DISTRIBUTED SYSTEMS

Compulsory Assessment

In recent years there has been a great increase in the number of exercise (activity) tracking applications, activity tracker. These systems usually consist of a mobile frontend application to record activity and a backend system that takes care of data analysis. At the same time, many of these systems enable social networking among their users with features such as viewing leaderboards with the best runners per route, comparisons with the average and others. Well-known services available online are e.g. Strava and MapMyRun. Within the coursework you are asked to create a simple such system for analyzing activity tracking data.

These systems have the capacity to serve a large number of users. This means that each user maintains a personal "profile", i.e. a space where he/she can add his/her activities, which can then be analysed through the application and compared with other users in similar activities. He can also see his overall statistics, such as number of activities, total distance, total exercise time, etc. **Each user can either record and upload his activity at that moment through the mobile application, or upload an activity he had recorded previously.** An activity is a sequence of GPS waypoints. A waypoint is characterized by its coordinates (latitude, longitude), elevation and the time of recording. The sequence of these waypoints is stored in a special XML file format called GPX.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx version="1.1" creator="user1">
   <wpt lat="52.2423614748556" lon="5.281985213730702">
      <ele>-0.45</ele>
      <time>2023-03-15T10:41:51Z</time>
   </wpt>
   <wpt lat="52.24078476451067" lon="5.294344832871327">
      <ele>-0.06</ele>
      <time>2023-03-15T10:43:59Z</time>
   </wpt>
   <wpt lat="52.2360542973452" lon="5.291083266709218">
      <ele>-0.18</ele>
      <time>2023-03-15T10:45:24Z</time>
   </wpt>
</gpx>
```

Users via the mobile app upload this GPX file to the system and the system performs some processing on this file. A GPX file contains only one activity/route.

Usually the waypoints are created every ***x*** meters and this depends on the accuracy of the GPS of the device. Therefore the size of the file grows in proportion to the total distance covered during the exercise. The file can be processed in parallel by multiple machines in MapReduce format to speed up the processing of large files.

The MapReduce framework is a programming model that allows parallel processing of large volumes of data.

MapReduce relies on the use of two functions:

map(key,value) -> [(key2, value2)]

reduce(key2,[value2]) -> [final_value]

- "Map" function: processes a key/value pair and produces an intermediate key/value pair. The input to the map function can be lines of a file, etc., and are in the form (key, value). The map function converts each such pair into another pair (key2, value2). The map function can be executed in parallel, on different data input and on different nodes. The degree of parallelism depends on the application and can be defined by the user.
- "Reduce" function: merges all intermediate values associated with the same key and produces the final results. For each individual key, a list of values corresponding to that key is generated. This function calculates a final value for the key by processing the list of values corresponding to that key. The reduce function is processed after all map functions have finished processing.

The mobile application first sends the GPX to a Master Node. Then the Master Node creates chunks of ***n*** waypoints and sends them to the Worker Nodes in Round Robin order. Each Worker calculates for the received chunk **the total distance, average speed, total climb and total time.** It then turns these intermediate results back to the Master to be reduced to the final result. Therefore in this version of the system **the Master is also the Reducer.**

