In an infrastructure with millions of components, something is always failing!
– Failure is the normal case

A lot of services within Amazon only need primary-key access to data
– Best seller lists, shopping carts, preferences, session management, sales rank, product catalog
– No need for complex querying or management offered by an RDBMS
– Full relational database is overkill: limits scale and availability

Amazon Dynamo: not exposed as a web service
– Used to power parts of Amazon’s services
– Highly available, key-value storage system

Core Assumptions & Design Decisions

• Two operations: get(key) and put(key, data)
  – Binary objects (data) identified by a unique key
  – Objects tend to be small (< 1MB)
• ACID gives poor availability
  – Use weaker consistency (C) for higher availability.
• Apps should be able to configure Dynamo for desired latency & throughput
  – Balance performance, cost, availability, durability guarantees.
• At least 99.9% of read/write operations must be performed within a few hundred milliseconds:
  – Avoid routing requests through multiple nodes
• Dynamo can be thought of as a zero-hop DHT

Consistency & Availability

• Strong consistency & high availability cannot be achieved simultaneously

• Optimistic replication techniques – eventually consistent model
  – Propagate changes to replicas in the background
  – Can lead to conflicting changes that have to be detected & resolved

• When do you resolve conflicts?
  – During writes: traditional approach – reject write if cannot reach all (or majority) of replicas
  – During reads: Dynamo approach
    – Design for all “always writable” data store – highly available
    – Reads/write operations can continue even during network partitions
    – Rejected customer updates won’t be a good experience
      – A customer should always be able to add or remove items in a shopping cart
# Consistency & Availability

- Who resolves conflicts?
  - **Choices:** the data store system or the application?

## Data store
- Application-unaware, so choices limited
- Simple policy, such as "last write wins"

## Application
- App is aware of the meaning of the data
- Can do application-aware conflict resolution
- E.g., merge shopping cart versions to get a unified shopping cart.
- Fall back on "last write wins" if app doesn't want to bother

# Reads & Writes

Two operations:

**get(key) returns**
1. object or list of objects with conflicting versions
2. context (resultant version per object)

**put(key, context, value):** stores replicas
- key is hashed with MD5 to create a 128-bit identifier that is used to determine the storage nodes that serve the key
- The nodes that hold replicas are based on the key
- Context ignored by the application but includes version of object

# Partitioning

- Break up database into chunks distributed over all nodes
  - Key to scalability
- Relies on consistent hashing
  - Regular hashing: change in # slots requires all keys to be remapped
  - Consistent hashing:
    - \( K/n \) keys need to be remapped, \( K = \) # keys, \( n = \) # slots

## Logical ring of nodes: just like Chord
- Each node assigned a random value in the hash space: position in ring
- Responsible for all hash values between its value and predecessor's value
- Hash(key): then walk ring clockwise to find first node with position>hash
- Adding/removing nodes affects only immediate neighbors

# Dynamo virtual nodes

- A physical node holds contents of multiple virtual nodes
- In this example: 2 physical nodes, 5 virtual nodes

## Advantages: balanced load distribution
- If a node becomes unavailable, load is evenly dispersed among available nodes
- If a node is added, it accepts an equivalent amount of load from other available nodes
- # of virtual nodes per system can be based on the capacity of that node
  - Makes it easy to support changing technology and addition of new, faster systems
Replication

- Data replicated on \(N\) hosts (\(N\) is configurable)
  - Key is assigned a coordinator node (via hashing)
  - Coordinator is in charge of replication

- Coordinator replicates keys at the \(N-1\) clockwise successor nodes in the ring

Versioning

- Not all updates may arrive at all replicas

- Application-based reconciliation
  - Each modification of data is treated as a new version

- Vector clocks are used for versioning
  - Capture causality between different versions of the same object
  - Returned as a context from a get() operation

Availability

- Configurable values
  - \(R\): minimum \# of nodes that must participate in a successful read operation
  - \(W\): minimum \# of nodes that must participate in a successful write operation

- Metadata hints to remember original destination
  - If a node was unreachable, the replica is sent to another node in the ring
  - Periodically a node checks if the originally targeted node is alive

- Data center failure
  - System must handle the failure of a data center
  - Each object is replicated across multiple data centers

Storage Nodes

Each node has three components

1. **Request coordination**
   - Coordinator executes read/write requests on behalf of requesting clients
   - State machine contains all logic for identifying nodes responsible for a key, sending requests, waiting for responses, retries, processing retries, packaging responses
   - Each state machine instance handles one request

2. **Membership and failure detection**

3. **Local persistent storage**
   - Different storage engines may be used depending on application needs
     - Berkeley Database (BDB) Transactional Data Store (most popular)
     - BDB Java Edition
     - MySQL (for large objects)
     - In-memory buffer with persistent backing store

Amazon S3 (Simple Storage Service)

Commercial service that implements many of Dynamo's features

- Storage via web services interfaces (REST, SOAP, BitTorrent)
  - Stores more than 449 billion objects
  - 99.9% uptime guarantee (43 minutes downtime per month)
  - Proprietary design
  - Stores arbitrary objects up to 5TB in size

- Objects organized into buckets and within a bucket identified by a unique user-assigned key

- Buckets & objects can be created, listed, and retrieved via REST or SOAP
  - `http://s3.amazonaws.com/bucket/key`

- Objects can be downloaded via HTTP GET or BitTorrent protocol
  - S3 acts as a seed host and any BitTorrent client can retrieve the file
  - Reduces bandwidth costs

- S3 can also host static websites
The end