DynamoDB Design for Skier Ride Data Processing

https://github.com/linmengl28/CS6650-Assignment3

Database Design Overview

After evaluating several database options (Redis, MySQL/RDS, DynamoDB, MongoDB), we selected **DynamoDB** for the following reasons:

- **Write Throughput**: Superior performance for high-volume write operations (crucial for our message processing)
- **Scalability**: Ability to handle massive throughput without complex sharding strategies
- **GSI Support**: Built-in indexing for efficient query access patterns without write locks
- **Managed Service**: No operational overhead for scaling, replication, or failover

  Integration: Native AWS integration with our EC2-based consumer architecture

Table Structure

Primary Table: SkierRides

- Partition Key: skierId (Number)
- Sort Key: sortKey (String) – format: "dayId#liftId#timestamp"
- Attributes: resortId, dayId, liftId, time, vertical

Global Secondary Indexes (GSIs)

resort-day-index:

- Partition Key: resortId
- Sort Key: dayId
- Projected Attributes: skierId
- Purpose: Count unique skiers per resort/day

skier-day-index:

- Partition Key: skierId
- Sort Key: dayId
- Projected Attributes: vertical, liftId
- Purpose: Analyze skier activity by day

Deployment Topology

Compute Resources

- **Consumer Application**: Java application running on EC2
  - Instance Type: t3.medium
  - Role: Consumes messages from RabbitMQ, processes data, writes to DynamoDB
  - Concurrency: 512 threads, 250 prefetch count per thread

Database Provisioning

- **DynamoDB Table**: Provisioned capacity mode
  - Base Table: 10 RCU, 2000 WCU
  - GSIs: 20 RCU, 2000 WCU each
  - Region: US-West-2 (Oregon)

Message Queue

- **RabbitMQ**: Deployed on EC2
  - Instance Type: t2.micro
  - Configuration: Persistent queues, durable messages

Performance Optimization

- **Batch Writing**: Implemented custom batching with 25 items per batch
- **Write Efficiency**: Combined flush interval of 100ms with maximum batch size
- **GSI Scaling**: Matched GSI write capacity to base table (2000 WCU)
- **HTTP Client Tuning**: MaxConcurrency=250, ConnectionTimeout=5s

Client performance metrics:

Phase 1 Results:

==========

Thread count: 32

Total requests: 32000

Successful requests: 32000

Failed requests: 0

Total time: 33.43 seconds

Throughput: 957.37 requests/second

==========

Phase 2 Results:

==========

Thread count: 300

Total requests: 168000

Successful requests: 168000

Failed requests: 0

Total time: 25.44 seconds

Throughput: 6602.74 requests/second

==========

Overall Results:

