Politenico di Milano

Middleware Technologies for Distributed Systems

Project Report

A Simple Model for Virus Spreading

Author:
Fabio Losavio
Matteo Velati
Niccolò Zangrando

Supervisor:
Dr. Alessandro Margara

April 11, 2021

1 Introduction

1.1 Description of the given problem

Scientists increasingly use computer simulations to study complex phenomena. In this project, you have to implement a program that simulates how a virus spreads over time in a population of individuals. The program considers N individuals that move in a rectangular area with linear motion and velocity v (each individual following a different direction). Some individuals are initially infected. If an individual remains close to (at least one) infected individual for more than 10 minutes, it becomes infected. After 10 days, an infected individual recovers and becomes immune. Immune individuals do not become infected and do not infect others. An immune individual becomes susceptible again (i.e., it can be infected) after 3 months.

The overall area is split into smaller rectangular sub-areas representing countries. The program outputs, at the end of each simulated day, the overall number of susceptible, infected, and immune individuals in each country. An individual belongs to a country if it is in that country at the end of the day.

A performance analysis of the proposed solution is appreciated (but not mandatory). In particular, we are interested in studies that evaluate (1) how the execution time changes when increasing the number of individuals and/or the number of countries in the simulation; (2) how the execution time decreases when adding more processing cores/hosts.

1.2 Assumptions and Guidelines

The program takes in input the following parameters:

- N = number of individuals
- I = number of individuals that are initially infected
- W, L = width and length of the rectangular area where individuals move (in meters)
- w, l = width and length of each country (in meters)
- v = moving speed for an individual

- d = maximum spreading distance (in meters): a susceptible individual that remains closer than d to at least one infected individual becomes infected

- t = time step (in seconds): the simulation recomputes the position and status (susceptible, infected, immune) of each individual with a temporal granularity of t (simulated) seconds You can make any assumptions on the behavior of individuals when they reach the boundaries of the area (for instance, they can change direction to guarantee that they remain in the area)

1.3 Technologies

The technology involved in the development of this project is Open MPI.

2 Implementation

Having to monitor the movements of all individuals, and therefore more individuals there are more workload we would have, we decided to assign to each process a part of the individuals. Once the individuals are equally splitted, the entire population is initialized (each process assign to every its individual a random position and a random direction) and the computation can start.

Before entering in the details of the application is important to understand how an individual and a country are represented. An individual is represented as a struct as shown below:

```c
typedef struct individual_t
{
    int id;
    int contact_time;
    int is_infected;
    int infection_time;
    int is_immune;
    int immune_time;
    double position[2];
    int direction[2];
    int country_id;
} individual;
```

Listing 1: Individual representation

where

- *contact_time* represents the duration the individual is in contact with an infected one;
- *is_infected* is a boolean (0 or 1); represents if the individual is infected;
- *infection_time* represents the duration the individual is infected;
- *is_immune* is a boolean (0 or 1); represents if the individual is immune;
- *immune_time* represents the duration the individual is immune;
- *position*[2] represents the individual position (x, y)
- *direction*[2] represents the direction vector. It can be [1,0], [0,1], [-1,0], [0,-1]

- *country_id* represents the country the individual belongs to.

Also a country is represented by a struct:

```c
typedef struct country_t
{
    int id;
    int n_susceptible;
    int n_infected;
    int n_immune;
} country;
```

Listing 2: Country representation

where

- *n_susceptible* represents the number of susceptible individuals for that country for that process;
- *n_infected* represents the number of infected individuals for that country for that process;
- *n_immune* represents the number of immune individuals for that country for that process.

The implementation of the simulator can be summarized as follow.
For each timestep in a day, every process:

- for each individual, computes the new position based on the velocity and direction, computes the country the individual belongs to and saves the positions of the infected individuals.
- receives and sends to other processes the positions of the infected individuals and save them in a matrix, with the same dimensions of the world, setting a given position to one if at least one infect is in that position.
- for each individual, if it is susceptible, check if it is near enough to an infected one, using the previously built matrix to speed up the process, and update the *contact_time*, otherwise reset the *contact_time* to 0; if the *contact_time* is over the threshold, set the individual as infected. If the individual is infected, update the *infection_time*; if the *infection_time* is over the threshold, set the individual as immune. If the individual is immune, update the *immune_time*; if the

immune_time is over the threshold, the individual returns as susceptible.

At the end of each day:

- each process calculates the statistics of each country;
- for each country, the statistics are summed together and printed by process 0.

3 Experimental Results

3.1 Hardware

For all the tests we have used two VMs connected to the same local network. The two VMs are identical and runs on the same machine with the following specifications:

- CPU: Intel Core i5-8259U @ 2.30GHz
- RAM: 8GB

To each VM has been assigned 2 cores and 2GB RAM.

3.2 Tests

All the tests simulate 100 days.

3.2.1 Parameters: N=5000, I=50, W=10000, L=5000, w=10000, l=2500, v=1, d=5, t=300

1 core (local)	2 cores (local)	2 cores (2 hosts x 1 core)	4 cores (2 hosts x 2 cores)
176.734 s	97.448 s	113.847 s	89.006 s

3.2.2 Parameters: N=10000, I=100, W=10000, L=10000, w=10000, l=5000, v=1, d=5, t=300

1 core (local)	2 cores (local)	2 cores (2 hosts x 1 core)	4 cores (2 hosts x 2 cores)
356.658 s	208.092 s	222.457 s	172.547 s

4 Conclusion

What we have observed from the tests is that using more cores/hosts increases the performance. On the other side, by using the same number of cores, the performance decrease if we run the application in the cluster. This is due, of course, to the network latency: the more powerful the network the better performance you will get.

5 References

- Open MPI documentation:
  https://www.open-mpi.org/doc/current/
- Setting up an Open MPI cluster within a LAN:
  https://www.vitaarca.net/post/tech/setting-up-openmpi-cluster-within-a-lan/