**Assignment 3 – CS6650**

Yuchen Tian

1. URL for Github Repo: <https://github.com/tyuchn/cs6650-hw3-ytian>
2. Database Design and deployment topology
   1. Database Design

CREATE SCHEMA SkierApplication;

use SkierApplication;

create table LiftRides(

id INT NOT NULL AUTO\_INCREMENT,

skierId INT NOT NULL,

dayId INT NOT NULL,

time INT NOT NULL,

liftId INT NOT NULL,

PRIMARY KEY ( id )

);

create table Resorts(

id INT NOT NULL AUTO\_INCREMENT,

dayId INT NOT NULL,

year INT NOT NULL,

time INT NOT NULL,

resortId INT NOT NULL,

skierId INT NOT NULL,

liftId INT NOT NULL,

PRIMARY KEY ( id ),

);

LiftRides table is used to store the skier and lift ride information for specific season. To consume message from message queue ASAP, each request will be stored as a single entry in the DB. As a result, the LiftRides table has the fields of skier id, id, time, lift id, the primary key is automatically assigned by MySQL. To enable the queries like “For skier N, XXX”, we need to group the rows by skier Id while executing SQL queries.

Vice versa, Resort table is used to store data pertinent to the ski resort. It has the fields of dayId, year, time, resortId, skierId and liftId. The primary key is also automatically incremented for each record. To enable queries like “For day N, XXX”, we can select on the specific day and do some “group by” operations.

* 1. Deployment topology

I created an MySQL instance on RDS, the attribute is db.t2.micro with 1 vCPU and 1 GB RAM at the beginning. The JDBC connection was established on consumer microservices.

1. Test Results – original setup

At the first step, both the consumer and db instance are t2.micro, the running result showed as below

* 1. Step1 – 128 threads
     1. Commands lines:

*-numThreads 128 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step1 – 256 threads
     1. Commands lines:

*-numThreads 256 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step2 – 128 threads
     1. Commands lines:

*-numThreads 128 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step2 – 256 threads
     1. Commands lines:

*-numThreads 256 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

1. Test Results – after increasing capacity

At this step, both the consumer and db instance are increased from t2.micro to t2.large

* 1. Step1 – 128 threads
     1. Commands lines:

*-numThreads 128 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step1 – 256 threads
     1. Commands lines:

*-numThreads 256 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step2 – 128 threads
     1. Commands lines:

*-numThreads 128 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

* 1. Step2 – 256 threads
     1. Commands lines:

*-numThreads 256 -numSkiers 100000 -address 54.82.5.13:8080*

* + 1. RMQ management console:

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + 1. Result:

Text

Description automatically generated

1. Result Analysis
   1. Comparisons:

|  |  |
| --- | --- |
|  | Max Queue Length (backlogs) |
| Step1 – 128 threads (before) | 24528 |
| Step1 – 128 threads (after) | 72 |
| Step 1 – 256 threads (before) | 45548 |
| Step 1 – 256 threads (after) | 6922 |
| Step 2 – 128 threads (before) | 28584 |
| Step 2 – 128 threads (after) | 100 |
| Step 2 – 256 threads (before) | 45095 |
| Step 2 – 256 threads (after) | 4886 |

* 1. Analysis: At the beginning, for both 128 threads and 256 threads in Step 1&2, we can see significant (tens of thousand) backlog in the message queue, which was caused by the DB bottleneck with no double. In order to eliminate it, I chose to increase the capacity of both the DB and consumer. After increase the size from t2.micro to t2.large, we can see the result for 128 thread, peek of the backlog is less than 100, while for 256 threads, peek of backlog is less than 7k. Because we enhanced the capacity of our RAM and add more memory to the instance, we have higher I/O speed. As a result, the decrement of backlogs is significant, the bottleneck was mitigated significantly. So, we can say our solution is successful.