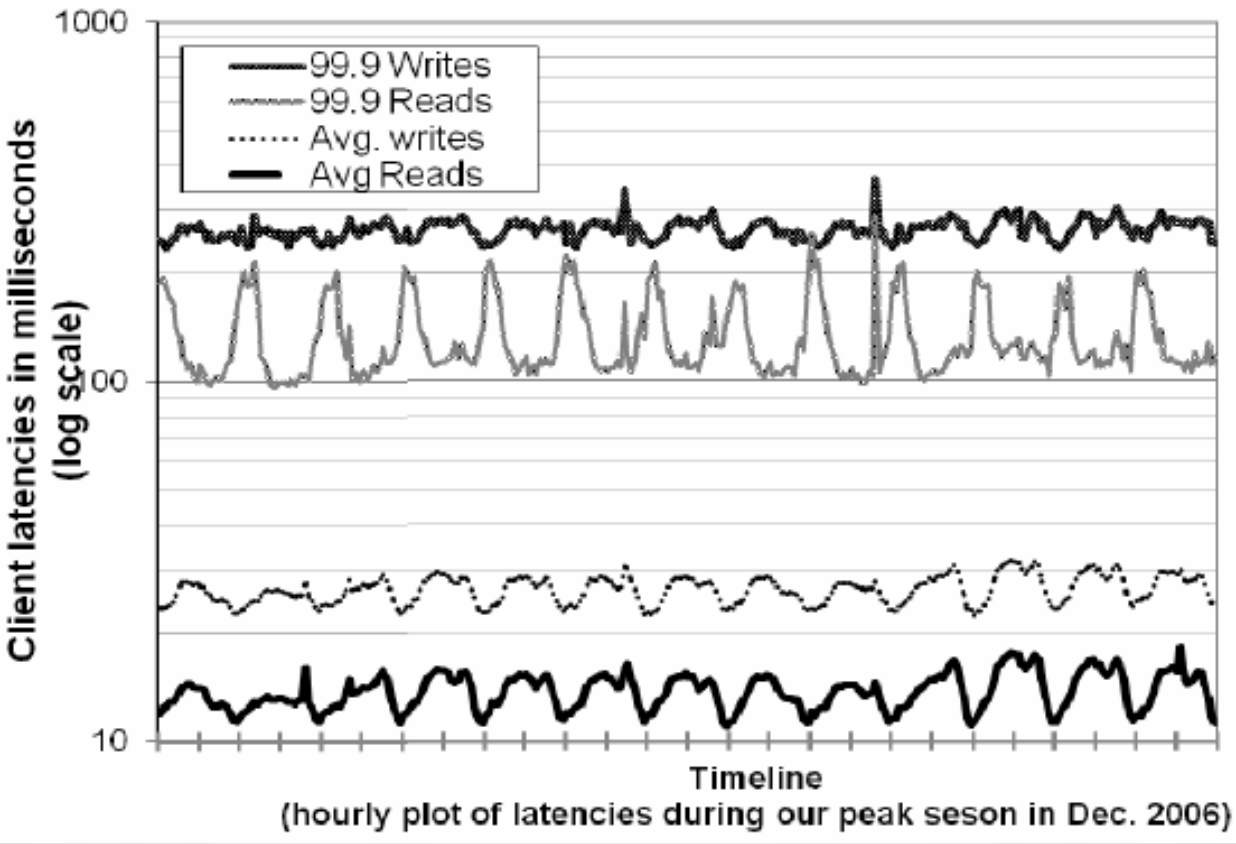
Failure Handling



- temporary failures
  - healthy nodes takes over
  - hinted handoff: node knows data is not his
  - periodic check for recovery
- permanent failures
  - replica synchronization based on key Merkle trees
  - no need to load the whole tree, usually root is enough
- large-scale failures
  - replication over multiple data centers

