A Simple Model for Virus Spreading

Design Document for the MPI project of the course "Middleware Technologies for Distributed Systems".

Authors:
Accordi Gianmarco
Buratti Roberto
Motta Dennis

POLITECNICO
MILANO 1863
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Italy

24/02/2021

Contents

1 Introduction

The purpose of this report is to explain the work done by this group on the project "A Simple Model for Virus Spreading" for the "Middleware Technologies for Distributed Systems" course of "Politecnico di Milano". The analysis of the task to be solved has been analyzed and taken from the projects' pdf provided by the professors [2].

2 Project Analysis and Assumption

The **goal of this project** is to define a simple model for the spreading of a virus. In our project the model we have considered takes into analysis an area of dimension W x L, that can be divided in an integer number of small areas of dimension w x l: we have assumed that W is a multiple of w and also L is a multiple of l, as shown in Figure 1. In this way we have that areas are equal among them, and also the whole global area is covered by this small areas. From now on we refer to the small area as sub area.

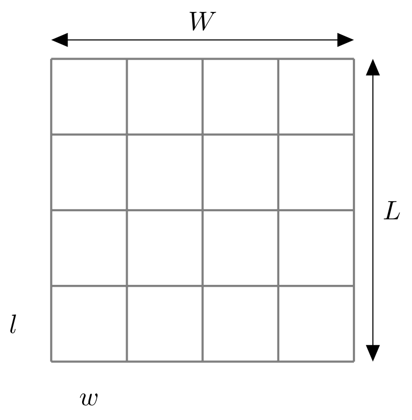


Figure 1: Model of the global area.

The model takes into consideration N individuals that are equally spread among the various sub area: if N is not a multiple of the number of areas, then some areas will have to manage more individuals. The same assumption is applied to the number of initially infected user. Then a user becomes infected if it is not already infected (so if infected cannot restart its infection) or immune, and if it has remained near an infected user (at a distance of less than d meters) for more than 10 minutes. If this is the case the individual (that in the model is represented by the User class) becomes infected for 10 days, and after that it remains immune for 3 months.
Individuals can move among the various areas, and they can infect users in other areas if near the border. This means that our model has to take into consideration that, before checking if a user has been infected, each area should exchange information among them on the users near the border, and also of users that have exited the area: this communications is done using the MPI framework.

2.1 Processors and Area

Our model split the work as much as possible equally to the various processors. Each area is managed by a processor, and at each area we associate a different ID, as shown with an example in Figure 2.

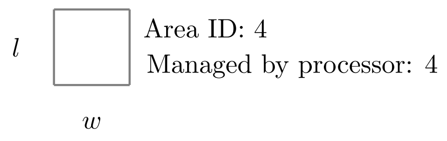


Figure 2: Model of the sub area.

So the program achieves the best performance if the number of areas is a multiple of the number of processors, otherwise some processor will have a slightly different workload. The workload of a processor

at a given time depends on the actual number of users it has to manage, and the number of users near its borders, since such value changes during computation.
Upon creation each area will create a data structure in order to save which are the near areas in each direction: this data structure will contain the unique ID of the area in the global area, and the processor that is responsible of managing that certain area.

2.2 Users and Areas

The model accept as input also a velocity v that is used to update the user position at each step, based on the provided time step t. Users can move also among near areas: at each steps after updating the user's position we check which area is responsible to manage it. If the user is near a border of the global area than the model computes the intersection between its previous direction and the border from which the user has exited the sub area, and than it get a new random direction. Otherwise we find the near area to which the user is directed based on the border from which the user has exited. It may be the case that when the velocity is too high, the user is able to go outside the global area or in general go to a another area, that is not one of the near regions: in this case, in order to simplify the model, we associate the position of the user to the center of the near region in the given direction from which the user has exited its previous region.

2.3 Users and Processor

Since users are managed by different areas, this means that they may be managed on different processors: when areas have to exchange information among the users near the borders, they have to find out if the information of users near a certain border has to be sent to a local area (a near subarea that is managed by the same processor) or to a remote area (a near area that is managed by another processor). To make computation faster, each area will maintain data structures of the users actually near its internal border and users that have exited the subarea. Since each area will send users only to near areas, if we have the case that a user has gone too far away in a single time step (so it already exited the area to which we are sending this user to) we reset its position to the center of such area.

