

CS6650 Assignment 2 Report

Shengdi Wang - wang.shengd@northeastern.edu

1. GitHub Repo URL

<https://github.com/leowang396/distributed-music-service>

2. Description of data model

Image used

Used a 67-byte PNG image instead of the stock image. The image is stored in the GitHub repo's client directory for reference.

Database solution

AWS DynamoDB.

Data model

album
album_id
artist
title
year
image

Since we are using the AWS DynamoDB (NoSQL) as the database, we apply the document data model to represent our data. The database models our data as a single album entity that contains the album information and album images posted by the clients. The album information is represented in the String fields "artist", "title", and "year", and the album image is stored in a binary field named "image".

By storing all fields in a single entity, we reduce the number of database write requests needed to fulfill the POST requests, which make up 50% of all client requests. To maintain a primary key for each record, we also generate a UUID for each record, which is stored in the "album_id" field in the database.

3. Output windows for the 3 client configuration tests run against a single server/DB

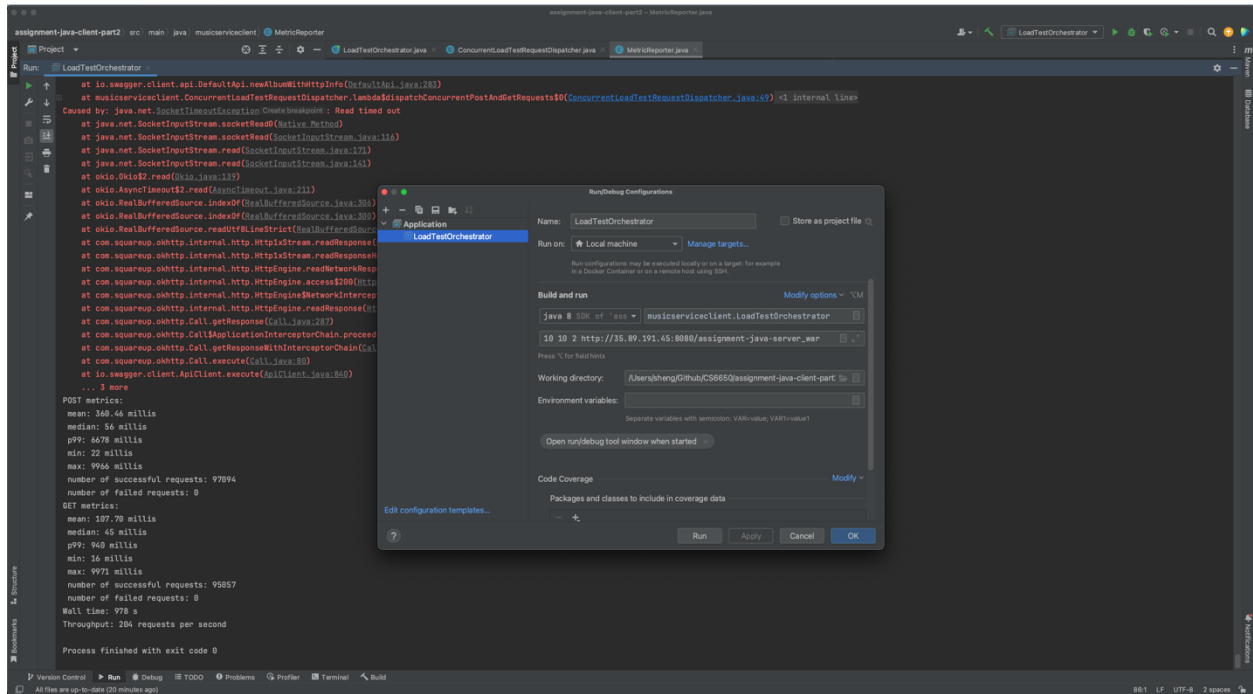


Figure 1: Single Server - 10 Thread Groups

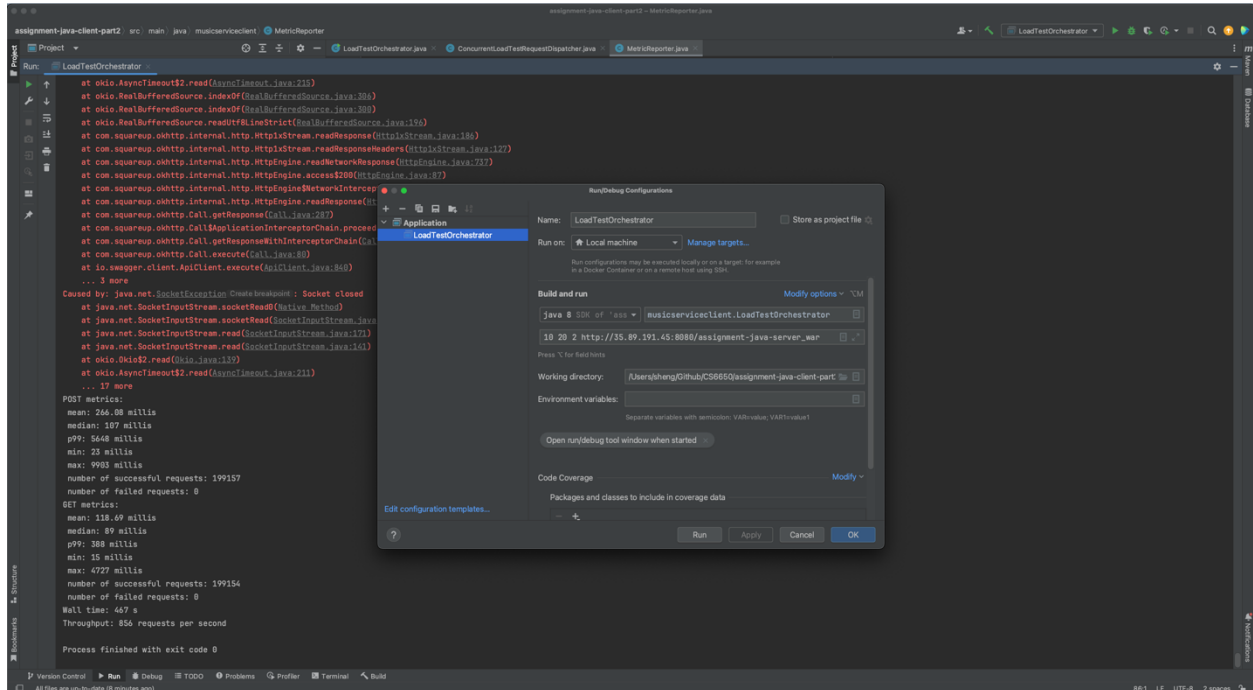


Figure 2: Single Server - 20 Thread Groups

```

assignment-java-client-part2 - MetricReporter.java
Project
Run
LoadTestOrchestrator
at io.swagger.client.api.DefaultApi.newAlbumWithHttpInfo(DefaultApi.java:283)
at musicServiceClient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndRequests$8(ConcurrentLoadTestRequestDispatcher.java:69) <1 internal lines
Caused by: java.net.SocketTimeoutException: Read timed out