==========

Phase 1: 32 threads with 1000 requests each

Phase 2: 300 threads with remaining requests

Total Requests: 200000

Successful requests: 200000

Failed requests: 0

Total time: 58.87 seconds

Overall Throughput: 3397.37 requests/second

==========

=== Overall Performance Metrics ===

Sample size: 200000 requests

Mean Response Time: 40.89 ms

Median Response Time: 36 ms

Min Response Time: 12 ms

Max Response Time: 470 ms

90th Percentile Response Time (p90): 60 ms

95th Percentile Response Time (p95): 76 ms

99th Percentile Response Time (p99): 112 ms

Throughput: 3397.37 requests/sec

=== Phase 1 Performance Metrics ===

Sample size: 32000 requests

Mean Response Time: 32.47 ms

Median Response Time: 30 ms

Min Response Time: 12 ms

Max Response Time: 469 ms

90th Percentile Response Time (p90): 38 ms

95th Percentile Response Time (p95): 45 ms

99th Percentile Response Time (p99): 86 ms

=== Phase 2 Performance Metrics ===

Sample size: 168000 requests

Mean Response Time: 42.50 ms

Median Response Time: 37 ms

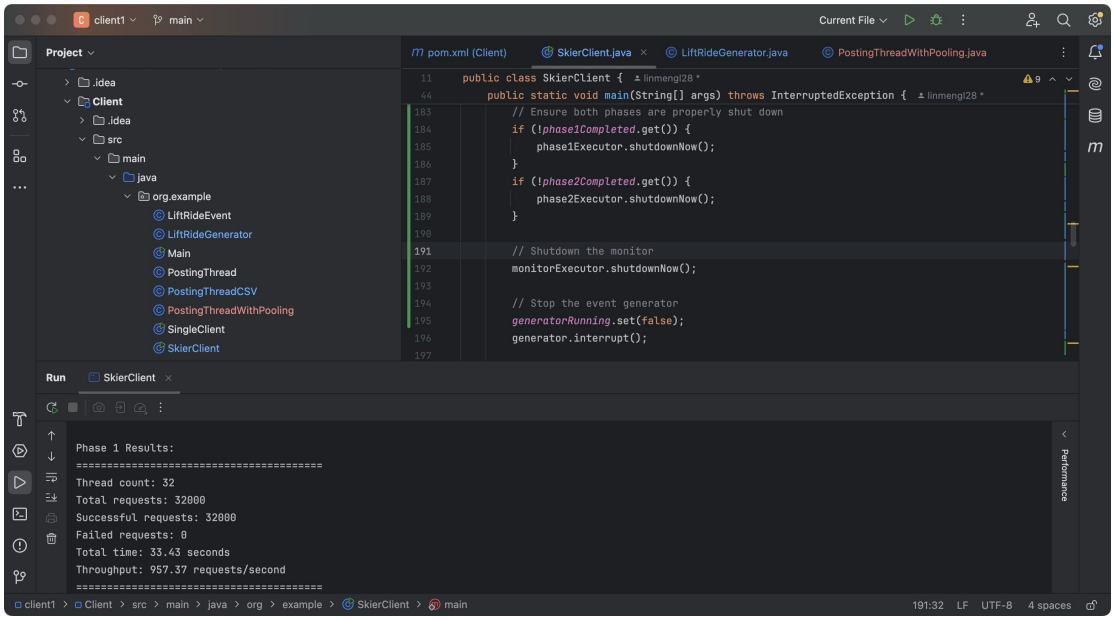
Min Response Time: 14 ms

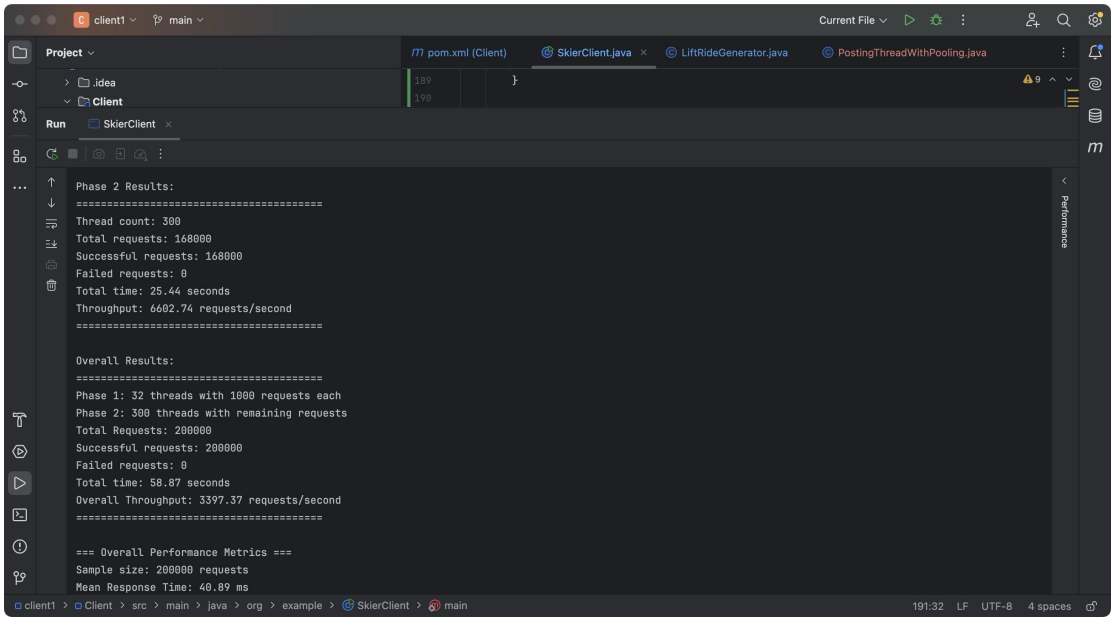
Max Response Time: 470 ms

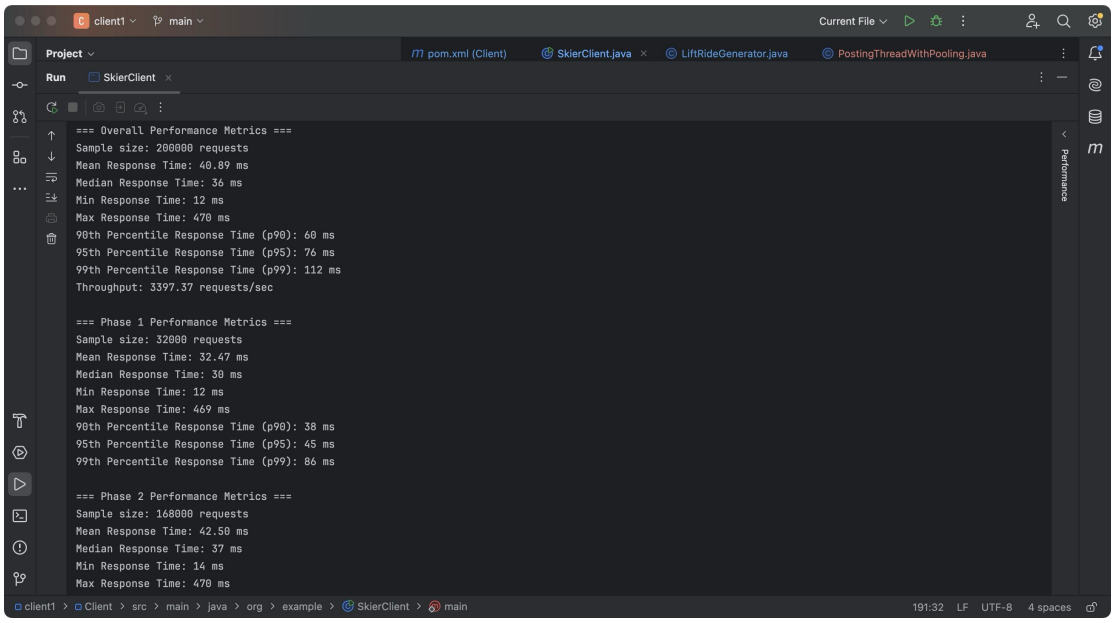
90th Percentile Response Time (p90): 63 ms

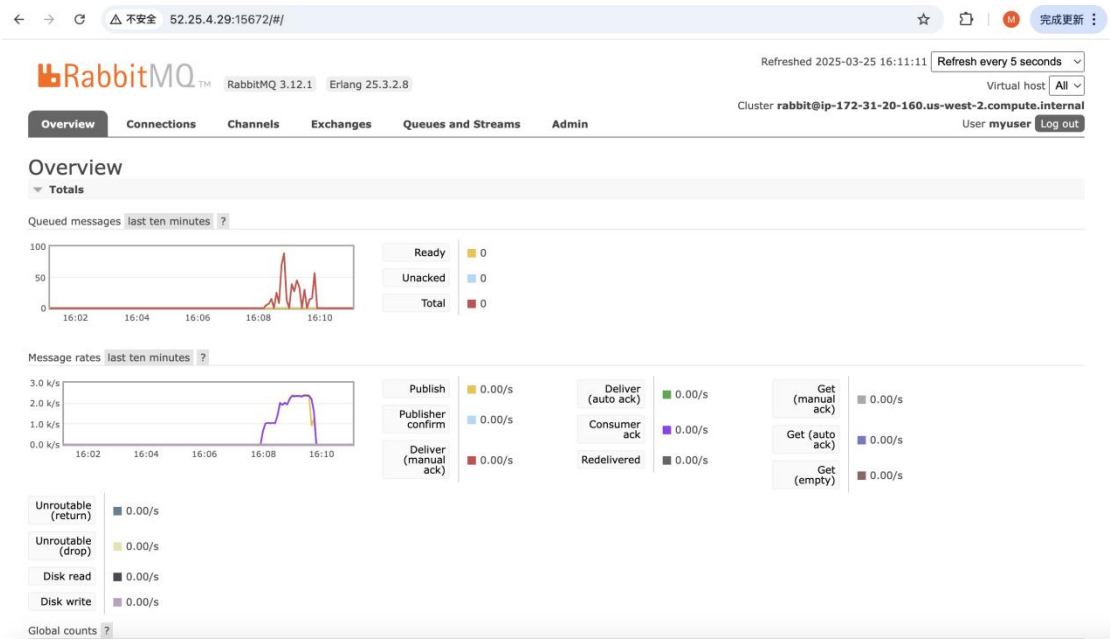
95th Percentile Response Time (p95): 79 ms

