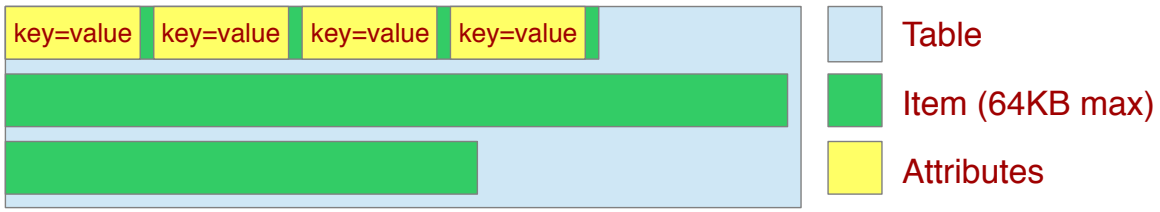
Amazon DynamoDB

Nicolas Travers
Inspiré de Advait Deo

Amazon DynamoDB – What is it ?

- Fully managed nosql database service on AWS
- Data model in the form of tables
- Data stored in the form of items (name – value attributes)
- Automatic scaling
  - Provisioned throughput
  - Storage scaling
  - Distributed architecture
- Easy Administration
- Monitoring of tables using CloudWatch
- Integration with EMR (Elastic MapReduce)
  - Analyze data and store in S3

Amazon DynamoDB – What is it ?



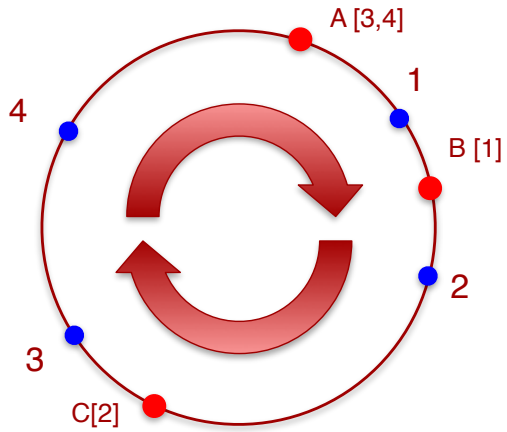
- Primary key (mandatory for every table)
  - Hash or Hash + Range
- Data model in the form of tables
- Data stored in the form of items (name – value attributes)
- Secondary Indexes for improved performance
  - Local secondary index
  - Global secondary index
- Scalar data type (number, string etc) or multi-valued data type (sets)

DynamoDB Architecture

- True distributed architecture
- Data is spread across hundreds of servers called storage nodes
- Hundreds of servers form a cluster in the form of a “ring”
- Client application can connect using one of the two approaches
  - Routing using a load balancer
  - Client-library that reflects Dynamo’s partitioning scheme and can determine the storage host to connect
- Advantage of load balancer – no need for dynamo specific code in client application
- Advantage of client-library – saves 1 network hop to load balancer
- Synchronous replication is not achievable for high availability and scalability requirement at amazon
- DynamoDB is designed to be “always writable” storage solution
- Allows multiple versions of data on multiple storage nodes
- Conflict resolution happens while reads and NOT during writes
  - Syntactic conflict resolution
  - Symantec conflict resolution

DynamoDB Architecture - Partitioning

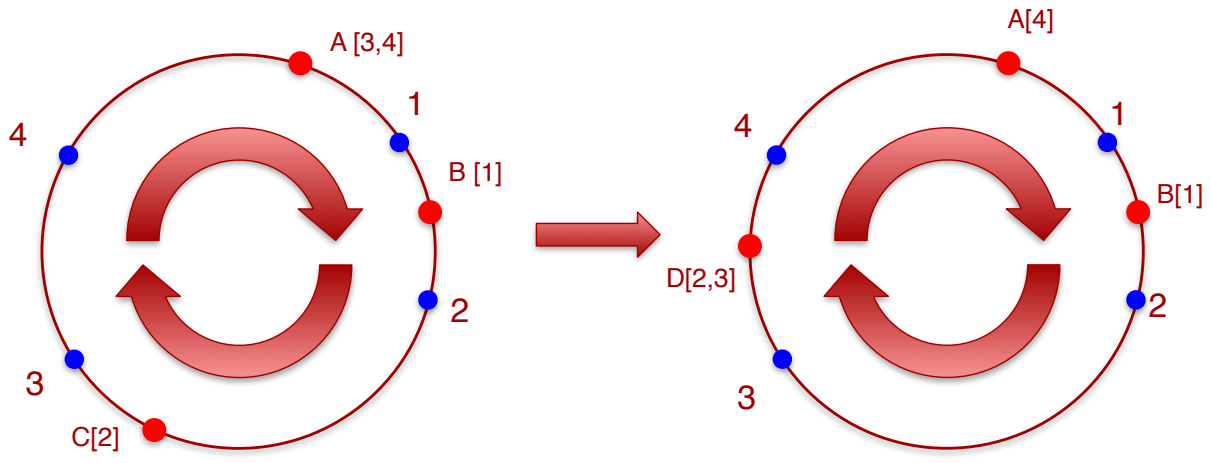
- Data is partitioned over multiple hosts called storage nodes (ring)
- Uses consistent hashing to dynamically partition data across storage hosts
- Two problems associated with consistent hashing
  - Hashing of storage hosts can cause imbalance of data and load
  - Consistent hashing treats every storage host as same capacity