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
at java.net.SocketInputStream.read(SocketInputStream.java:171)
at java.net.SocketInputStream.read(SocketInputStream.java:141)
at okio.Okio$2.read(Okio.java:139)
at okio.AsyncTimeout$2.read(AsyncTimeout.java:211)
at okio.RealBufferedSource.indexOf(RealBufferedSource.java:306)
at okio.RealBufferedSource.indexOf(RealBufferedSource.java:288)
at okio.RealBufferedSource.readUtf8LineStrict(RealBufferedSource.java:196)
at com.squareup.okhttp.internal.http.Http1xStream.readResponse(Http1xStream.java:186)
at com.squareup.okhttp.internal.http.Http1xStream.readResponseHeaders(Http1xStream.java:127)
at com.squareup.okhttp.internal.http.HttpEngine.readNetworkResponse(HttpEngine.java:737)
at com.squareup.okhttp.internal.http.HttpEngine.access$280(HttpEngine.java:87)
at com.squareup.okhttp.internal.http.HttpEngine$NetworkInterceptorChain.proceed(HttpEngine.java:722)
at com.squareup.okhttp.internal.http.HttpEngine.readResponse(HttpEngine.java:976)
at com.squareup.okhttp.Call.getResponse(Call.java:287)
at com.squareup.okhttp.Call$ApplicationInterceptorChain.proceed(Call.java:243)
at com.squareup.okhttp.Call.getResponseWithInterceptorChain(Call.java:205)
at com.squareup.okhttp.Call.execute(Call.java:80)
at io.swagger.client.ApiClient.execute(ApiClient.java:848)
... 3 more
POST metrics:
mean: 646.91 millis
median: 182 millis
p99: 9766 millis
min: 23 millis
max: 10011 millis
number of successful requests: 291416
number of failed requests: 0
GET metrics:
mean: 372.16 millis
median: 164 millis
p99: 6367 millis
min: 17 millis
max: 9999 millis
number of successful requests: 289915
number of failed requests: 0
Wall time: 1374 s
Throughput: 436 requests per second
Process finished with exit code 0

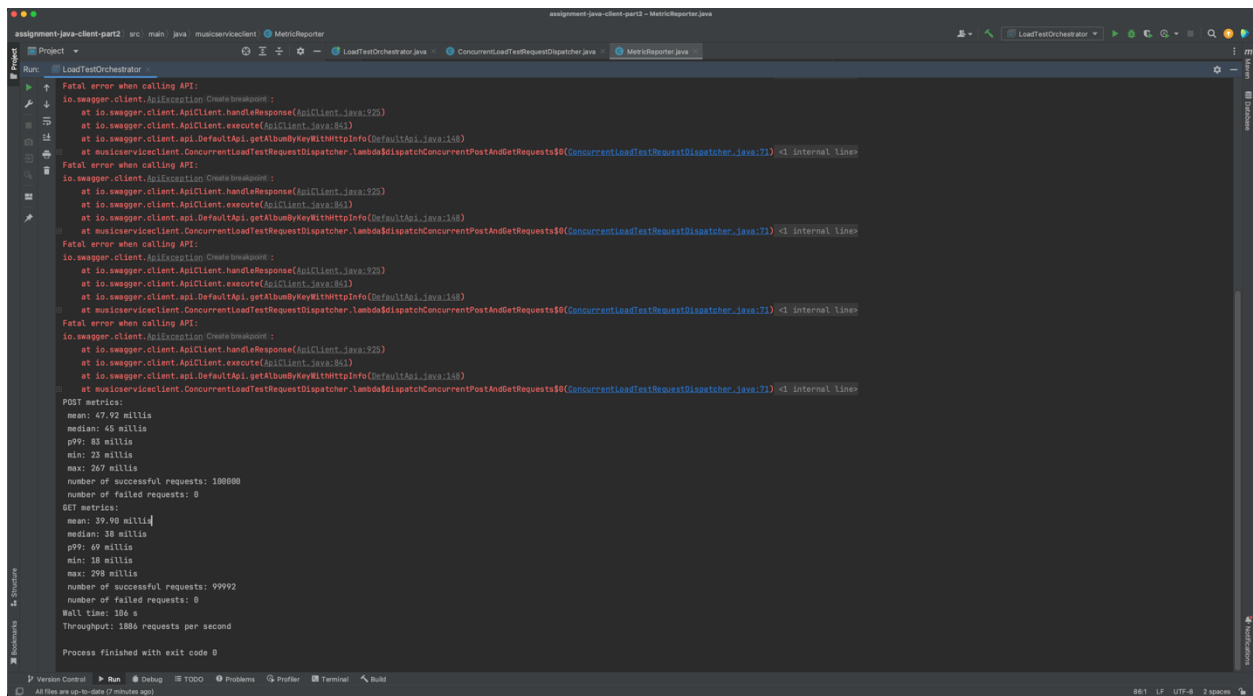
```

