A Simulated Bike Race
Course project - LSINF2345 - April 2016

Ruma Paul

A bike race is announced to take place at Louvain-la-Neuve on May 13, 2016. A student team from each institute of UCL must participate. From ICTEAM a team is built. One member of the team is a student of Informatics and that is You!! The team had a discussion about the strategy to win the race. Everyone proposed different strategies; resulting in a debate of whose strategy is the best. Being fed-up, you have come up with a proposal: "Let us do a simulation of the bike race". Everybody will play by following his/her own strategy, whoever wins we will follow that strategy in the race!! Everybody has agreed to your proposal. You, being a student of Informatics, have taken up the challenge to implement this idea.

Rules of the game: Everybody is given a fixed number of point, E, denoting the energy of each player. Using E, each player has to make a total distance of D. The player who finishes first is the winner!! At each round,

- A player can increase or decrease his/her current speed. The energy consumption is proportional to the speed.
- A player can be behind of another player. In that case the energy consumption will be lower.
- A player can use a *boost* Boost can be used only once and consumes all the remaining energy during next round. Boost is used to sprint at the end of the race.

Implementation Details: We will use Riak-core to implement this application. This application works in rounds. Each round is timed to take 10 seconds in real time at each player.

1. Initialization:
    a. Distance to cover = 100km
    b. $Energy_{Remaining}$ = 100
    c. My_Position = 0
    d. My_Speed = 0
    e. Round_Length = 10 seconds;

2. At each round t,
    a. Update Display: Display can be textual, where the position of each player is shown in a decreasing order. That implies, the player who has crossed maximum distance will be shown first. If one player was behind another they will be shown as a set, e.g., if player 3 was behind player 7 then it will show {3,7} = $Position_7$; Implementation of a GUI is optional.
    b. Calculate my speed during t-1:
        i. If I did biking by myself: $Speed_{t-1}$ = My_Speed;
        ii. If I was behind player k : $Speed_{t-1}$= $Speed_{t-1}(P_k)$
    c. Calculate my new Position: My_Position = My_Position + $Speed_{t-1}$;
    d. If I took boost during t-1, $Energy_{Remaining}$ =0; otherwise, $Energy_{Remaining}$ = $Energy_{Remaining}$ - b* $Speed_{t-1}^2$
        i. b = 0.5; if I was behind another player
        ii. b = 1.5; otherwise
    e. If $Energy_{Remaining}$!=0 then take an input from the user about his/her strategy during this round. There can be 3 possible inputs:
        i. New speed: My_Speed= $Speed_{input}$; or
        ii. Choose a player (P_B) to be behind who is at a position such that, Position(P_B) <= My_Position+ $Speed_{t-1}$;
        iii. Boost: My_Speed = 3.87*Sqrt($Energy_{Remaining}$).
    f. Broadcast my updates, i.e., My_Speed or decision to be behind P_B.

Installing Erlang and Riak-Core:

We suggest installing Erlang/OTP R16B or newer. Source and binaries can be found at https://www.erlang-solutions.com/downloads/download-erlang-otp and http://www.erlang.org/download.html.

Riak is a distributed NoSQL key-value store that offers high scalability, high availability and fault tolerance. Follow the lab "LSINF2345: A Simple Application using Riak-Core" to familiarize yourself with Riak. You can also try this blog: https://github.com/rzezeski/try-try-try; to get started with Riak-Core and build applications on top of it (Be Careful!! The bog is a bit old, so you need to configure some version parameters; otherwise you might face strange errors).

Deliverables

Turn in a zip file containing your src, and test directories, and a PDF report (of Maximum 5 pages). Your code should be properly commented (for reading/debugging purposes) and you should also include a README file with instructions for compiling and running it.

Optionally, you may include up to 2 pages worth of diagrams. A cover page is optional.

First we will look at your broadcast code, and visually inspect the behavior of the bikers during your interview with the assistant. The race should finish with a winner, even when some bikers crash.

Optionally, you can write no more than 150 words about any strange behavior observed or problems you have spotted while doing this project.

Problem 0: Setting the track, the uncoordinated bikers [**8 point**]

(Extension: Implementation of GUI for the display [Optional: 2 points])

We start by setting the track, where no biker fails and no special

consideration has been given to broadcasting messages. Implement best-effort broadcast, beb, and observe the behavior of the system for a number of bikers of your choice. It is possible that you will already observe strange behavior. Think about the properties of beb, and the way in which the bikers are periodically broadcasting what they think about the global state of the system and deciding based on that.

Problem 1: Same view for all bikers [**6 points**]

You will soon realize that message ordering is important to guarantee the same view to all bikers: the updates must be totally ordered across the bikers. Implement total-order broadcast: Totally–ordered delivery of updates and their execution. For this problem, you can assume that there are no failures (no bikers fall down).

Deliverable: Explain your solutions in your report. Aim for about one page. In particular, explain how you have implemented your total-order broadcast module and how that solves the problem of Problem 0.

Problem 2: Failing nodes with a bounded (synchronous) network [**6 points**]

To make things worse, suppose a biker may fall down during the race. Despite this, the race must go on.

Modify your solution of problem 1 to survive failing bikers. Assume fail-stop (crash-stop) failures. Once a biker has fallen down, he/she is out of the race. Your design may require the use of a failure detector. You can build a failure detector P by using a simple heartbeat algorithm. You can assume that this failure detector is perfect, if the heartbeat is chosen correctly. In order to select a time bound for P, determine how long is a reasonable time for a perfectly delivered message to be sent (heartbeat request) and returned (heartbeat response).

Deliverables: Explain your solution, and parameters for your failure detector, if used, in no more than 500 words. Your solution will be tested by letting some bikers fall-down during the race.

Marking

The assignment is out of a maximum of 20 points. The points for each section are explained in their "deliverables" section. 1 discretionary point is available for a nicely presented report and implementation code.

Where to find help

In order to find documentation for Erlang functions, try Google, the reference manual (http://erlang.org/doc/apps/stdlib/index.html) and possibly http://erldocs.com. To better understand the algorithms, refer to the course slides or textbook. The assistant will announce their availability separately. Ask them about Erlang problems, or to discuss your design. It's ok to talk to the other students about how to solve the problems, but don't copy their code.

Deadline: **Wednesday, May 11 @ 23:59**

Contact: ruma.paul@uclouvain.be