Using DynamoDB Transactions

Sean Shriver

DynamoDB/DAX SA
AWS

Girish Gangadharan

Enterprise Support Lead
AWS

May 1, 2020

aws

Agenda

- What are DynamoDB Transactions?
    - ACID compliance
- Multi-Item operations
    - Batch APIs
    - Transactions API
- Cost
- Examples
- Best practices
- Q&A

aws

What are DynamoDB Transactions?

- Make coordinated, all-or-nothing changes to multiple items both within and across tables
- Write or read a batch of items from DynamoDB, and the entire request will succeed or fail together

aws

What are DynamoDB Transactions?

- Provides ACID characteristics to maintain data correctness
    - Atomicity
    - Consistency
    - Isolation
    - Durability
- ACID support for single-item operations (from day one)
- ACID support for multi-item operations (launched in 2018)
- Simplifies complex workflows

aws

Batch Operations Vs. Transactions

- BatchGetItem
  - Up to 100 GetItem requests
  - Read up to 16MB of data
  - Across multiple tables
  - Allows partial success
  - Performs the individual read operations in parallel

Batch Operations Vs. Transactions

- BatchWriteItem
  - Up to 25 PutItem and/or DeleteItem requests
    - Does not support UpdateItem request
    - Cannot specify conditions
  - Write up to 16MB of data
  - Across multiple tables
  - Fails only if all requests fail
  - Allows partial success
  - Performs the individual write operations in parallel

Batch Operations Vs. Transactions

- Batch operations have 2 failure modes
    - Entire batch fails
        - Total request exceeds 16MB
        - More than 25 requests
        - Etc.
    - Individual requests fail
        - Throughput errors
        - Server side errors
        - Etc.

Batch Operations Vs. Transactions

- TransactionGetItems
  - Up to 25 items per request
  - All-or-nothing. No partial success/failure
  - Aggregate size cannot exceed 4 MB
  - Across multiple tables in the same region

Batch Operations Vs. Transactions

- TransactionWriteItems
  - Up to 25 items per request
  - All-or-nothing. No partial success/failure
  - Supports Idempotency (pass a ClientRequestToken)
  - Across multiple tables in the same region
  - Actions
    - Put
    - Update
    - Delete
    - ConditionCheck (different from items being processed)
  - No two actions can target the same item

aws

Batch Operations Vs. Transactions

- Entire TransactGetItems request fails if:
  - A conflicting update operation on item.
    - Transaction Conflicts metric can be viewed in CloudWatch
  - Insufficient provisioned capacity.
  - User error, such as an invalid data format.
  - Total size of the items in the transaction exceeds 4 MB.

Batch Operations Vs. Transactions

- Entire TransactWriteItems request fails if:
  - A conflicting update operation on item.
    - Transaction Conflicts metric can be viewed in CloudWatch
  - Insufficient provisioned capacity.
  - User error, such as an invalid data format.
  - Total size of the items in the transaction exceeds 4 MB.
  - Even if one of the condition expressions is not met.
  - Etc.

Isolation level

- Serializable
    - Operations occur as they are executed in a sequence. No operation begins until the previous one has finished
- Read-committed
    - Query or Scan will always be returning data that is in committed state

Cost

- Costs twice as much as a non-transactional read
    - One for preparing
    - One for committing

Examples

- Social media ‘favorites’/’likes’
- Maintaining uniqueness on multiple attributes
- Ecommerce product inventory

Best Practices

- Follow the usual rules of good NoSQL design
    - Design for scale
    - Avoid hot keys
    - Avoid contention on writes (but deal with conflict exceptions)
- Avoid multi-table transactions
    - E.g.: transactions across microservices
    - Design smell? Relational modeling?
- Limit the transaction scope
    - Simpler transactions scale better, execute faster, and so are more likely to succeed

aws

Thank you!