Figure 3: Single Server - 30 Thread Groups

Configuration		Wall Time (s)	Throughput
Single-server	threadGroupSize = 10, numThreadGroups = 10, delay =	978	204
	threadGroupSize = 10, numThreadGroups = 20, delay =	467	856
	threadGroupSize = 10, numThreadGroups = 30, delay =	1374	436

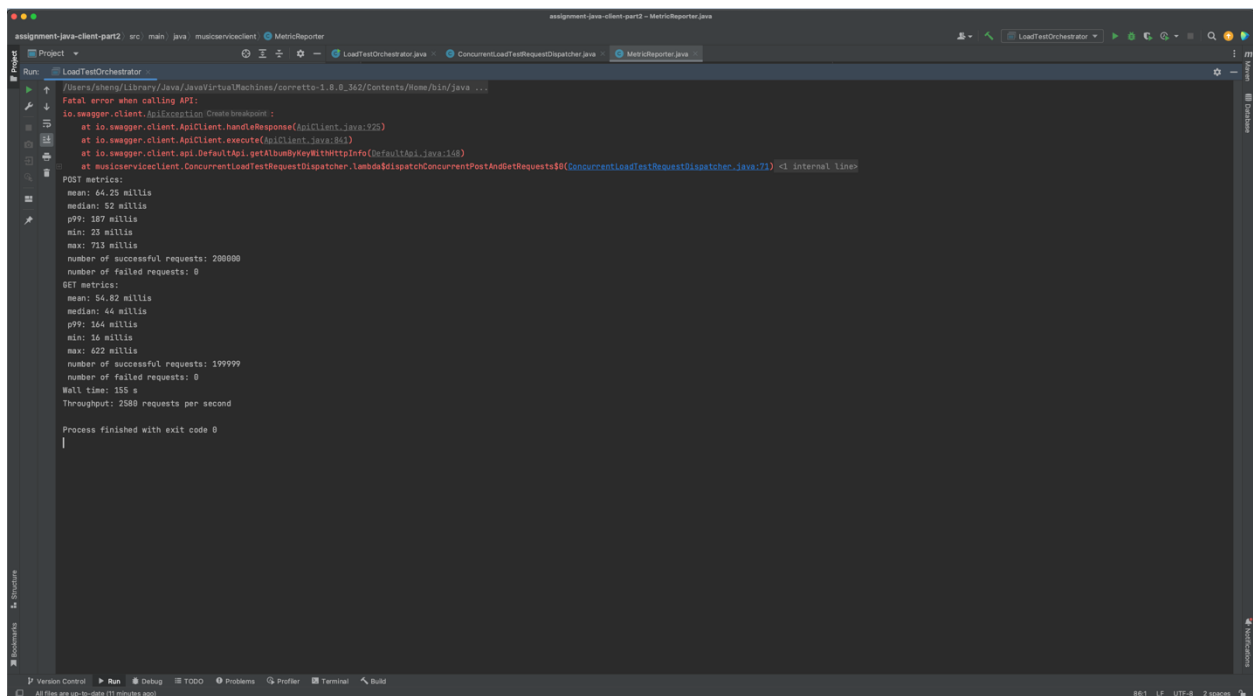
Figure 4: Table of results from Step 3

4. Output windows for the 3 client configuration tests run against two load-balanced servers/DB



```
assignment-java-client-part2 - MetricReporter.java
Run: LoadTestOrchestrator
Fatal error when calling API:
at io.swagger.client.ApiException.createBreakpoint:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$8(ConcurrentLoadTestRequestDispatcher.java:71) <1 Internal Lines
Fatal error when calling API:
at io.swagger.client.ApiException.createBreakpoint:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$8(ConcurrentLoadTestRequestDispatcher.java:71) <1 Internal Lines
Fatal error when calling API:
at io.swagger.client.ApiException.createBreakpoint:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$8(ConcurrentLoadTestRequestDispatcher.java:71) <1 Internal Lines
POST metrics:
mean: 47.92 millis
median: 45 millis
p99: 83 millis
min: 23 millis
max: 267 millis
number of successful requests: 100000
number of failed requests: 0
GET metrics:
mean: 39.90 millis
median: 38 millis
p99: 69 millis
min: 18 millis
max: 290 millis
number of successful requests: 99992
number of failed requests: 0
Wall time: 186 s
Throughput: 1886 requests per second
Process finished with exit code 0
```

Figure 5: Two Load-balanced Servers - 10 Thread Groups



```
assignment-java-client-part2 - MetricReporter.java
Run: LoadTestOrchestrator
Fatal error when calling API:
at io.swagger.client.ApiException.createBreakpoint:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$8(ConcurrentLoadTestRequestDispatcher.java:71) <1 Internal Lines
POST metrics:
mean: 64.25 millis
median: 52 millis
p99: 187 millis
min: 23 millis
max: 713 millis
number of successful requests: 200000
number of failed requests: 0
GET metrics:
mean: 54.82 millis
median: 44 millis
p99: 164 millis
min: 16 millis
max: 827 millis
number of successful requests: 199999
number of failed requests: 0
Wall time: 155 s