Initial Situation

DynamoDB Architecture - Partitioning

- Data is partitioned over multiple hosts called storage nodes (ring)
- Uses consistent hashing to dynamically partition data across storage hosts
- Two problems associated with consistent hashing
  - Hashing of storage hosts can cause imbalance of data and load
  - Consistent hashing treats every storage host as same capacity

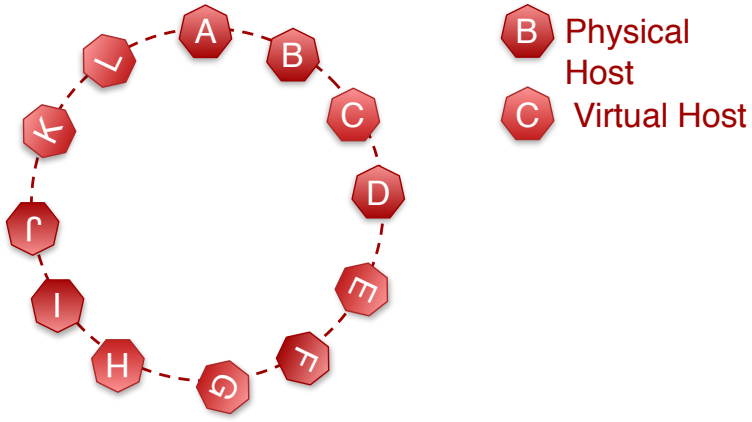


Initial Situation

Situation after C left and D joined

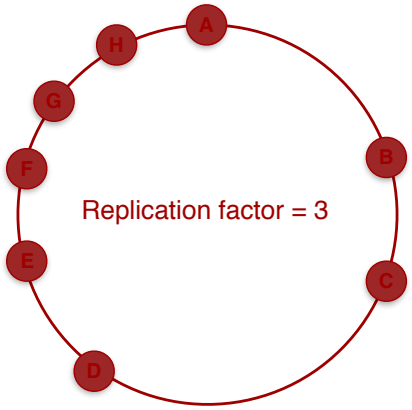
DynamoDB Architecture - Partitioning

- Solution – Virtual hosts mapped to physical hosts (tokens)
- Number of virtual hosts for a physical host depends on capacity of physical host
- Virtual nodes are function-mapped to physical nodes



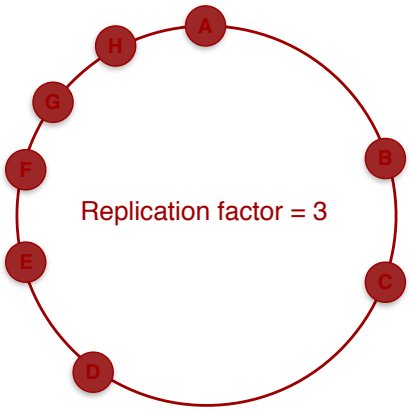
DynamoDB Architecture – Membership Change

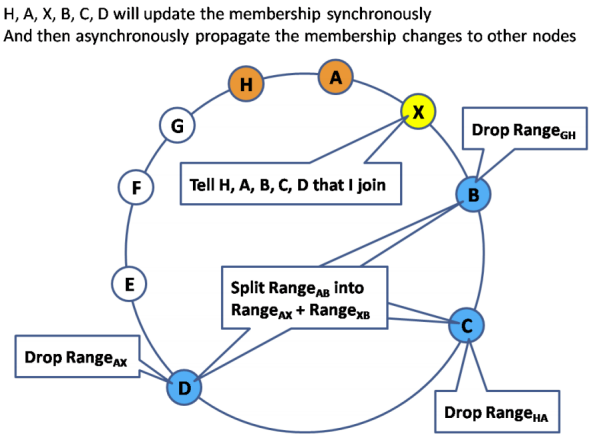
- Gossip protocol used for node membership
- Every second, storage node randomly contact a peer to bilaterally reconcile persisted membership history
- Doing so, membership changes are spread and eventually consistent membership view is formed
- When a new members joins, adjacent nodes adjust their object and replica ownership
- When a member leaves adjacent nodes adjust their object and replica and distributes keys owned by leaving member



DynamoDB Architecture – Membership Change

- Gossip protocol used for node membership
- Every second, storage node randomly contact a peer to bilaterally reconcile persisted membership history
- Doing so, membership changes are spread and eventually consistent membership view is formed
- When a new members joins, adjacent nodes adjust their object and replica ownership
- When a member leaves adjacent nodes adjust their object and replica and distributes keys owned by leaving member

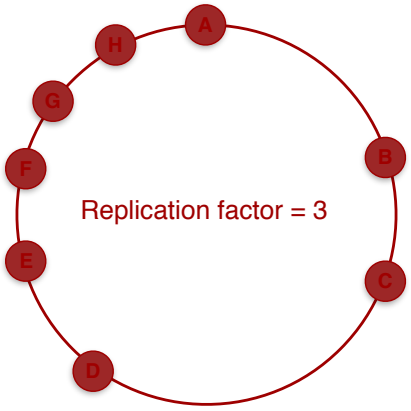


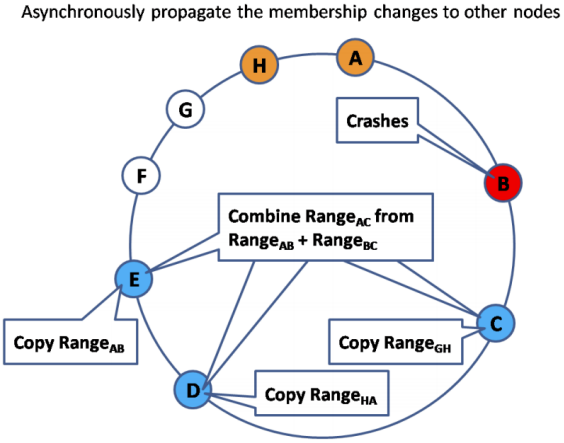


ESILV : Dynamo le cnam

DynamoDB Architecture – Membership Change

- Gossip protocol used for node membership
- Every second, storage node randomly contact a peer to bilaterally reconcile persisted membership history
- Doing so, membership changes are spread and eventually consistent membership view is formed
- When a new members joins, adjacent nodes adjust their object and replica ownership
- When a member leaves adjacent nodes adjust their object and replica and distributes keys owned by leaving member

