Amazon’s DynamoDB

@ryan_sb
sb@ryansb.com

What is Dynamo (not DB)

- Amazon paper from 2007 http://bit.ly/1mDs0Yh
- NOT the same as DynamoDB
- High availability NoSQL store
- Eventual consistency by default
- Tunable conflict resolution

What is DynamoDB

- High availability
- Low latency with 2-4ms query times
- Managed by Amazon “in the cloud”
- Provisioned for capacity, not # of nodes
- Closed source
- Only on AWS

Capacity?

- Every action takes DB capacity, of course!
- Queries and GET operations use read capacity units (RCU’s)
- Writes take write capacity units (WCU’s)
- Max of 40,000 WCU and 40,000 RCU per table

Ideal DynamoDB Uses

- Key-Value store with well-known query patterns
- The “hot” data from a larger SQL database
- Working table for Lambda/“stateless” pipelines

Bad DynamoDB Uses

- Adhoc querying
- Analytics/data science
- Spiky workloads

Fun Feature: Burst Capacity

- Reserves up to 5 minutes of unused read/write capacity
- Not **always** available due to maintenance
- Can help with unexpected spikes

Vocabulary (1/2)

- Attribute
  - A property on an object that has a name and a type. “id” and “S” (string) for example
  - Think of it like a column
- Item
  - A row in the DynamoDB table (collection of attributes)
- Table
  - Big (or less big…) collection of items

Vocabulary (2/2)

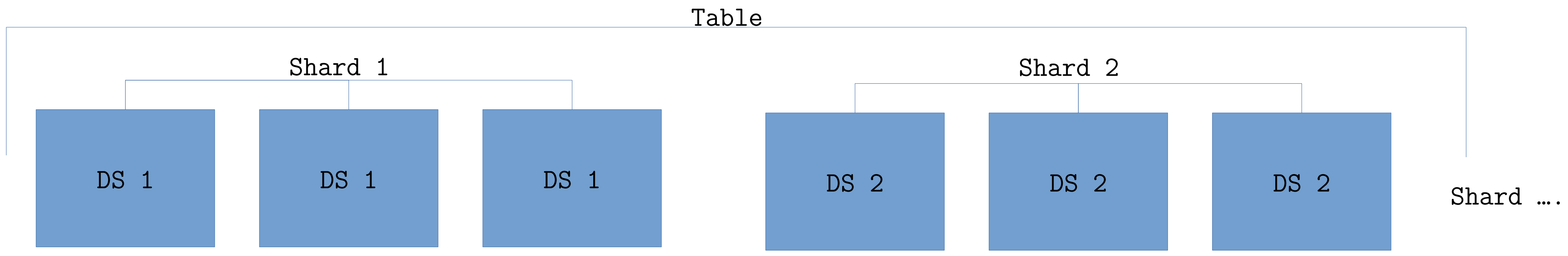
- Key
  - A blessed attribute that is used to find objects
  - Keys are always required
- Query
  - A search for objects meeting specific parameters that uses an index
- Scan
  - A search for objects that can’t use an index, which is much more expensive to perform

Vocabulary (3/2)

- Shard
  - A piece of the full database
- Replication
  - How data is stored. All data is stored 3x
- ProvisionedThroughputExceeded
  - An error that makes you very sad
  - No more capacity, come back later

How Many Shards

- Reads / 3000 + Writes / 1000 = Partitions
- 10 reads/10 writes: 10/3000 + 10/1000 <= 1
- 500/3000 + 200/1000 <= 1
- 4000/3000 + 2000/1000 < 4



Data Types

- String, Number, Binary, Boolean
- Number set, string set, binary set
- List (any type)
- Map (any type)

In the DynamoDB Engine

```
{
    "Id": {"S": "oid83274"},
    "owner": {"N": 23823974},
    "1": {"S": "7567c87a816f5f77e8fb9fb3d93fde30"},
    "2": {"S": "ed3a35212c1f9d38103d5633ab86fb62"}
}
```