3 Main Loop Pseudocode

The next figure provides a snippet of the main loop that is executed by the *main* at each time step:

```
Data: program input
Result: infection status at each day
initialization;
while accumulated elapsed time is lower than the target number of days of simulation do
    if accumulated elapsed time is multiple of a day then
        print the actual infection status of the computation in a csv file;
    end
    for Areas in processor do
        Collect users that have gone out-of-area in locally managed areas;
        Collect users that are near the borders in locally managed areas;
    end
    Exchange users out-of-area between local areas on the same processor;
    Exchange users near borders between local areas;
    for p in processors do
        if p is the same as the current processor then
            Collect out-of-area users to this processor;
            Collect users near borders in other processors that are interest to this processor;
        else
            Send out-of-area users to p;
            Send users near borders of interest to p;
        end
    end
    for Areas in processor do
        Update infection status of users in the area;
    end
    for Areas in processor do
        Update position of users in the area;
    end
end
```

Algorithm 1: Main Loop Pseudocode.

Remember that this code is executed on each processor, following the MPI paradigm, so *current processor* in this pseudocode is the the one that is actually executing the code. This pseudocode highlight that the processors has to exchange among them information about users that have gone to another area, and on the users that are near the borders of interest.

4 MPI Communication

As stated in the previous sections processors has to exchange information about users that have gone out-of-area and about users that are near borders. The communication paradigm we have used is the same for both the cases, and it is shown in the following pseudocode. In order to do so we have used Open MPI [3] as shown:

```
for processorID in processorIDs do
    Synchronized with MPI_Barrier(MPI_COMM_WORLD);
    if myID==processorID then
        Allocate enough buffer in order to receive from each processor information about the
          amount of data it will send;
    else
        Prepare the information about the amount of data to be sent;
    end
    Perform an MPI_Gather;
    if myID==processorID then
        Allocate enough buffer, based on the information received in the previous gather, in order
          to receive from each processor the data it will send;
    else
        Prepare the data to be sent;
    end
    Perform an MPI_Gatherv;
end
```

Algorithm 2: Communication paradigm.

As the previous pseudocode this one is executed on each processors. So this code is executed for each processor ID on each processors, then in the first part each processors will gather the amount of data it will receive from each other processor, and then it allocate enough memory for receiving such data from each other processor. To do so each processor uses as primitive MPI_GATHERV [1].

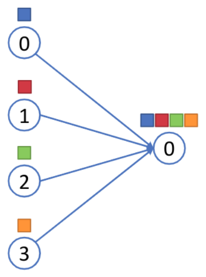


Figure 3: How gather works in MPI.

5 Integration with the Spark project

The output of the model is saved in multiple csv files (one file for each processor) in a way that it is easy to be processed by the project called ”Analysis of COVID-19 Data using Apache Spark”. So the output of this model can be directly put as input of the Apache Spark project.

6 Performance analysis

In the following we provide some graphs about the performance of our model under different loads.

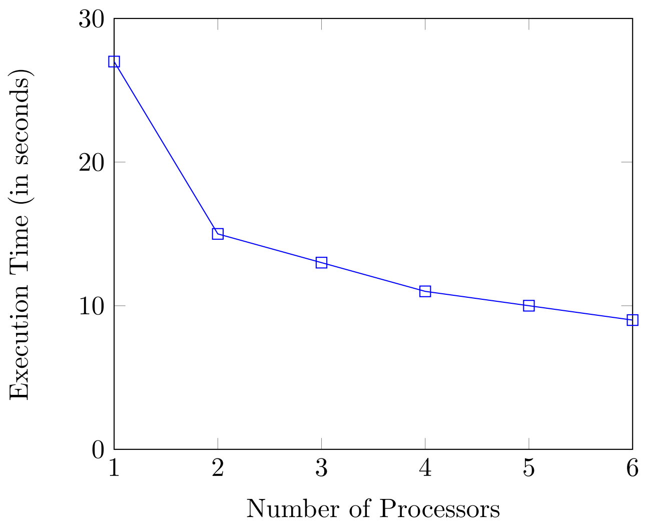


Figure 4: Shows the the relation between the execution time and the number of processor used, with the same amount of users and area, for a week

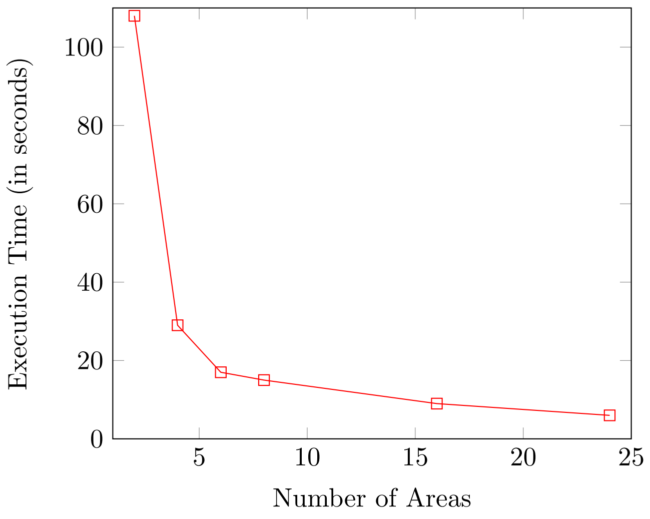


Figure 5: Shows the the relation between the execution time and the number of areas, with the same amount of processors (6 in this case) and users, for a week

