Amazon Dynamo

distributed key-value storage

Michal Oniszczuk

October 10, 2012

Vast Distributed System

- Tens of millions of customers
- Tens of thousands of servers
- Failure is a normal case

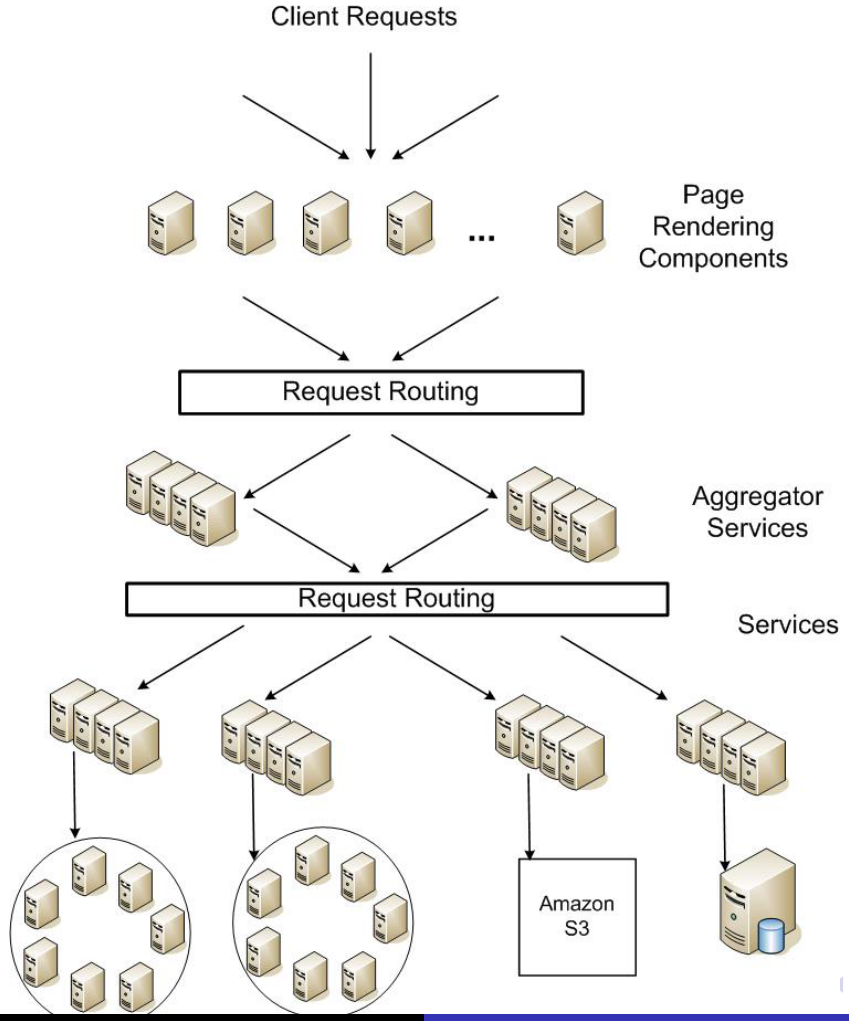
Outage means

- Lost Customer Trust
- Financial loses

Goal: great customer experience

- Always Available
- Fast
- Reliable
- Scalable

SOA - (*Service Oriented Architecture*)



- Simple put(key, object) and get(key) operations
- Targets small objects ($< 1MB$)
- No operations on multiple data
- Relational databases are not needed, they do not scale

ACID Properties

(Atomicity, Consistency, Isolation, Durability)

- Properties that guarantee that database transations are processed reliably
- Atomicity ("A") does not apply
- Weaker Consistency ("C")
- No Isolation ("I")
- Dynamo is configurable per application. Tradeoffs between: Durability ("D"), Availability, Performance, Cost Efficiency

SLA – Service Level Agreements

Example: *Service A is demanding key-value storage for objects that in 99.9% of cases can be read in 300 ms*

Just one number is describing the agreement - the latency of 99.9% percent of cases. Each application in Amazon's architecture must obey the performance contract.