Throughput: 2589 requests per second
Process finished with exit code 0
```

Figure 6: Two Load-balanced Servers - 20 Thread Groups

```

assignment-java-client-part2 src: main java musicserviceclient ConcurrentLoadTestRequestDispatcher dispatchConcurrentPostAndGetRequests Lambda LoadTestOrchestrator LoadTestReporter.java
Run LoadTestOrchestrator
Fatal error when calling API:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.newAlbumWithHttpInfo(DefaultApi.java:283)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$0(ConcurrentLoadTestRequestDispatcher.java:49) <1 internal lines>
Fatal error when calling API:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$0(ConcurrentLoadTestRequestDispatcher.java:71) <1 internal lines>
Fatal error when calling API:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$0(ConcurrentLoadTestRequestDispatcher.java:71) <1 internal lines>
Fatal error when calling API:
at io.swagger.client.ApiClient.handleResponse(ApiClient.java:925)
at io.swagger.client.ApiClient.execute(ApiClient.java:841)
at io.swagger.client.api.DefaultApi.getAlbumByKeyWithHttpInfo(DefaultApi.java:148)
at musicserviceclient.ConcurrentLoadTestRequestDispatcher.lambda$dispatchConcurrentPostAndGetRequests$0(ConcurrentLoadTestRequestDispatcher.java:71) <1 internal lines>
POST metrics:
mean: 103.88 millis
median: 57 millis
p99: 618 millis
min: 22 millis
max: 9979 millis
number of successful requests: 294554
number of failed requests: 0
GET metrics:
mean: 94.11 millis
median: 49 millis
p99: 626 millis
min: 17 millis
max: 9977 millis
number of successful requests: 294550
number of failed requests: 0
Wall time: 255 s
Throughput: 2352 requests per second
Process finished with exit code 0
  
```

Figure 7: Two Load-balanced Servers - 30 Thread Group

Configuration		Wall Time (s)	Throughput
2-load-balanced-servers	threadGroupSize = 10, numThreadGroups = 10, delay =	106	1886
	threadGroupSize = 10, numThreadGroups = 20, delay =	155	2580
	threadGroupSize = 10, numThreadGroups = 30, delay =	255	2352

Figure 8: Tables of Results from Step 4

5. Output window for optimized server configuration for the client with 30 Thread Groups. Briefly describe what configuration changes you made and what % throughput improvement you achieved

