CS5010 Fall 2017

Assignment 7 - Basic Concurrency

Overview

The aim of this assignment is to give you experience designing and implementing a concurrent programs to process a reasonably large amount of data. We'll start by implementing a serial version of the problem, and then redesign to create a highly concurrent solution.

The Problem Scenario

You are an avid skier, who, unlike most nerds, loves to talk to strangers on chair lift rides up the mountain. One day, you are riding a lift at Blackler-Whistcomb, your favorite place to ski, when the lift breaks down and stops for 30 minutes. By sheer coincidence, your chair lift partner turns out to be the CTO of the resort. She tells you all about her job, and the challenges they have faced implementing the software systems to capture data from new RFID ticket readers at all lifts.

Using this new system, when a skier (snowboarders are evil and hence banned) wants to get on a ski lift they place their RFID-enabled lift pass next to an RFID reader. It checks the pass is valid and then records:

- Resort ID: (numeric, e.g., 0, 1, ...)
- Day Number: (1-365 representing the operational day in the season of the resort, eg opening day is Day 1)
- Timestamp: (0-360 representing number of minutes after lifts open (9am) until they close (3pm)
- Skier ID: (unique, numeric, e.g,….)
- Lift ID: (1-40)

You may assume that all the values above can be stored as Strings.

The resort expects an average of 40K skiers per day, and on average each skier will ride 20 lifts between 9am and 3pm when the lifts close. This creates a large amount of records to capture and store everyday.

At this stage, the CTO explains they are simply capturing the data and have not had the resources to analyze it. As a nerdy skier who loves writing code, you start describing all the wonderful things you could do to get intelligence from the data. By the time the lift start moving again, the CTO has your email and she has promised to send you a data set representing one day of skier data. She also sends you the following information about the ski lifts:

- There are only 40 ski lifts.
- Lifts 1-10 rise 200m vertical.
- Lifts 11-20 are 300m vertical.
- Lifts 21-30 are 400m vertical,
- Lifts 31-40 and 500m vertical.

As you can probably guess, this data doesn't change very often :)

You can download the data set from here.

Step 1 - Sequential Solution

After exploring the data, you decide that the following outputs can be generated from the data set:

For each skier: 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)
For each lift: the total number of lift rides per day
For each hour in a ski day: the order of popularity of each lift, ordered from the lift with the most rides to the lift with the least.

Your task therefore is to read in the data set and generate these outputs. You want to send the CTO some results quickly, so you decide to write a sequential solution, but realize that because this is just a fraction of the data, you'll need to make it run in parallel eventually. Out of curiosity, you decide to capture the total time it takes for your program to completely process the data.

After all the required statistics have been calculated, the program should output the results as explained in the Results section below.

Step 2 - Concurrent Solution

You send your results to the CTO and she loves them, as well as your plans to implement the analysis concurrently so that it can scale as the data grows.

You decide to design your concurrent solution using data processing pipelines to parallelize the solution. These are the major elements of your design:

1) A data source, which reads through the input file and stores data from every record in 3 queues, namely:
    a) A skier queue, which contains a record used to calculate every skier's lift rides and daily vertical. This is a KV pair, namely <skierID, liftID>

   b) A lift queue, which contains a record for every lift ride, namely <liftID>.
   c) An hour queue, which contains a record for every lift ride in each hour, namely <hour number (1-6), liftID>
2) 3 multithreaded data sinks that each read from one of the queues, and process the inputs to calculate the required statistics

After all the required statistics have been calculated, the program should output the results as explained in the Results section below. It should also terminate cleanly and output the total execution time as per step 1.

Results

For both the sequential and concurrent version, the results should be exactly the same. You'll produce 3 .csv files, namely:

Skiers:
skier.csv, containing the skiers with the Top 100 vertical totals in descending order. The header should be:
SkierID, Vertical

Lifts
lifts.csv, with a line for each lift in ascending order of LiftID. The header should be:
LiftID, Number of Rides

Hours:
hours.csv, with 6 sections, one for each hour in the day in ascending order. Each section should contain the top 10 busiest lifts for that hour. The header for each section should be:
Hour, Number of Rides

Bonus Points
1) Experiment with the number of threads you use to process the data. Can you come up with an optimal thread count to minimize the run time. This will be very dependent on the configuration of your machine so may take some exploration if you have a powerful multicore machine. Write up/plot your results in a .pdf file in your submission.

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 SkiDataProcessor. This class will be assumed to be the starting point of your program, and will be called from the command line.

2. The 3 .csv files generated by each version of your program.
3. All classes that you have developed for this assignment.
4. All abstract classes and interfaces that you extended and implemented in this assignment.
5. All classes that you have developed to test your code.
6. A UML diagram, corresponding to the design of your program.
7. A brief write-up, which summarizes the main relations between your classes, and how does your program handle errors and/or exceptions. .
8. If you attempted the bonus question, provide a short write up explaining how you investigated the problem and what results you obtained.

Deadline Monday 20th November 6pm PST