Assignment 7 Writeup

The goal of this assignment is to design and implement a concurrent program to process a large amount of data.

The main controller is the SkierDataProcessor class. It firstly parses the input csv files to a List<String[]> using a CsvParser, which is from an exotic package. Then it created two processors to process the data. At the end, it generates the output file using ResultAnalyser and IoLibrary. The run time for each processor is recorded and printed to console.

We implemented two processors to process the data:

1) Sequential processor
2) Concurrent processor

The sequential processor processes the elements in the input List<String[]> one by one, and records the required information to some data collections (see details below).

The concurrent processor parallelizes the data processing using the producer – consumer pattern. The producer stores data from the each record in the List<String[]> to three queues, which are skier queue, lift queue and hour queue. The consumer takes data from the queue and records the information to some data collections (see details below). There are three types of producers and three types of consumers corresponding to three queues. The process of reading data from input List<String[]> to the queue is a sequential process, whereas the process of taking the data from queue by consumers can be a multi-threaded process. In other words, each queue will have only one producer, but can have multiple consumers taking data from it. The number of consumers for each queue is a constant in the ConcurrentProcessor class called NUM_THREADS. (the bonus part will modify this number to see the optimal number of threads that gives the shortest run time)

The data collections used in the processors to store the information are:

Map<String, ISkier> skierMap

- The key is the skier id, the value is the skier object
- A map is used here instead of a list because we don't know the bound of skier id before reading the entire input data, and we need a key (the skier id) to map to the actual skier object
- This data collection records the number of rides that each skier has done and the total vertical meters the skier has achieved

List<ILift> liftList

- The index for each lift is converted from the lift id (index = lift id - 1)
- The lift objects in the list are created upon the initialization of the list, thus this list has a fixed size 40, corresponding to the number of lifts considered
- This data collection records the number of rides for each lift id

List<List<ILift>> hourRides

- The index of the outside list is calculated from the time (index = (time - 1) / 60)
- The index of the inside list is calculated from the lift id (index = lift id - 1)
- This data collection records for each hour the number of rides for each lift

The following the output csv files and how we generate the files based on the data collections above:

1) skiers.csv
    - contains the top 100 vertical totals in descending order. The headers are skier id and vertical meters
    - we used the values in the skiersMap and sort the skier objects by their vertical totals
2) lifts.csv
    - contains the number of rides for each lift in the ascending order of lift id. The headers are lift id and number of rides
    - we used the liftList to generate this file, the number of rides for each lift are fields in the lift objects
3) hours.csv
    - contains 6 sections, each section corresponds to one hour. Each section contains the top ten busiest lifts for that hour. The headers for each section are lift id and number of rides
    - we used the hourRides to generate this file. Each hour has a list of lifts, which are sorted by the number of rides

BONUS section

By comparison of run time between sequential and concurrent processors, we found that concurrent process takes longer than sequential process even when the number of threads is set to one. Increasing the number of threads does not help with minimizing the run time.

The sequential process takes about 400 - 600 ms on a MacBook pro 2014, whereas the concurrent process takes about 2000 – 2300 ms on the same machine when the number of threads is in the range of [1, 10]. When the number of threads is over 15, the run time of concurrent processor apparently increases. The reason of that is the threads management is gradually becoming the deterministic part of the time cost.