In Amazon it turns out that more sophisticated ways of describing SLAs such as mean, median, average and variance are not good enough.

- No security guaranteed
- Each service runs its own instance of Dynamo
- Design targets hundreds of hosts

Usually: strong consistency

Data is unavailable until all storage replicas (copies) are the same.

Other way: optimistic replication

Changes are allowed to *propagate*. This causes *conflicts*

When to resolve *conflicts*?

- Each write **must** be successful.
- The conflict resolution is pushed to **read** operations.

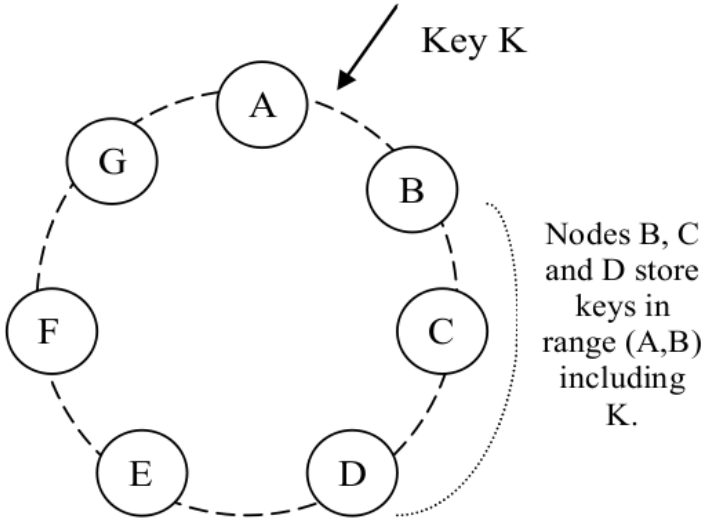
Who performs the process of conflict resolution?

- the client application
- data store itself

Design principles

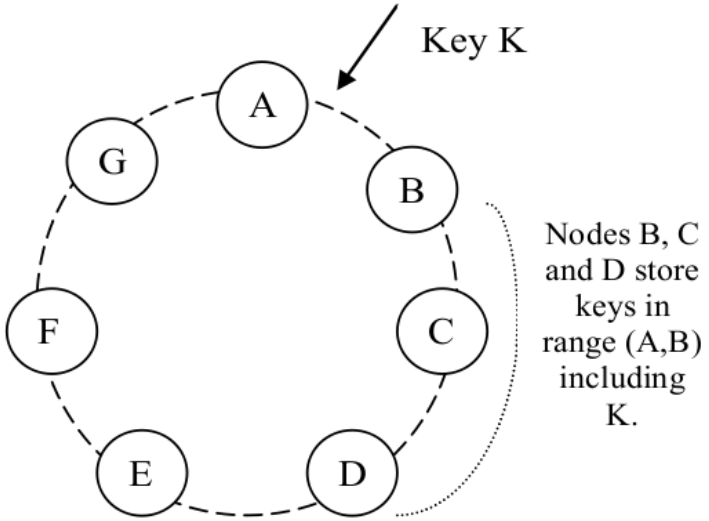
- Incremental scalability
- Symmetry
- Decentralization
- Heterogeneity

- $put(key, context, object)$ - stores replicas of *object* under the *key*. *context* is used to store the metadata used by Dynamo to resolve conflicting versions.
- $get(key)$ - returns the object and its context. It may return multiple results.