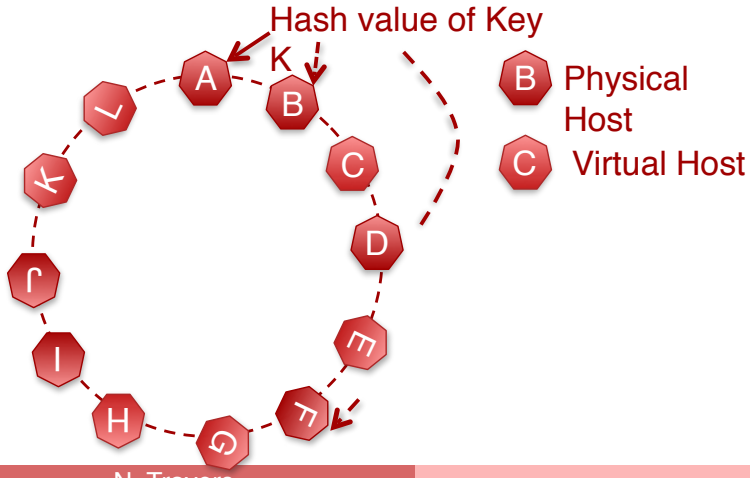




DynamoDB Architecture – Replication

- Each data item is replicated N times – replication factor
- Storage node in charge of storing key K replicates to its N-1 successor clockwise
- Every node has “Preference list” which has mapping of which key belongs where based on consistent hashing
- Preference list skips virtual nodes and contains only distinct physical nodes