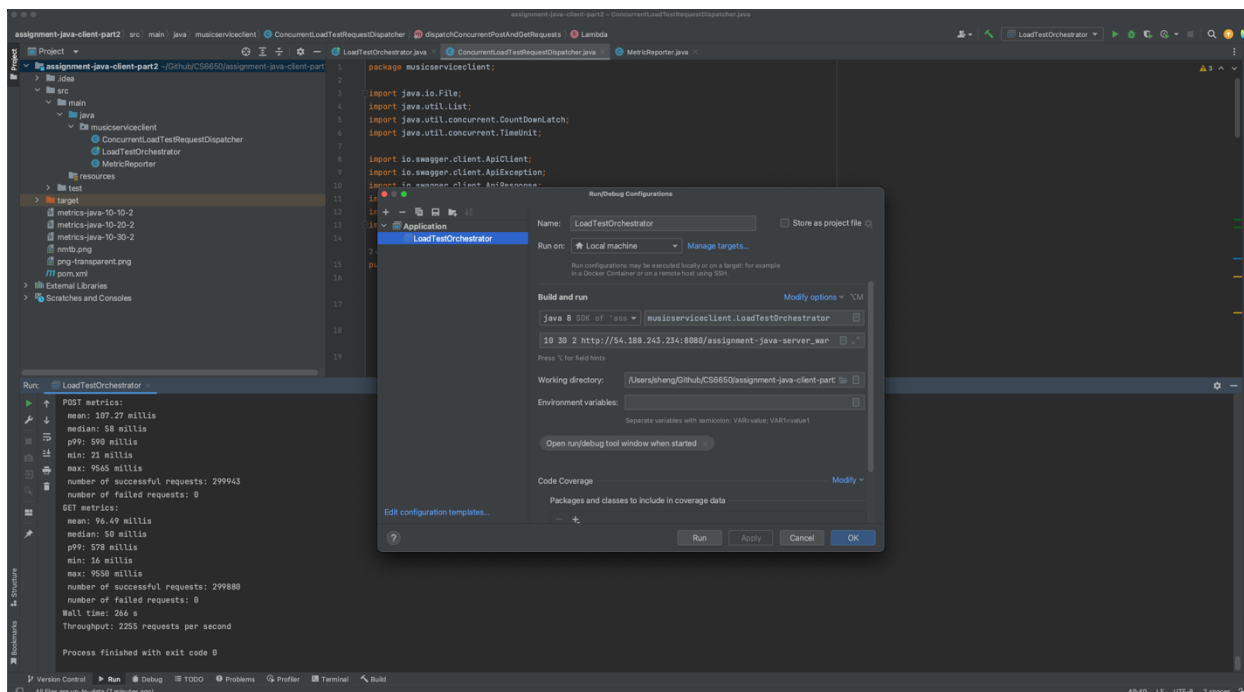


Figure 9: Optimization Configuration - 30 Thread Groups

Configuration		Wall Time (s)	Throughput
Optimized	threadGroupSize = 10, numThreadGroups = 10, delay =	94	2137
	threadGroupSize = 10, numThreadGroups = 20, delay =	155	2580
	threadGroupSize = 10, numThreadGroups = 30, delay =	266	2255

Figure 10: Table of Results from Step 5

Multiple iterations of changes were made. A few of these optimizations were done as a part of Steps 3 and 4, and the rest were done after ALB was added.

- I started off using **MySQL/AWS RDS and Java DBCP connection pool** as the database technology. However, despite achieving 2000+ throughput on a local MySQL DB, adding AWS RDS caused the server throughput to drop to **~450** on EC2.
- After changing the **DB instance type** from db.t2.micro to db.t4g.micro, and swapping the stock image with the smaller **67-byte PNG image**, I was able to achieve a throughput of **600** with MySQL, a significant **33% improvement**.
- Since this is still short of the 1000 throughput target, I decided to try out DynamoDB. **DynamoDB** came with a default read/write capacity of 5 capacity units each, which proved to be way too small for even just 10 thread groups. To save time, I enabled auto-scaling and increased the minimum capacity for read/write from 1 to 100 capacity units. This led to a throughput that fluctuated between **200 to 900**. The wild fluctuation in throughput is due to incremental and often time-consuming responses with

DynamoDB's auto-scaling feature. If we take the highest throughput for comparison, this is a **50% improvement**.

- At this point, I observed the servlet server is getting the exception "java.io.IOException: Too many open file", which indicates that a single server is getting too busy and running out of file descriptors. I then proceeded to add a second server by using the **Application Load Balancer**. This led to a **100% improvement** in throughput from 900 to **1800** for the 10 Thread Groups experiment.
- To address the fluctuations in throughput and frequent reset of read/write capacity by DynamoDB's auto-scaling feature, I raised DynamoDB auto-scaling's **minimum read/write capacity to 2000** before I started the final experiments. This is the observed peak in the read/write capacity when fully managed by the auto-scaling feature. By doing so, I was able to cut away a few minutes of incremental ramp-up by DynamoDB's auto-scaling and achieve a consistent throughput of **over 2000**.

Configuration		Wall Time (s)	Throughput
Single-server	threadGroupSize = 10, numThreadGroups = 10, delay =	978	204
	threadGroupSize = 10, numThreadGroups = 20, delay =	467	856
	threadGroupSize = 10, numThreadGroups = 30, delay =	1374	436
2-load-balanced-servers	threadGroupSize = 10, numThreadGroups = 10, delay =	106	1886
	threadGroupSize = 10, numThreadGroups = 20, delay =	155	2580
	threadGroupSize = 10, numThreadGroups = 30, delay =	255	2352
Optimized	threadGroupSize = 10, numThreadGroups = 10, delay =	94	2137
	threadGroupSize = 10, numThreadGroups = 20, delay =	155	2580
	threadGroupSize = 10, numThreadGroups = 30, delay =	266	2255

Figure 11: Comparison of results across steps 3, 4, 5

6. Screenshots of the ALB setup

EC2 > Load balancers > cs6650-assignment2-alb

cs6650-assignment2-alb

Details

Load balancer type Application	Status Active	VPC vpc-014e0740d3beac6c4	IP address type IPv4
Scheme Internet-facing	Hosted zone Z1H1FLSHABSF5	Availability Zones subnet-00a48a70e82a4344e us-west-2a (usw2-az2) subnet-0319d4e8472ca0a6b us-west-2b (usw2-az1)	Date created October 30, 2023, 20:55 (UTC-07:00)
Load balancer ARN arn:aws:elasticloadbalancing:us-west-2:536170621232:loadbalancer/app/cs6650-assignment2-alb/78674897b2d7c1ee		DNS name cs6650-assignment2-alb-2132574844.us-west-2.elb.amazonaws.com (A Record)	