99th Percentile Response Time (p99): 114 ms




client1
main
Current File
Project
pom.xml (Client)
SkierClient.java
LiftRideGenerator.java
PostingThreadWithPooling.java
.idea
Client
Run
SkierClient
Phase 2 Results:
==========
Thread count: 300
Total requests: 168000
Successful requests: 168000
Failed requests: 0
Total time: 25.44 seconds
Throughput: 6602.74 requests/second
==========
Overall Results:
==========
Phase 1: 32 threads with 1000 requests each
Phase 2: 300 threads with remaining requests
Total Requests: 200000
Successful requests: 200000
Failed requests: 0
Total time: 58.87 seconds
Overall Throughput: 3397.37 requests/second
==========
=== Overall Performance Metrics ===
Sample size: 200000 requests
Mean Response Time: 40.89 ms
Performance
client1 › Client › src › main › java › org › example › SkierClient › main
191:32 LF UTF-8 4 spaces


client1
main
Current File
Project
pom.xml (Client)
SkierClient.java
LiftRideGenerator.java
PostingThreadWithPooling.java
Run
SkierClient
=== Overall Performance Metrics ===
Sample size: 200000 requests
Mean Response Time: 40.89 ms
Median Response Time: 36 ms
Min Response Time: 12 ms
Max Response Time: 470 ms
90th Percentile Response Time (p90): 60 ms
95th Percentile Response Time (p95): 76 ms
99th Percentile Response Time (p99): 112 ms
Throughput: 3397.37 requests/sec
=== Phase 1 Performance Metrics ===
Sample size: 32000 requests
Mean Response Time: 32.47 ms
Median Response Time: 30 ms
Min Response Time: 12 ms
Max Response Time: 469 ms
90th Percentile Response Time (p90): 38 ms
95th Percentile Response Time (p95): 45 ms
99th Percentile Response Time (p99): 86 ms
=== Phase 2 Performance Metrics ===
Sample size: 168000 requests
Mean Response Time: 42.50 ms
Median Response Time: 37 ms
Min Response Time: 14 ms
Max Response Time: 470 ms
Performance
client1 › Client › src › main › java › org › example › SkierClient › main
191:32 LF UTF-8 4 spaces


不安全 52.25.4.29:15672/#/
完成更新
RabbitMQ
RabbitMQ 3.12.1
Erlang 25.3.2.8
Refreshed 2025-03-25 16:11:11
Refresh every 5 seconds
Virtual host All
Cluster rabbit@ip-172-31-20-160.us-west-2.compute.internal
User myuser
Log out
Overview
Connections
Channels
Exchanges
Queues and Streams
Admin
Overview
Totals
Queued messages last ten minutes ?
100
50
0
16:02
16:04
16:06
16:08
16:10
Ready 0
Unacked 0
Total 0
Message rates last ten minutes ?
3.0 k/s
2.0 k/s
1.0 k/s
0.0 k/s
16:02
16:04
16:06
16:08
16:10
Publish 0.00/s
Publisher confirm 0.00/s
Deliver (manual ack) 0.00/s
Deliver (auto ack) 0.00/s
Consumer ack 0.00/s
Redelivered 0.00/s
Get (manual ack) 0.00/s
Get (auto ack) 0.00/s
Get (empty) 0.00/s
Unroutable (return) 0.00/s
Unroutable (drop) 0.00/s
Disk read 0.00/s
Disk write 0.00/s
Global counts ?