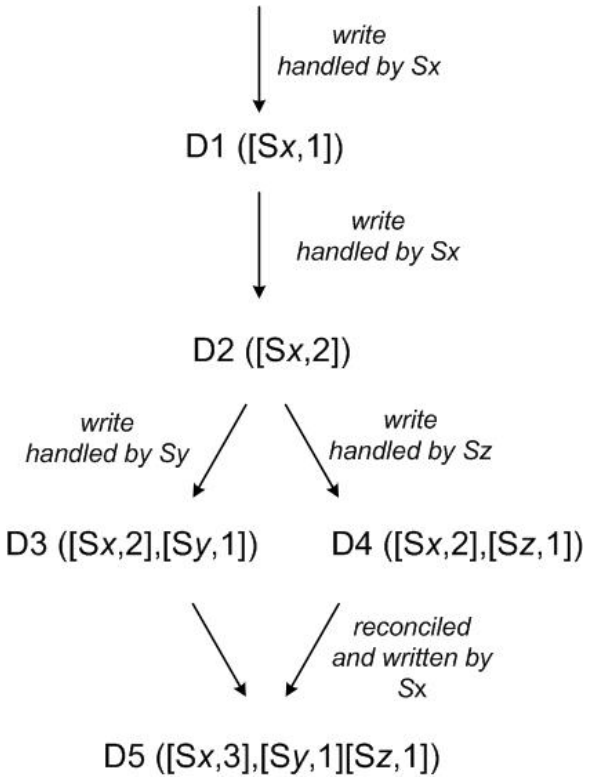
Consistent Hashing

- Each key is assigned to *coordinator node* (first clockwise encountered node on the ring from key's position)
- Virtual nodes (tokens)



Coordinator stores object both locally and also at $N - 1$ clockwise successor nodes in the ring.

- Vector clocks - a list of tuples $[(node, counter)]$
- Syntactic reconciliation
- Semantic reconciliation



Key properties

- get and put are invoked over HTTP (Amazon's internal request processing framework)
- it is possible to use generic load balancer - then *internal forwarding*
- or else the client may use the library that routes requests directly to the appropriate coordinator nodes

- Two configureable values R and W
- W - minimum number of nodes that must participate in successful *write* operation
- R - minimum number of nodes that must participate in successful *read* operation

Hinted handoff

- concept of *healthy nodes*
- connected with *only distinct physical nodes* approach

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

- N - number of tokens that are responsible for storing data from particular range in hash space
- W - minimum number of nodes that must participate in successful *write* operation
- R - minimum number of nodes that must participate in successful *read* operation