When the Master receives **all intermediate results** and completes the reduce, it outputs the final results for the activity. Finally, it **asynchronously** forwards the results back to the mobile application for the user to see.

At the same time the Master must keep statistics from all users. In particular, he keeps the **Average Exercise Time, Average Distance and Average Climb** for each user individually and for all users.

Backend implementation requirements:

- The **Master** must be implemented in **Java** and implement **TCP Server**. It is **not allowed** to use ready-made libraries beyond the default Java ServerSocket or HTTP protocol using a ready-made server, such as Java or Apache.

- The **Master** must be **multithreaded** and be able to serve multiple users simultaneously and communicate with the workers at the same time.

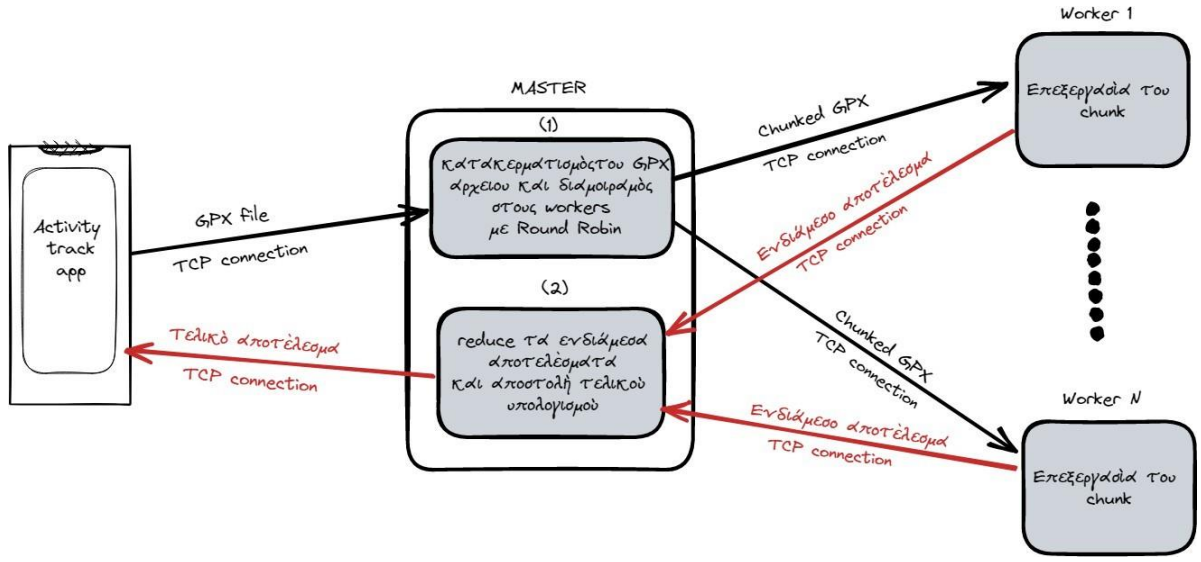
- **Workers** must be implemented in **Java** and be **multithreaded** to execute multiple requests from the Master in parallel.

- **Workers** should be defined dynamically during Master initialization (from arguments or config file) and their number can be arbitrary.

- The Master / Worker communication should also be implemented **exclusively through TCP sockets**. Specifically, the Workers must open a socket with the Master. Therefore **TCP Server will only be implemented by the Master.** You can implement two different TCP Servers on the Master, listening on different ports. One for communication with the different users using the Application and one for internal communication with the Workers.

- There must be timing at the points you deem necessary. Synchronization must be done **exclusively** using **synchronized, wait - notify** techniques and not using ready-made tools of the *java.util.concurrent* library or other ready-made tools.

- There is no need to use files to store statistics and user profiles. All data structures can be stored in the Master's memory. **The use of a database is prohibited.**



Frontend implementation requirements:

You will develop an application that will run on Android devices and provide an interface for the system. Through this the user:

- He will be able to select a GPX file stored on his device and send it to the backend for **asynchronous** processing.

- It should be able to receive a notification from the Master that the processing is finished and be able to show the results of the GPX processing **(total distance, average speed, total climb and total time)**.

- He will also be able to see his personal statistics (**Total Exercise Time, Total Distance and Total Climb)** that he has done and graphically display their difference from the general average (e.g. he has run a total of 24% more than the average).

- The communication between the Application and the Master should be done exclusively using **TCP Sockets**. The Application should be **connected to the Master with a TCP Socket**. Through this Socket the GPX is sent. The Master sends back the processing results **via the same Socket** which remains open until they are received. **This process should be implemented using Threads so that the application remains interactive until the results are received.**

Bonus (+20%)

In many applications, such as Strava, users can define a sequence of Waypoints as a Segment, e.g. a small part of the route of the Athens Classic Marathon. Then each time the system detects a sub-sequence

of Waypoints that is identified with a Segment can and does keep the previous user stats for that Segment. At the same time it keeps a leaderboard with the performance of all users in descending order. You are encouraged to integrate this additional functionality into the system by modifying MapReduce appropriately.
Finally, the android application should also display the leaderboard for the selected segment in a graphical way, e.g. in a table.

Observation: Two devices at the same point may read slightly different coordinates; e.g. show a difference of 5m. This is called GPS Drift.