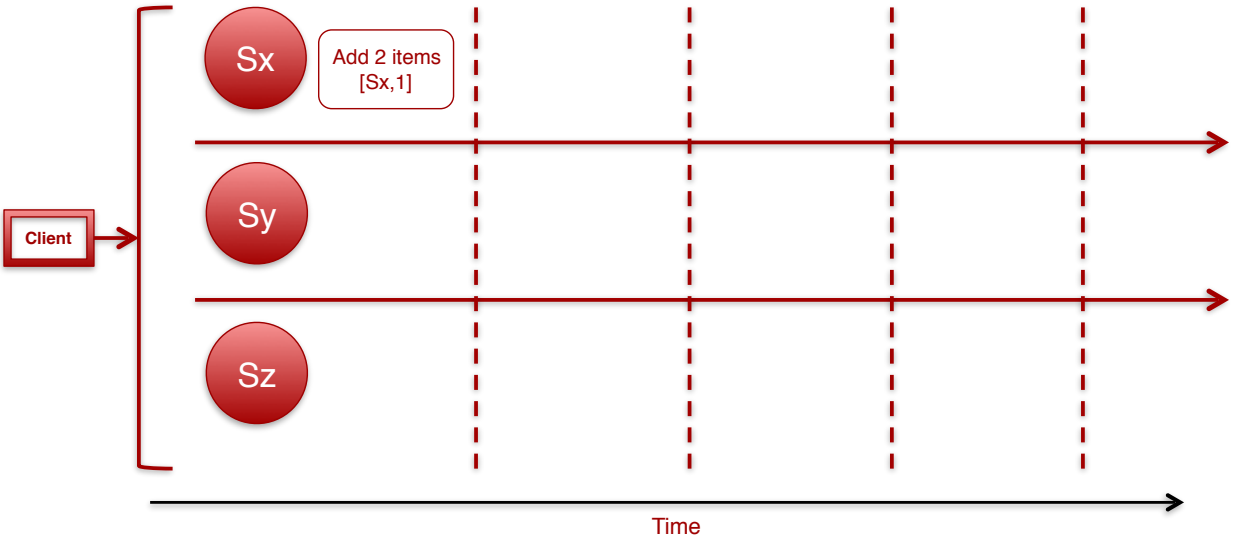
Preference list
A
B
D



DynamoDB Architecture – Data Versioning

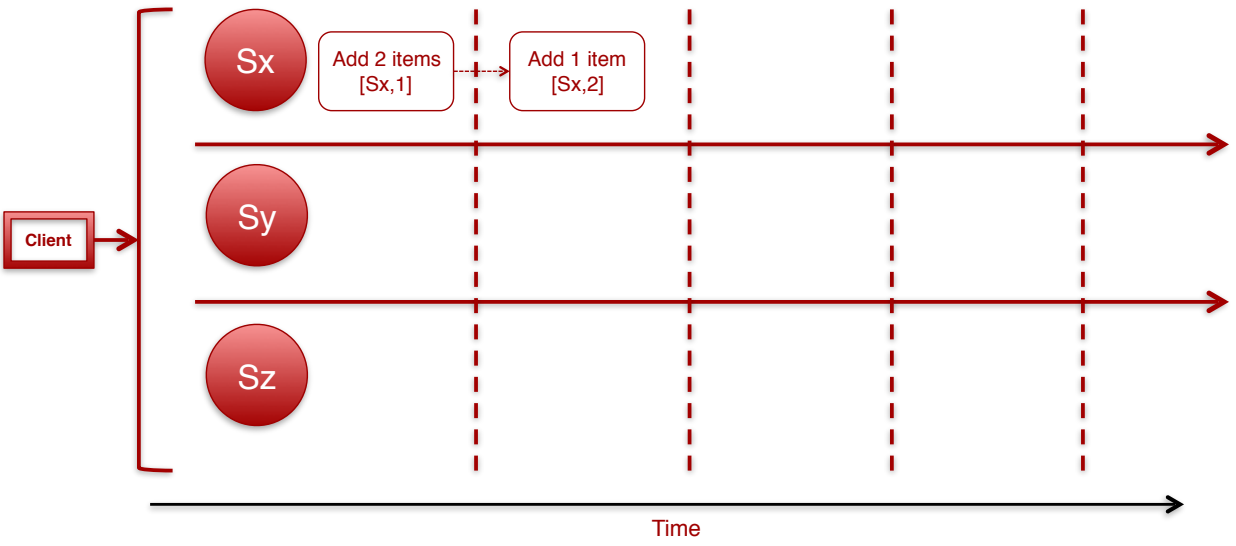
- Eventual Consistency – data propagates asynchronously
- Its possible to have multiple version on different storage nodes
- If latest data is not available on storage node, client will update the old version of data
- Conflict resolution is done using “vector clock”
- Vector clock is a metadata information added to data when using get() or put()
- Vector clock effectively a list of (node, counter) pairs
- One vector clock is associated with every version of every object
- One can determine two version has causal ordering or parallel branches by examining vector clocks
- Client specify vector clock information while reading or writing data in the form of “context”
- Dynamo tries to resolve conflict among multiple version using syntactic reconciliation
- If syntactic reconciliation does not work, client has to resolve conflict using semantic reconciliation
- Application should be designed to acknowledge multiple version and should have logic to reconcile the same