Eventual Consistency

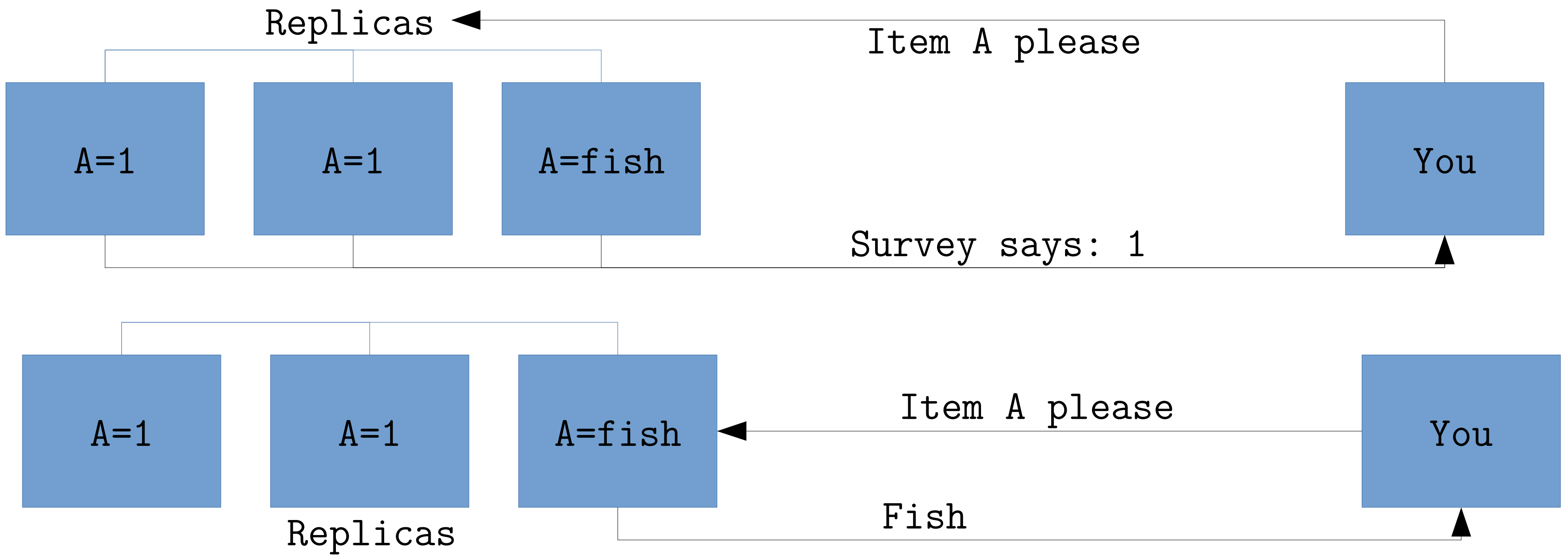
- MySQL and Postgres guarantee read-your-writes and monotonic reads
- DynamoDB doesn't have that guarantee
  - Cache your own writes
  - Consistent reads ($$$, up to 3x slower)

Python (with boto3)

```
>>> response = STATE_TABLE.get_item(
    Key={
        'Id': input_id
    },
    ConsistentRead=True
)
>>> print(json.dumps(response[‘Item’]), indent=2)
{
    "Id": "oid83274",
    "owner": 23823974,
    "1": "7567c87a816f5f77e8fb9fb3d93fde30",
    "2": "ed3a35212c1f9d38103d5633ab86fb62"
}
```

Consistent=true

- ConsistentRead is an **optional** flag
- Eventual consistency reads use 0.5 RCU



Easy Operators

- Scans are an iterator over a whole table
- ATTRIBUTE_EXIST, CONTAINS, and BEGINS_WITH
- =, >, <, <=, >=

Python (with boto3)

```
STATE_TABLE.query(
    ExpressionAttributeValues={":val": uname},
    ExpressionAttributeNames={
        "#p": "group_members"
    },
    KeyConditionExpression="(:val IN #p)"
)
```

Advanced Operators

- <> (not equals), BETWEEN
- Check set membership with IN
- Booleans for expressions: NOT, AND, and OR

Conditional Writes

- Condition: attribute_not_exists(ProfilePic)
- Action: SET ProfilePic = http://…..

Update Expressions (Action)

- SET likes = likes + :num
- SET likes = 11

Python (with boto3)

```
STATE_TABLE.update_item(
    Key={
        'Id': input_id
    },
    UpdateExpression="set #key=:val",
    ExpressionAttributeValues={":val": data},
    ExpressionAttributeNames={
        "#key": object_id,
        "#p": "processed"
    },
    ConditionExpression="attribute_not_exists(#p)"
)
```

Queries and Scans

- QUERY Name = “Ryan”
- QUERY Price < 10

IDs and Ranges

- Hash key is the first way to retrieve data
- Range key is (optional) secondary ordering
  - UID:epoch might be a way to order user actions
  - Range keys must be sortable
- Hash key and range key **must** be a unique composite

Know Your Questions

- Once a hash key is set, it’s permanent
- Hot shards can bite you later
- The number (and type, and size) of indexes are limited

Local Secondary Index

- Uses the same hash (UID)
- Different range key for querying and sorting
- Can't be deleted after creation