Listeners and rules | Network mapping | Security | Monitoring | Integrations | Attributes | Tags

Listeners and rules (1) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

Filter listeners

Protocol/Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	Tags
HTTP:8080	Forward to target group • cs6650-assignment2-alb 1 (100%) • Group-level stickiness: Off	1 rule	ARN	Not applicable	Not applicable	0 tags

Figure 12: ALB Listener Setup

EC2 > Load balancers > cs6650-assignment2-alb

cs6650-assignment2-alb

Details

Load balancer type Application	Status Active	VPC vpc-014e0740d3beac6c4	IP address type IPv4
Scheme Internet-facing	Hosted zone Z1H1FLSHABSF5	Availability Zones subnet-00a48a70e82a4344e us-west-2a (usw2-az2) subnet-0319d4e8472ca0a6b us-west-2b (usw2-az1)	Date created October 30, 2023, 20:55 (UTC-07:00)
Load balancer ARN arn:aws:elasticloadbalancing:us-west-2:536170621232:loadbalancer/app/cs6650-assignment2-alb/78674897b2d7c1ee		DNS name cs6650-assignment2-alb-2132574844.us-west-2.elb.amazonaws.com (A Record)	

Listeners and rules | Network mapping | Security | Monitoring | Integrations | Attributes | Tags

Network mapping Info

Targets in the listed zones and subnets are available for traffic from the load balancer using the IP addresses shown.

Edit IP address type | Edit subnets

VPC vpc-014e0740d3beac6c4	IP address type IPv4
IPv4: 172.31.0.0/16	
IPv6: -	

Mappings

Including two or more Availability Zones, and corresponding subnets, increases the fault tolerance of your applications.

Zone	Subnet	IPv4 address	Private IPv4 address	IPv6 address
us-west-2a (usw2-az2)	subnet-00a48a70e82a4344e	Assigned by AWS	Assigned from CIDR 172.31.32.0/20	Not applicable
us-west-2b (usw2-az1)	subnet-0319d4e8472ca0a6b	Assigned by AWS	Assigned from CIDR 172.31.16.0/20	Not applicable

Figure 13: ALB Network Mapping Setup

EC2 > Target groups > cs6650-assignment2-alb

cs6650-assignment2-alb

Actions

Details

arn:aws:elasticloadbalancing:us-west-2:536170621232:targetgroup/cs6650-assignment2-alb/d2d2ed3695143a88

Target type

Instance

Protocol : Port

HTTP: 8080

Protocol version

HTTP1

VPC

[vpc-014e0740d3beac6c4](#)

IP address type

IPv4

Load balancer

[cs6650-assignment2-alb](#)

Total targets

2

Healthy

2

Unhealthy

0

Unused

0

Initial

0

Draining

0

► Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets

Monitoring

Health checks

Attributes

Tags

Registered targets (2)

Filter targets

< 1 >

Instance ID

Name

Port

Zone

Health status

Health status details

[i-05d7e40d0b0506892](#)

Web Service Gin

8080

us-west-2b

Healthy

[i-0077c191bca40b8ae](#)

Web Service Backup

8080

us-west-2b

Healthy

Figure 14: ALB Target Setup

EC2 > Target groups > cs6650-assignment2-alb

cs6650-assignment2-alb

Actions

Details

arn:aws:elasticloadbalancing:us-west-2:536170621232:targetgroup/cs6650-assignment2-alb/d2d2ed3695143a88

Target type

Instance

Protocol : Port

HTTP: 8080

Protocol version

HTTP1

VPC

[vpc-014e0740d3beac6c4](#)

IP address type

IPv4

Load balancer

[cs6650-assignment2-alb](#)

Total targets

2

Healthy

2

Unhealthy

0

Unused

0

Initial

0

Draining

0

► Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets

Monitoring

Health checks

Attributes

Tags

Health check settings

Edit

Protocol

HTTP

Path

/assignment-java-server_war

Port

Traffic port

Healthy threshold

5 consecutive health check successes

Unhealthy threshold

2 consecutive health check failures

Timeout

5 seconds

Interval

30 seconds

Success codes

200-399

Figure 4: ALB Target Group Health Check Setup

7. Screenshots of the database after testing

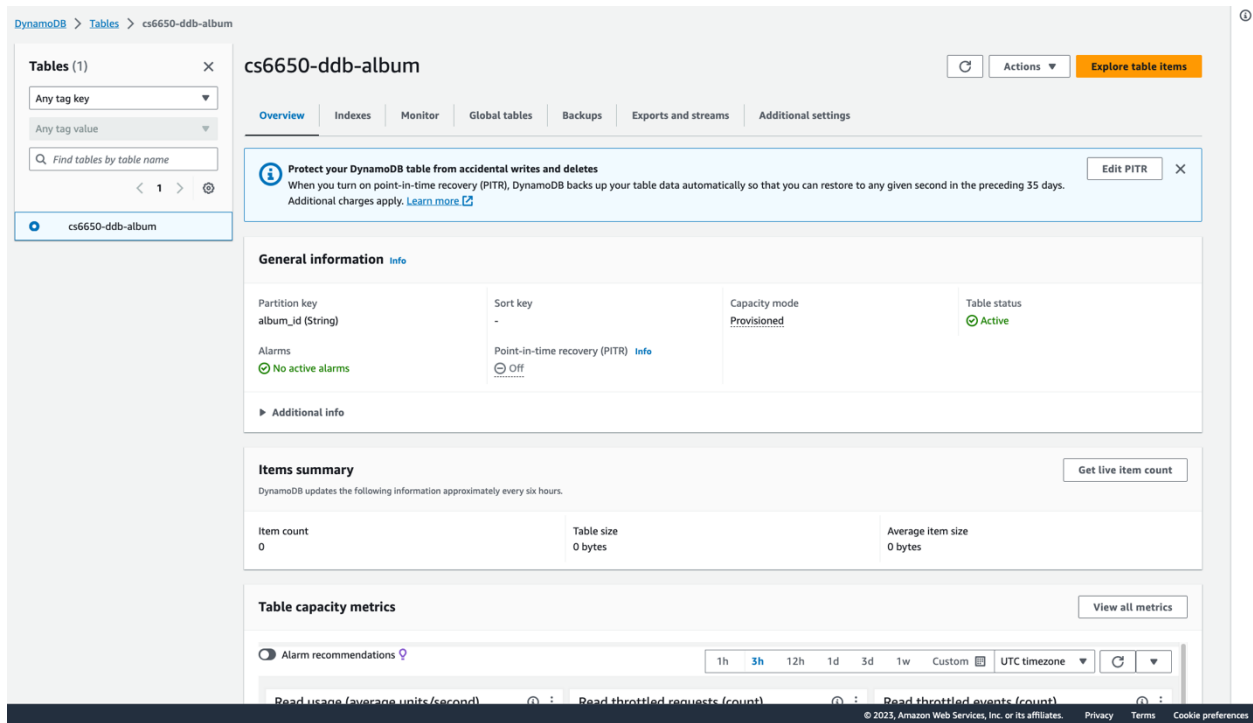


Figure 15: DynamoDB Overview of General Information

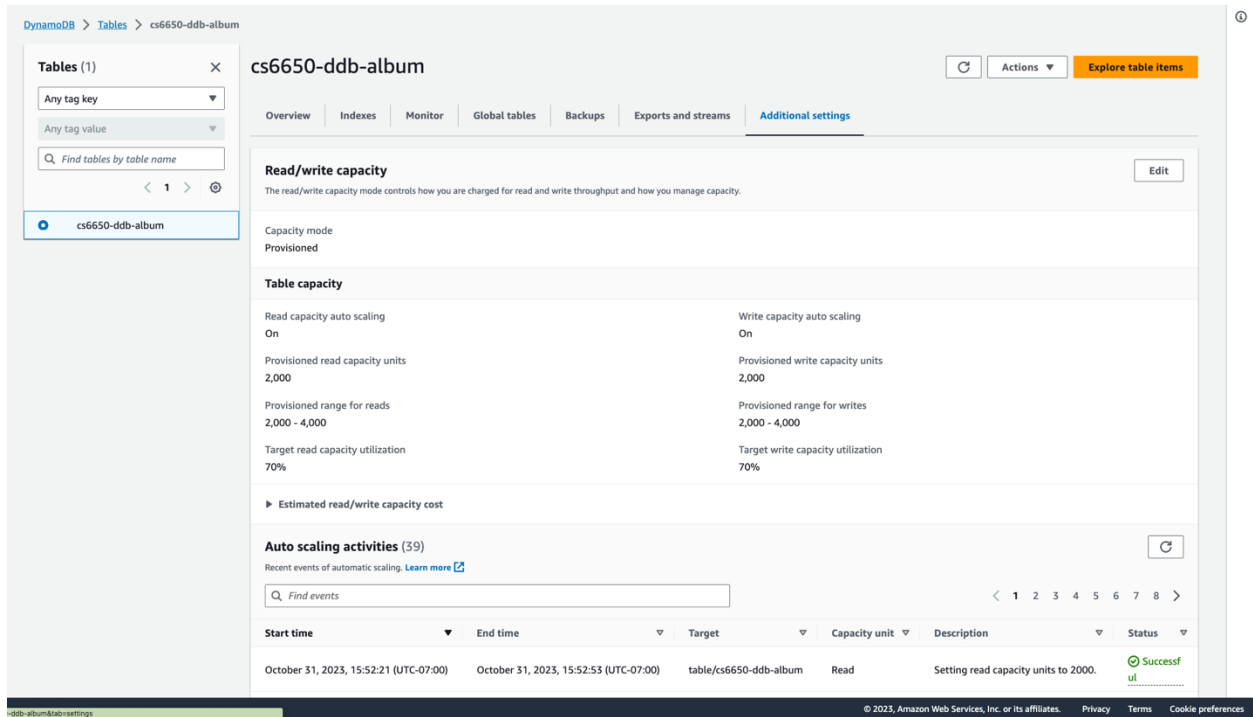


Figure 16: DynamoDB Additional Settings including Auto Scaling Settings

DynamoDB > Explore items > cs6650-ddb-album

Tables (1)

Any tag key

Any tag value

Find tables by table name

< 1 > @

cs6650-ddb-album

cs6650-ddb-album

AutopreviewView table details

Scan or query items

Expand to query or scan items.

This table has more items to retrieve. To retrieve the next page of items, choose Retrieve next page.

Retrieve next page

Items returned (50)

ActionsCreate item

< 1 > @

	album_id (String)	artist	image	title	year
<input type="checkbox"/>	2d0e81af-fe90-46cf-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	20346e4f-9b02-40e9-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	e7a7462c-bfcd-4d7f-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	91562698-003c-4c1a-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	12e7b0d9-54db-4a66-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	27283f6f-b3f6-486a-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	109e3ae8-7302-4c12-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	e7524e14-4b25-477e-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	1d350ee7-e02c-495f-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	fab2afc2-0c57-4118-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	50acb5b8-dfdd-4f16-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	3550e753-45ff-4b2c-...	new-artist	IVBORwOK...	new-title	2000
<input type="checkbox"/>	3a5d8705-b2ae-4c91-...	new-artist	IVBORwOK...	new-title	2000

Showing selection

© 2023, Amazon Web Services, Inc. or its affiliates. PrivacyTermsCookie preferences

Figure 17: DynamoDB Samples of Items

8. Any other screenshot to demonstrate the effect of your configuration change for task 5.

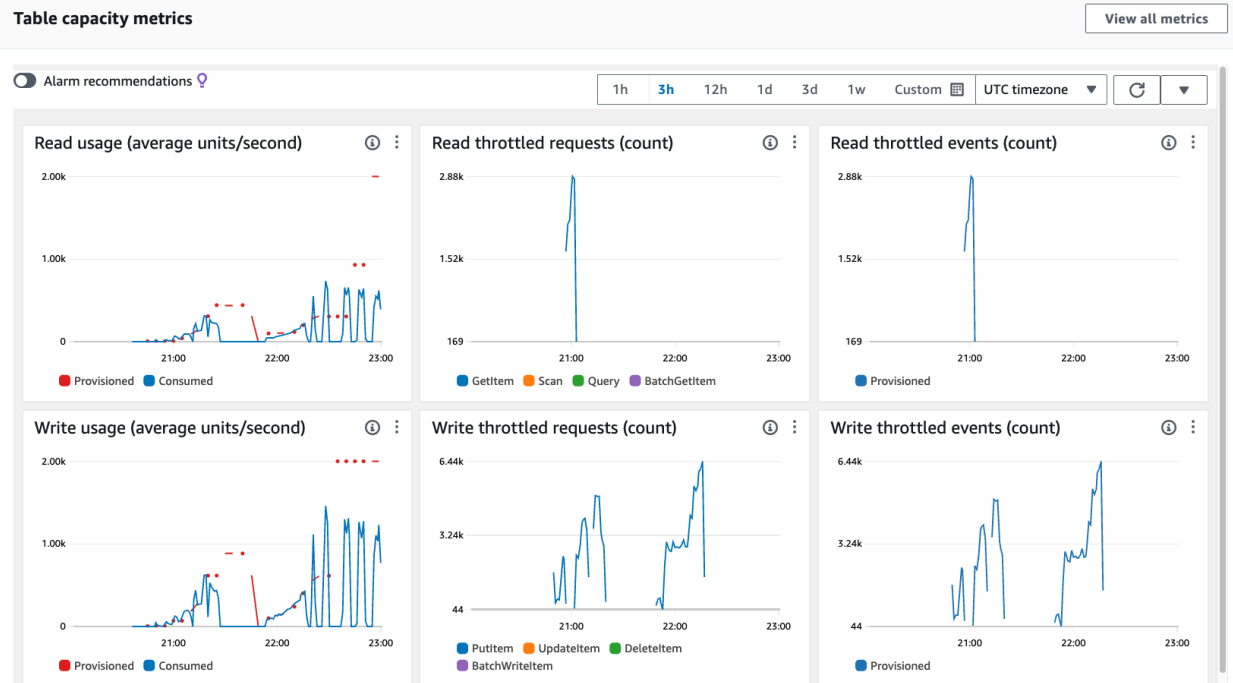


Figure 18: DynamoDB Basic Monitoring Graphs

As we can observe, before manually setting the minimum read/write capacity to a high enough value (i.e. 2000), auto-scaling took a number of minutes to effect one iteration of capacity increase. The capacity increases are also incremental in magnitude, which requires multiple iterations to reach a suitable capacity. As a result of this relatively slow increase in capacity, DynamoDB's consumed read/write capacity is frequently limited by the provisioned capacity. We also observed a significant amount of throttled read and write requests.

In task 5 (at around time 23:00 in the graphs), I manually set the minimum read/write capacity to 2000, which is sufficiently large for our load tests. Read/write request throttling immediately dropped to 0. We therefore observed a consistent throughput above 2000 in our load tests.