DynamoDB Architecture – Data Versioning

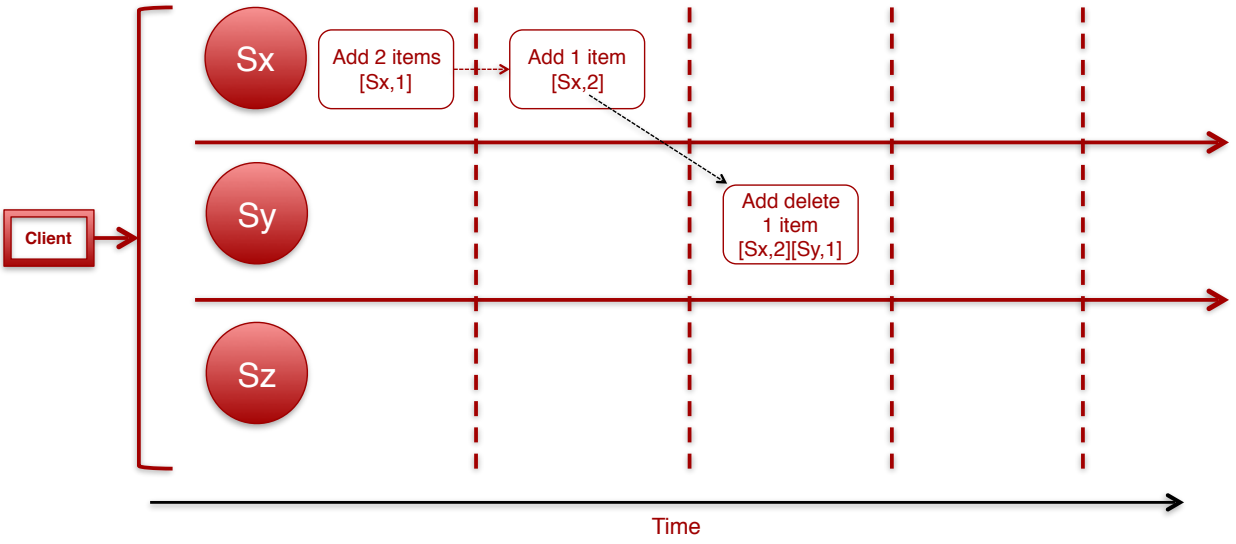


ESILV : Dynamo
le cnam

DynamoDB Architecture – Data Versioning

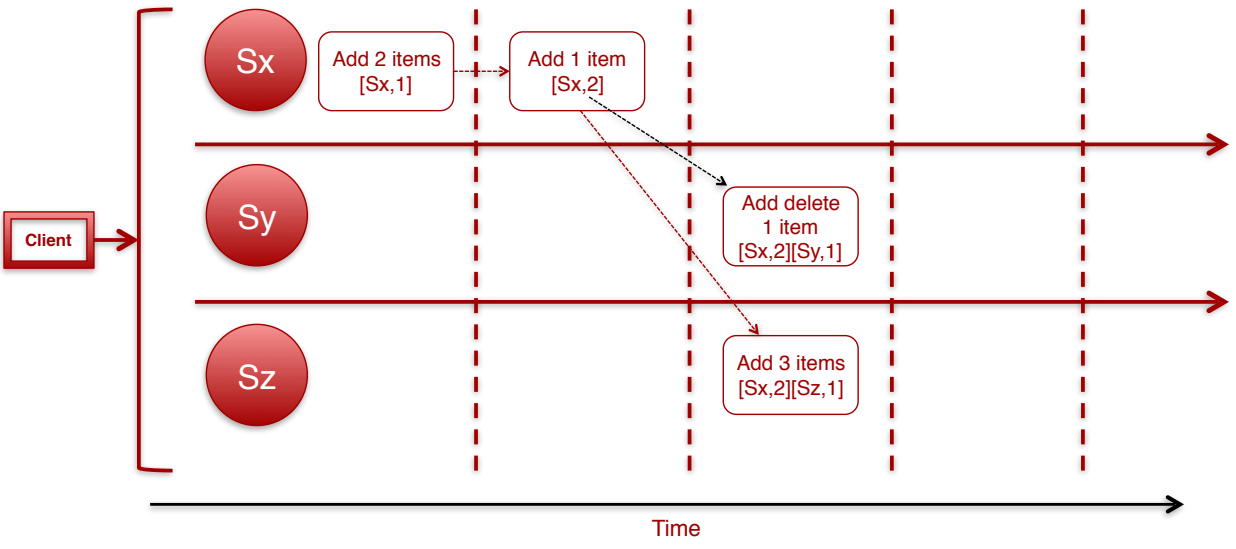


DynamoDB Architecture – Data Versioning

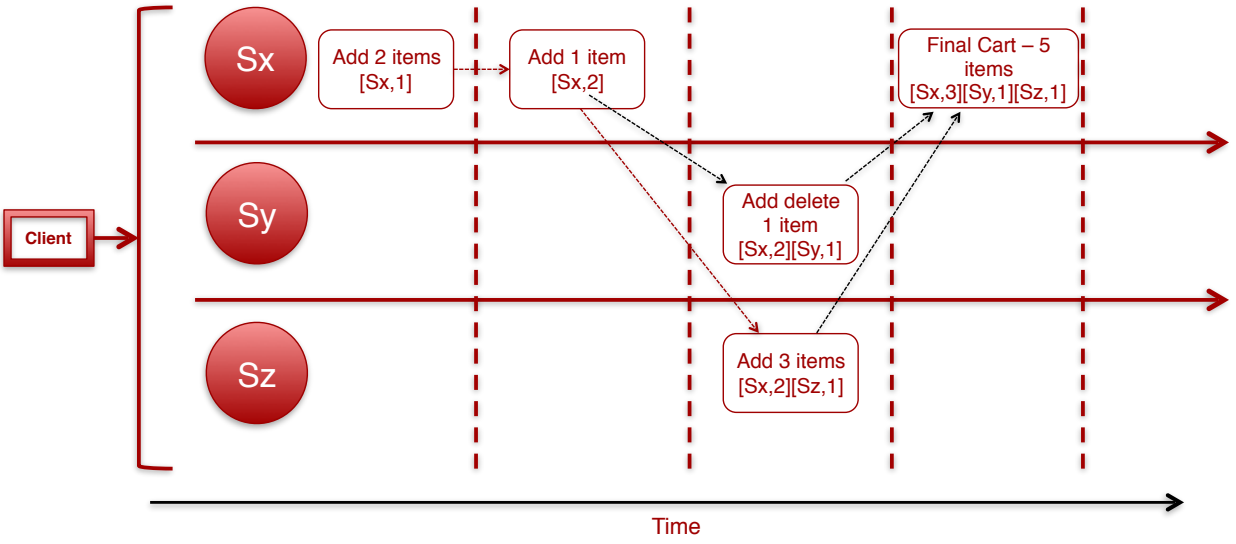


ESILV : Dynamo
le cnam

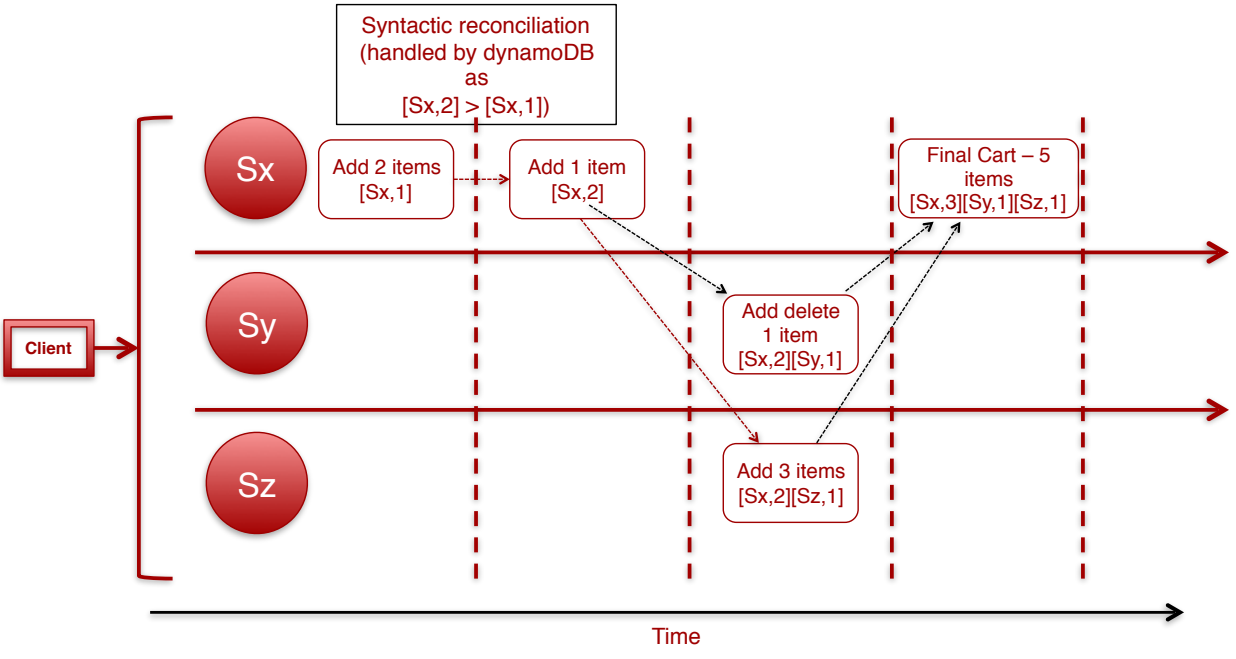
DynamoDB Architecture – Data Versioning



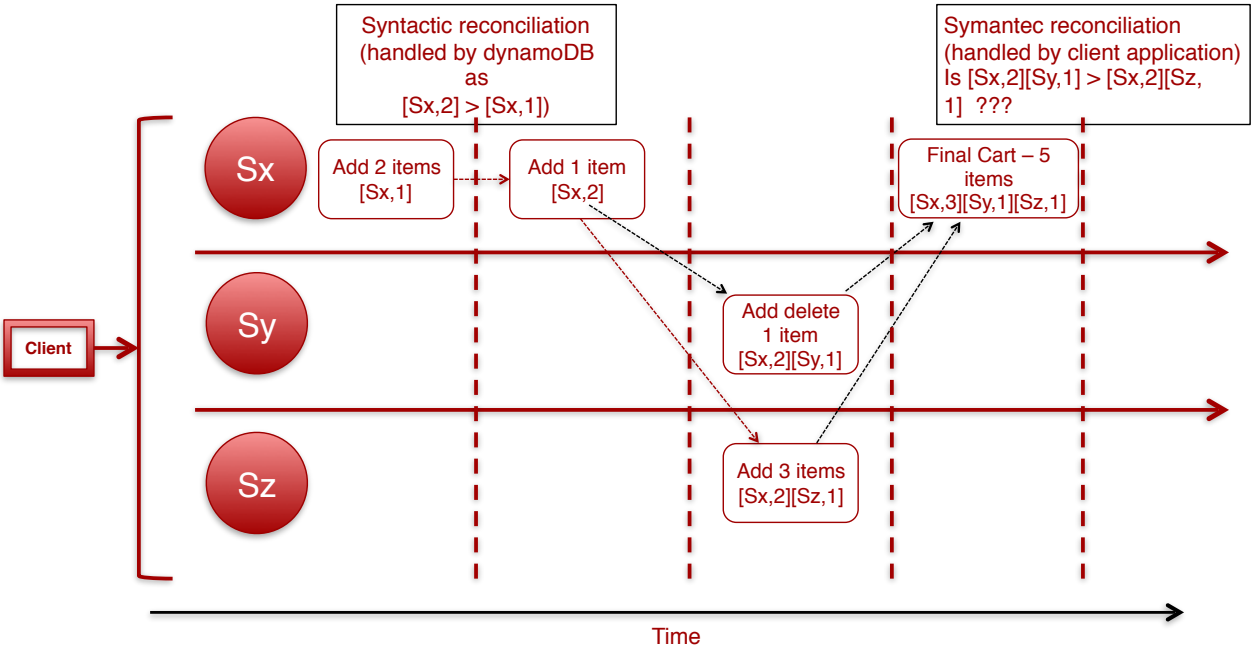
DynamoDB Architecture – Data Versioning



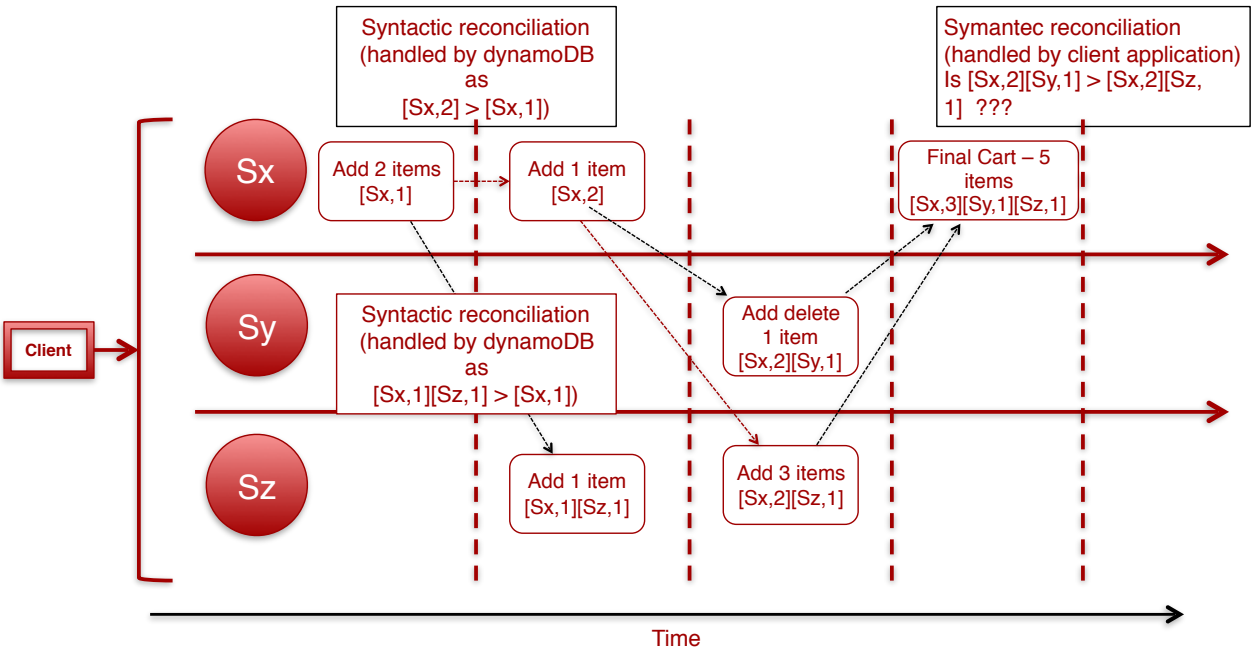
DynamoDB Architecture – Data Versioning



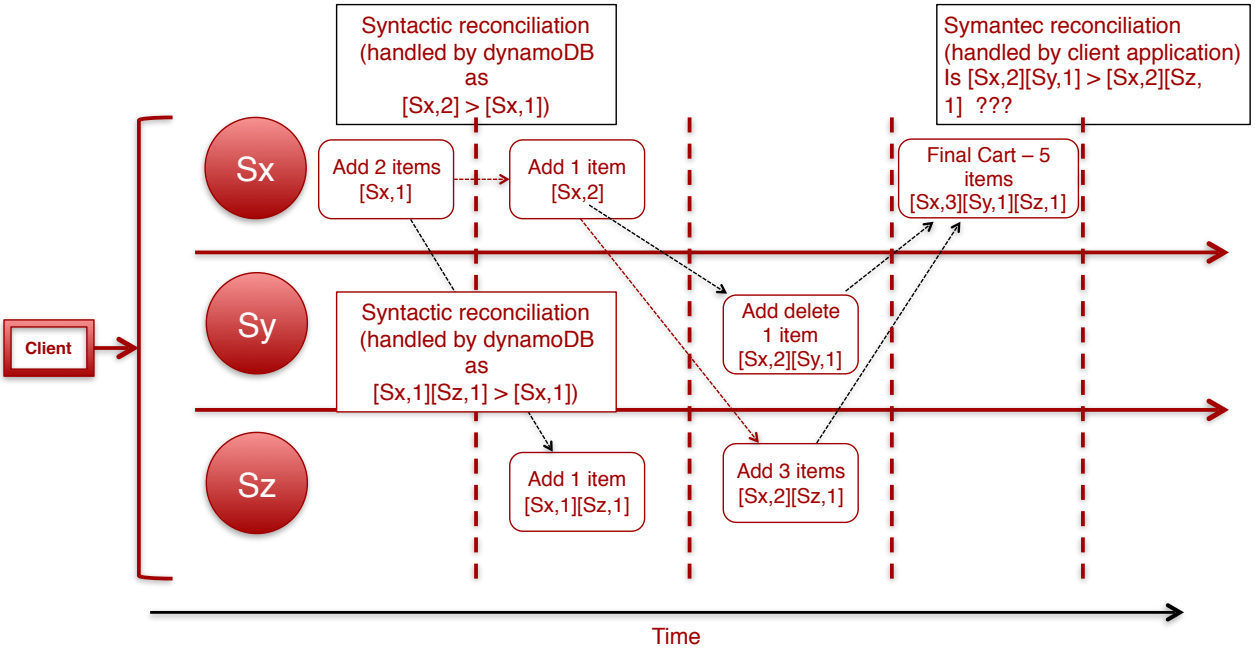
DynamoDB Architecture – Data Versioning



DynamoDB Architecture – Data Versioning



DynamoDB Architecture – Data Versioning