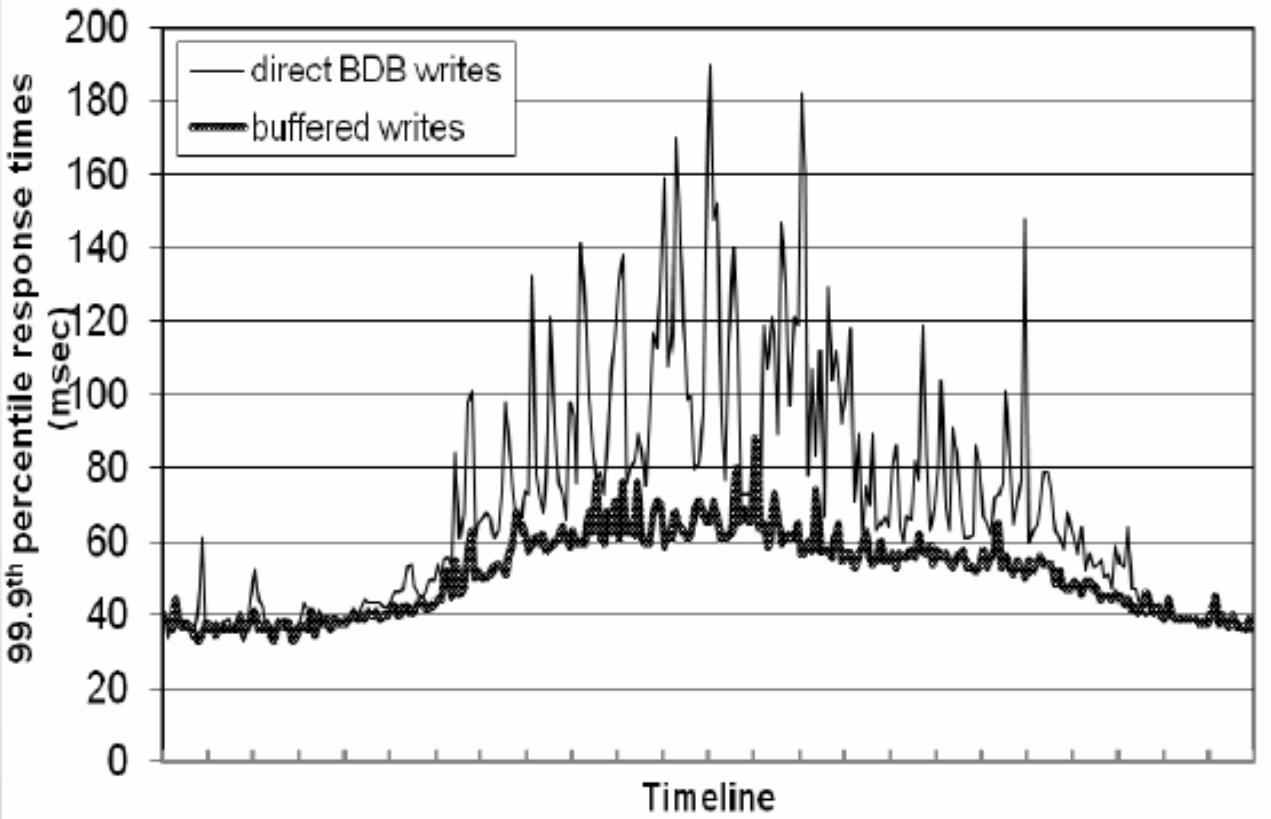
Evaluation - Availability/Latency

- real production environment
- (N,R,W) = (3,2,2), HW = ??? :-(



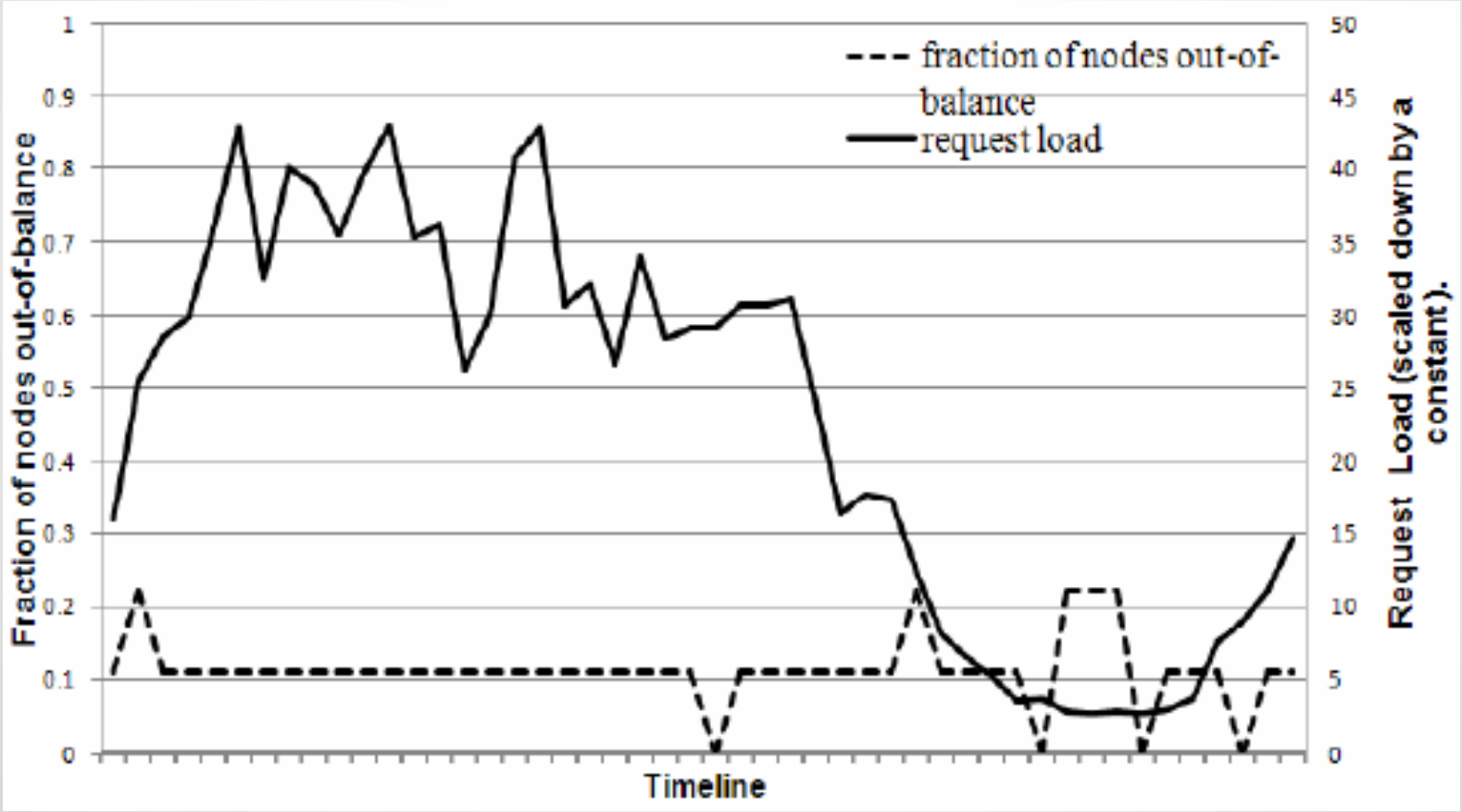
Evaluation - Performance/Durability

- buffered writes - operations check the buffer
- coordinator chooses so. for durable write



Evaluation - Load Distribution

- node imbalance vs. load



Evaluation - Version Divergence

- 99.94%: 1 version

  0.00057%: 2 versions
- very interesting, but no details due to "sensitive nature of the story" :-(

Conclusions

- provides high availability + scalability
- can be tailored to what the services need
- shows a new mixture of different techniques
- proves that eventual consistency can work in a very demanding environment
- just for Amazon :-(

Thanks! Questions?