The previous graph highlight how having more areas allows for a better balancing of the computation among the processors (since each area is assigned to a different processor). And also that small areas are more efficient since they have to check the distance among users in the same area a lower number of times, and they can also rely on better structured data since they have saved the users that are near the border.

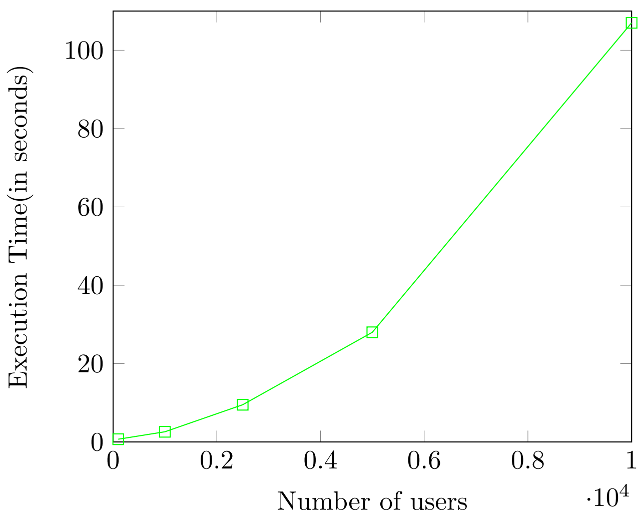
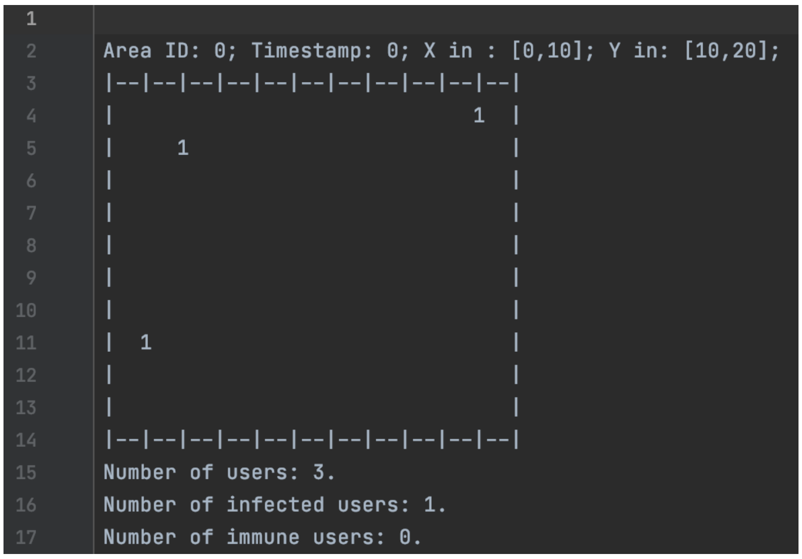


Figure 6: Shows the the relation between the execution time and the number of users, with the same amount of processors (6 in this case) and area, for a week

7 Testing

We created many automatic tests using the Catch2 test framework. We used these tests to check the correctness of the various classes that we developed, like Position, Area and User.
We also tested manually the correctness of the algorithm by writing a debug function that prints the whole area in a file using ASCII codes. So we were able to check the movements and the status of the users visually as can be seen in Figure 7.



```

Area ID: 0; Timestamp: 0; X in : [0,10]; Y in: [10,20];
|--|--|--|--|--|--|--|--|--|--|--|
|                             1  |
|     1                          |
|                                |
|                                |
|                                |
|                                |
|                                |
|  1                             |
|                                |
|                                |
|--|--|--|--|--|--|--|--|--|--|--|
Number of users: 3.
Number of infected users: 1.
Number of immune users: 0.
```

Figure 7: An example of the output of the visual debugger that we implemented.

References

[1] GatherV examples. https://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node70.html.

[2] Luca Mottola and Alessandro Margara. Middleware technologies for distributed systems - exam projects 2020/2021, 2021.

[3] The Open MPI Project. Open MPI FAQ. https://www.open-mpi.org/faq/.