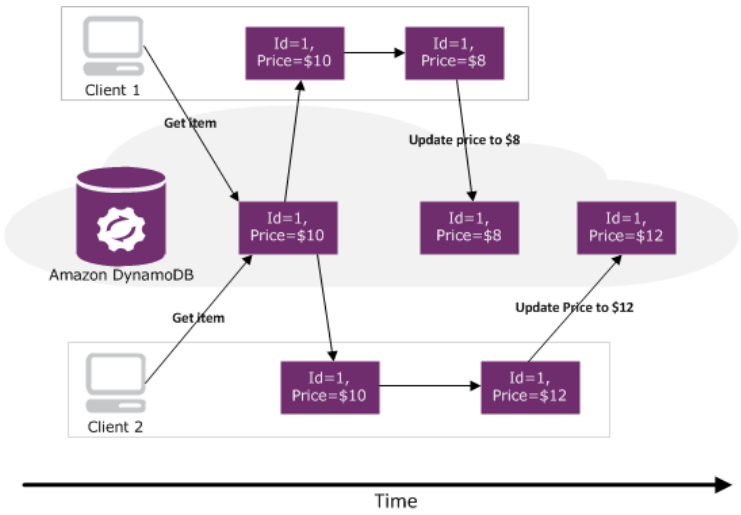
If counters on first object's vector clock are less than or equal to all of the nodes in second vector clock, then first is an ancestor of second and can be forgotten

DynamoDB Architecture – get() and put()

- 2 strategies client uses to select a node
  - Generic load-balancer that will select a node based on load
  - Partition aware client-library which routes request to right node
- Node handling read and write operation is called "coordinator"
- Coordinator should be among first in top N nodes in preference list
- For reads and writes dynamo uses consistency protocol similar to quorum system
- Consistency protocol has 3 variables
  - N – Number of replicas of data to be read or written
  - W – Number of nodes that must participate in successful write operation
  - R – Number of machines contacted in read operation
- Put()
  - Upon receiving put() request coordinator generates vector clock and writes data locally
  - Coordinator sends new version (along with vector clock) to top N nodes from preference list
  - If at least W-1 nodes respond, write is successful
- Get()
  - Coordinator requests all existing version of data from top N nodes from preference list
  - Waits for at least R nodes to respond
  - In case of multiple versions, coordinator send all casually unrelated versions to client
  - Client reconcile divergent versions and supersede current version with reconciled version

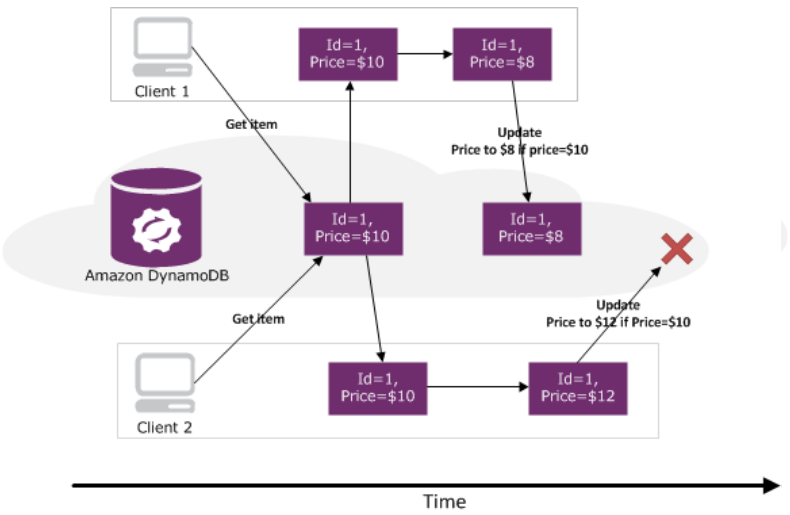
Conditional Writes

- Multiple clients can access the same item and try to update the same attributes



Conditional Writes

- Multiple clients can access the same item and try to update the same attributes



- idempotent operation – Running same request multiple times will have not effect

Strongly consistent vs Eventually Consistent

- Multiple copies of same item to ensure durability
- Takes time for update to propagate multiple copies
- Eventually consistent
  - After write happens, immediate read may not give latest value
- Strongly consistency
  - Requires additional read capacity unit
  - Gets most up-to-date version of item value
- Eventually consistent read consumes half the read capacity unit as strongly consistent read

Query vs Scan

- Query
  - Search based on primary key and examines only matching data
  - Maximum 1MB result
  - Query result sorted by range key
  - Query result can be opted to give strong consistency
  - Query can be done based on secondary index
  - Requires less provisioned throughput
  - Query performance depends on amount of data retrieved
- Scan
  - Examines every item and then applies filter
  - Maximum 1MB result
  - Scan result is always eventually consistent
  - Required high provisioned throughput
  - Scan performance depends on table size
  - Secondary index does not have any impact on scan performance
  - Parallel scan can be performed using `TotalSegments`

DynamoDB Parameters

- LastEvaluatedKey
  - Key of last returned value for operation > 1 MB
  - If LastEvaluatedKey is null then all data is transfered
- ExclusiveStartKey
  - Starting point of second query request for operation > 1MB
- Count
  - During request if count = true, dynamoDB provides only count(*) from table
- ScannedCount
  - Total number of items scanned in scan operation before applying any filter
- Limit
  - Limit number of rows in result
- Segments
  - Segment to be scanned by a worker in parallel scan
- TotalSegments
  - Degree of parallelism set in parallel scan

Secondary Index

- For efficiently accessing data using keys other than primary key
- Can only be created on scalar data type
- Can include projections from table into index
- Primary key attributes always part of secondary index
- DynamoDB copies items from table to index
- 2 types of secondary index
  - Local secondary index
  - Global secondary index
- Access to index similar as access to table
- Indexes are maintained automatically
- Maximum 5 local and 5 global secondary indexes for a table

Local vs Global Secondary Index

- Local Secondary Index
  - Hash + Range key
  - Has the same hash key as primary key. Range key defers
  - For each hash key total size of all indexed items <= 10GB
  - We can choose between eventual consistency or strong consistency
  - Consumes read capacity units from table
  - Write to table consumes write capacity from table to update index
  - Can request other table attributes which are not part of local index
- Global Secondary index
  - Hash or Hash + Range
  - Hash key different that hash key of primary key
  - No Size restrictions
  - Supports only eventual consistency
  - Has own provisioning throughput and doesn't use table's throughput
  - Can only request attributes which are part of global index

Item Collections

- Items with same hash key
- Includes table items and all local secondary indexes
- Max size allowed for any item is 10GB (applicable to tables with secondary indexes)
- If size exceeds, dynamoDB won't be able to add any data to item
- Table and index data for same item is stored in same partition
- View item sizes using `ReturnItemCollectionMetrics` parameter

Limitations of DynamoDB

- 64KB limit on item size (row size)
- 1 MB limit on fetching data
- Pay more if you want strongly consistent data
- Size is multiple of 4KB (provisioning throughput wastage)
- Cannot join tables
- Indexes should be created during table creation only
- No triggers or server side scripts
- Limited comparison capability (no not_null, contains etc)

References

- https://docs.aws.amazon.com/amazondynamodb/
- http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf
- http://home.aubg.bg/students/ENL100/Cloud%20Computing/Research%20Paper/nosqldbs.pdf
- http://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html - CLI reference
- http://aws.amazon.com/sdkforpython/ - SDK for python
- http://docs.pythonboto.org/en/latest/dynamodb_tut.html - boto tutorial and command reference
- http://en.wikipedia.org/wiki/Consistent_hashing