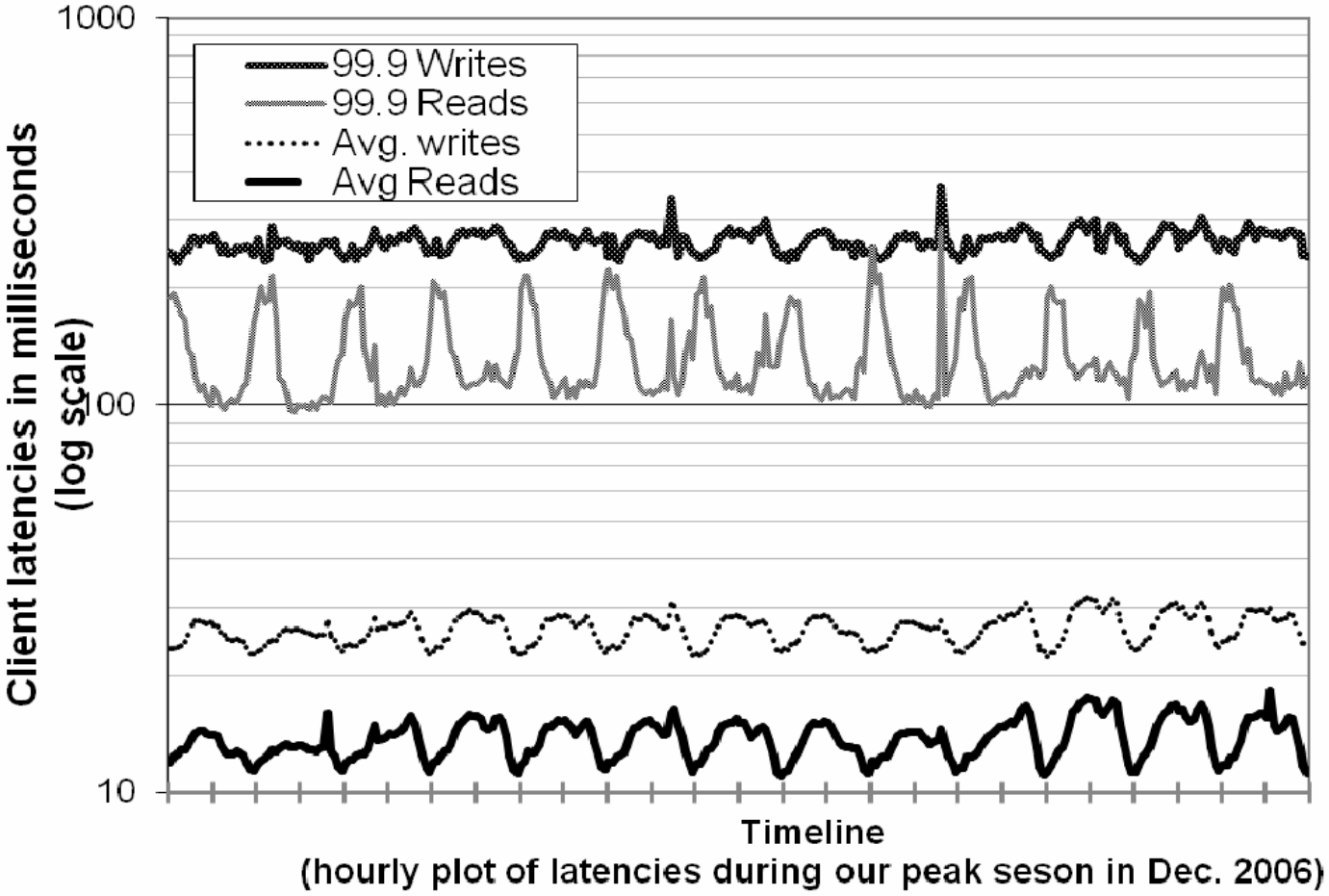
The most common configuration of this values in production evironment was $(N, W, R) = (3, 2, 2)$.
For the massively read storage it could be set to $(N, W, R) = (3, 1, 1)$.

- Berkeley DB
- MySQL
- tailored *in-memory buffer* with persistent backing store

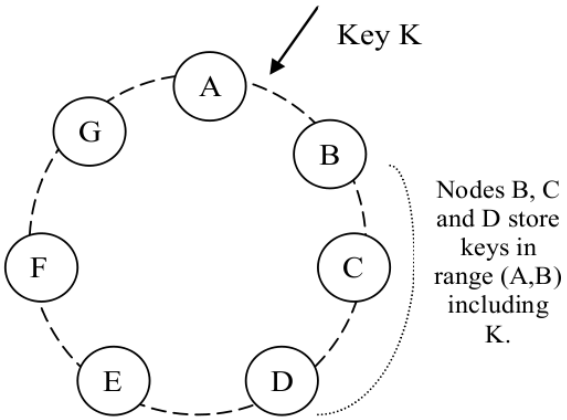
- 99.9995% of applications' calls had been returned successfuly without timing out
- **no data loss** have occured during measurements
- if client's application perform using libraries some of the request coordination it reduces latencies by 50%
- it turns out that: 99.94% of requests saw exactly one version of object, 0.00057% saw two versions, ...

Further problems

Balancing Performance & Durability Ensuring Uniform Load Distribution


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)

- P2P systems (Gnutella)
- Distributed File Systems and Databases (GFS, BigTable)



Dynamo

- highly available and scalable
- configurable

Questions

Some slides are used from presentation by Marcin Walas.

Based on the article Dynamo: Amazons Highly Available Key-value Store by Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels.