CS5010 Fall 2017

Assignment 8 - More Concurrency. Lots More ...

Overview

The aim of this assignment is to give you more experience designing and implementing a concurrent programs to process a reasonably large amount of data.

The Problem Scenario

Your success in impressing the CTO of Blackler-Whistcomb with your work in Assignment 7 leads to you soon becoming employed as a consultant.

Your first task in this new role is to explore how you can start to persist the data that is produced by ski lift data analysis and start to make this available to potentially a large amount of skiers concurrently through a web site or mobile app.

In this assignment you will design and implement a simple data model for your analyzed and raw data, persist the data, and explore mechanisms to provide rapid response times to queries..

We will use the same data as assignment 7. You can download the data set from here.

Step 1 - Store the Data

Assignment 7 produced results stored in csv files. We're going to build on those, and add to them of course (as it would be far too easy if we didn't!).

You will need to persist the following data:
Lift Data: The raw ski lift data you get from the csv file (liftrides.dat)
For all 40K skiers: the number of lift rides they do each day, and the total amount of vertical metres they ski (equal to the total vertical rise of all lifts they ride) (skier.dat)
For each of the 40 lifts: the total number of lift rides in this day (lifts.dat)
For each hour in the ski day: the top 10 most popular lifts, ordered from the lift with the most rides to the lift with the least. (hours.dat)

You will need to design these files so they can be randomly accessed. For example, if i want to see the data for skier #27894, I should be able to read this from disk with the same access time as skier #1.

Design and implement a set of classes that can create and access these files. The specific queries you will need to process are described in the next step. Take these into account when you are describing your data model. We would suggest you modify one of your submissions from Assignment 7 to write these files.

In addition, you will be required to store the number of times each skier views their summary data. When you create the skier data set, initialize this value to 0 for every skier. Whenever you get a **Skier Summary Query** (see below), add one to this field and return the new value as part of the results. This is the only update (write) you need to satisfy. All other queries are read only. Remind you of the readers-writers problem? We hope so.

As you only have data for one ski day (oddly, Day 2), you can ignore the day completely.

Step 2 - Queries

Once the data has been stored in files and indexed, create a program that will query this data. You will be supplied with test data that contains sample queries in the format:

<queryID, parameters>

The general query formats are as follows:

Skier Summary Query: <queryID=1, parameter = skierID>
Returns: <skierID, numRides, totalVertcal, numberOfViews>

Skier Details Query: <queryID=2, parameter = skierID>
Returns a list of the lift rides for that skier ordered by time, namely <time, LiftID>

Avoid Lift Query: <queryID = 3, parameter = hourID (1-6)>
Returns the top 10 popular lifts ordered by most to least popular for that hour

Lift Query: <queryID = 4, parameter = LiftID>
Returns the total number of rides for the lift, namely <LiftID, numRides>

Your program should

1) Accept 2 command line arguments, namely test data file name, and number of queries. If 20 is not a factor of the the number of queries, throw an exception and terminate.
2) Read the query data supplied as a csv file into a suitable collection.
3) Create 20 threads and give each thread a disjoint segment of the queries to process (the test data conveniently has 40000 queries).
4) Start all the threads at the same time, with each thread processing its allocated queries sequentially.
5) Synchronize the termination of all the threads

6) After all threads are complete, write to the console the elapsed time it took for all threads from start to completion.

Results

Each thread should write the results of its queries to a text file named threadN.txt. Write the query results to a single line in the output file.

What To Submit?

When submitting your assignment, you should continue using the same Maven archetype that we used in previous assignments.

You will want to submit the following:

1. Class SkiQueryProcessor. This class will be assumed to be the starting point of your program, and will be called from the command line.
2. The 20 text files generated by your program when run against the input test file.
3. A ‘clean’ version of your skier.dat file, with all view fields for every skier set to zero.
4. All classes that you have developed for this assignment.
5. All abstract classes and interfaces that you extended and implemented in this assignment.
6. All classes that you have developed to test your code.
7. A UML diagram, corresponding to the design of your program.
8. A brief write-up, which summarizes the main relations between your classes, and how does your program handle errors and/or exceptions.

Deadline Monday 27th November 6pm PST