Local

- Contained inside the shard of the hash key
- Consistent with main table

Global Secondary Index

- Works across all shards
- Hash and range key are **both** configurable
- Great for queries on secondary attributes
- Separate throughput provisioning
- Updates can be inconsistent

Projected Attributes

- Keys only
  - Small, fast responses
  - Needs a second request to get the full object
- Include specific attributes
  - Nice medium – query planning pays off here
- All
  - Full duplication of the object, by far the most expensive option

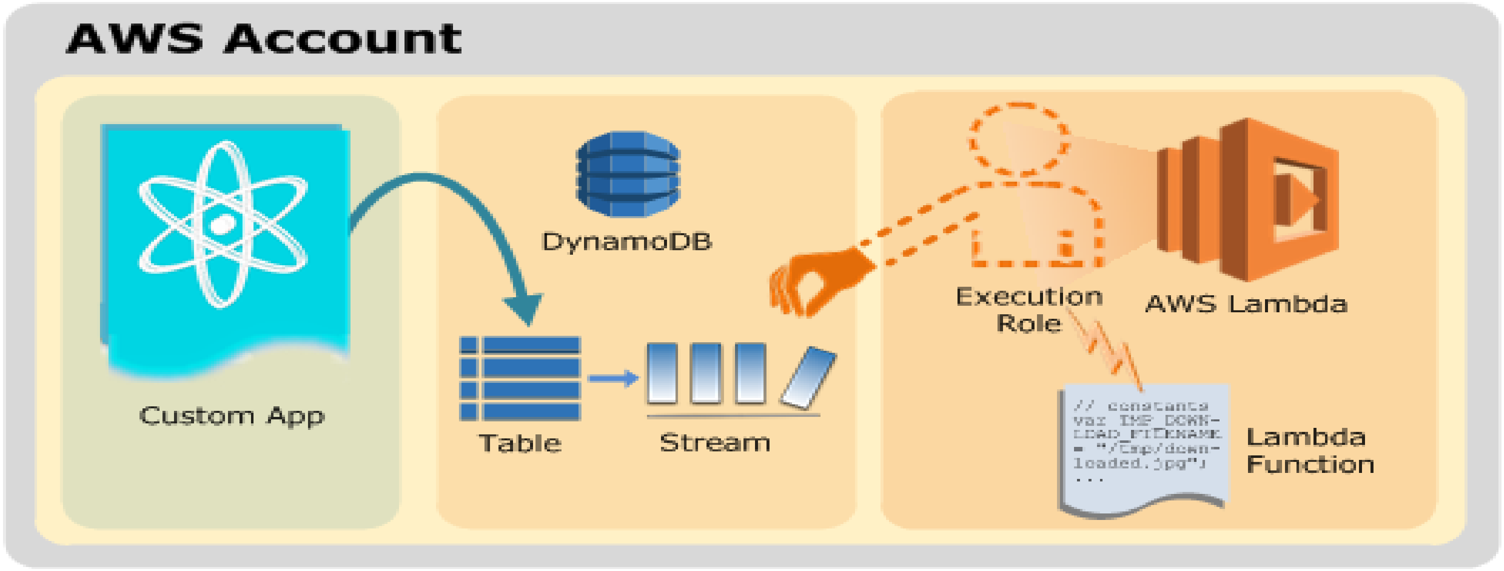
Limits

- Watch out for hot keys or ranges
  - Time series data
  - Super-active users
- 256 tables per region
- 400KB per item
- 5 LSI's and 5 GSI's per table
- 100 items per BatchGet
- 1MB of results per query or scan call

Big Items

- 400KB size limit **total**
- Store S3 URLs and fetch separately
- Compress or split payloads (not foolproof)

Streams



```
// constants
var TMP_DOWN-
LOAD_FILENAME
= "/tmp/down-
loaded.jpg";
...
```

Streaming

- Ordered (per stream of updates
- Hook up to Lambda functions for stored procedure-like work
- 24 hour lifetime
- Exactly once
- Retries on failed processing

Events

- Action: CREATE, MODIFY, REMOVE
- Keys: Hash & Range key of changed records
- NewImage: Current state of item in DynamoDB
- OldImage: Previous state (only in MODIFY or REMOVE events)

Saving $$$

- Caching in-application or on user sessions
- Decrease capacity at nonpeak times (4x/day)
- Reserved capacity for reads & writes
  - 45%-75% discount for buying ahead

Fin

Slides: https://rsb.io/talks/dynamodb.pdf

Twitter: @ryan_sb

Email: sb@ryansb.com