An approach to parallelization of respiratory disease spread simulations in emergency rooms

Martín Paradiso
Informatics Research Institute LIDI.
Universidad Nacional de La Plata
La Plata, 1900, Argentina
martinparadiso@outlook.com

Lucas Maccallini
Informatics Research Institute LIDI.
Universidad Nacional de La Plata
La Plata, 1900, Argentina
lucas.maccallini@gmail.com

Agustina Vericat
SISDIC Hospital San Roque de Gonnet
La Plata, 1900, Argentina
SimHPC-TICAPPS
Universidad Nacional Arturo Jauretche.
Florencio Varela, 1888, Argentina
agustina.vericat@hospitalelcruce.org

Fernando Romero
Informatics Research Institute LIDI.
Universidad Nacional de La Plata
La Plata, 1900, Argentina
fromero@lidi.info.unlp.edu.ar

Diego Encinas
Informatics Research Institute LIDI.
Universidad Nacional de La Plata
La Plata, 1900, Argentina
SimHPC-TICAPPS
Universidad Nacional Arturo Jauretche.
Florencio Varela, 1888, Argentina
dencinas@lidi.info.unlp.edu.ar

***Abstract*—The agent-based modeling (ABM) is a flexible simulation model that can be easily adapted to the simulation of different problems. A disadvantage of some tools for ABM implementation is the low performance obtained. This paper presents a re-implementation of an in-hospital disease simulator, developed in Repast Simphony, in its high-performance alternative Repast for High Performance Computing. This tool allows multinode execution, allowing execution on multi-core machines as well as on physical and virtual clusters.**

The objective of this work is to analyze the performance of the new implementation in physical and virtual clusters, and to determine in which scenarios it is justifiable to implement a system in a lower-level framework.

***Index Terms*—Agent-based modeling and simulation, Disease transmission, HPC, Simulation**

I. Introduction

In the research conducted by Maccallini [1], an emergency room simulator was developed with the aim of investigating the transmission of in-hospital diseases. In particular, the transmission of pneumonia is investigated. For the study, a hospital was designed and statistics were collected from various sources [2], [3], [4] in order to obtain a model with which to calibrate the system parameters. This implementation is designed following the agent-based modeling [5].

The original simulation designed by Maccallini [1] is implemented in the Repast Simphony *toolkit* [6], which focuses on the simplicity of the development. While the framework is sequential, it supports parallelism [6] by delegating to the user the responsibility of proper synchronization. Due to this, the original simulator is completely sequential. This is why the performance of the simulations is not conducive to a large number of executions or systems with a large number of agents. Repast HPC [7] is a "high performance" tool that uses the same concepts as Repast Simphony, providing a reduced set of functionalities, in exchange for allowing distributed execution through MPI and a parallel design.

This investigation aims to re-implement the simulation —developed in Repast Simphony— in Repast for High Performance Computing, allowing us to compare both frameworks. In particular, Repast HPC is aimed at large-scale simulations, the implementation of this model will give us insights about the viability of said framework for small models. Furthermore, a faster version of the model will result beneficial while conducting large amount of experiments.

A summary of Maccallini's original model [1] is presented in section II. Section III presents Repast for High Performance Computing, and section IV the new implementation. In section V the infection results of both implementations are shown, while section VI offers a deep analysis in the performance response of the new implementation, in single-node and multi-node environments, both in virtual clusters and cloud. Conclusions are presented in section VII.

II. Simulator Development

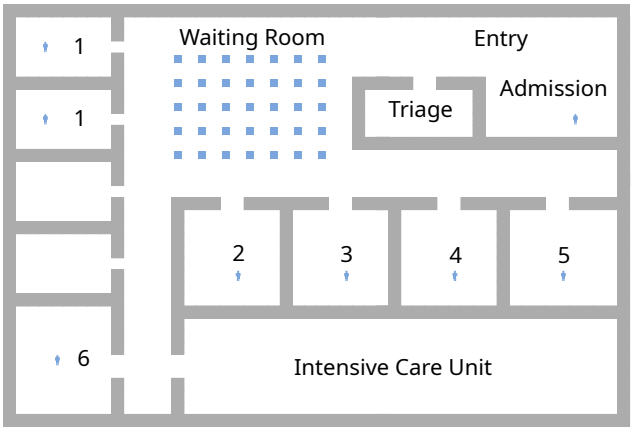
The simulation model used is the *agent-based modeling* [5]. Its main feature is the existence of a set of entities, *agents*, which interact with each other and allow predictions of the modeled system behavior. The system is executed by *ticks*, where each agent executes its own logic.

The final model obtained by Maccallini contains the patient circulation logic, the transmission logic, a hospital, and various distributions: rate of patient entry per year and day, severity of diagnosis, assigned area and hospitalization times.

979-8-3503-2028-2/22/$31.00 ©2022 IEEE
DOI 10.1109/CSCI58124.2022.00295

A. Hospital

The hospital (Fig. 1) used in the research is "unreal", based on the design of several hospitals. In particular, the emergency area is implemented, which contains a waiting room, reception, triage, medical offices and intensive care unit. In order to provide a more realistic scenario, the hospital contains the most common external services.



1) Consulting rooms
2) Surgery
3) Pediatrics
4) Gynecology
5) Geriatrics
6) Psychiatry

Fig. 1: Hospital map

As a special case, the ICU (Intensive Care Unit) has no physical representation, the patients assigned to it are removed from the map. To model contagion within this area —which was of no particular interest— a special type of transmission described below is used.

B. Disease transmission

The model designed by Maccallini describes 4 situations of transmission:

- between two people,
- between an object and a person,
- between a person and an object, and
- between a person and the ICU environment.

In this model, persons and objects have different infection cycles. People have three states: healthy, incubating and sick; in the last two states the person is able to infect those within 2 meters distance. Objects, on the other hand, have two states: clean and contaminated; an object becomes contaminated when it "interacts" with an unhealthy person, the contaminated state allows it to infect people.

1) The route of transmission: The main route of human respiratory infections is that produced through droplets of Pflügge (smaller than 5 microns) caused by speaking, coughing or sneezing by the infected person, subsequently reaching the oral, nasal or conjunctival mucosa of a new host. These droplets do not remain suspended in the air, but settle quickly, so they only allow transmission at a distance of just under 1-2 meters [8].). Transmission also occurs by direct contact of the mucous membranes with the respiratory secretions of an infected person or, indirectly, through recently contaminated hands or objects [9].

Objects are cleaned periodically, unconditionally passing to the clean state. Adequate cleaning was considered to be that carried out with disinfectants such as sodium hypochlorite, chlorine, quaternary ammonium, hydrogen peroxide —among the most important— on contaminated surfaces [10].

The special case of infection through the ICU environment (which has no physical representation, and therefore there is no person-to-person transmission) is performed by means of equation 1. It is evaluated at each simulation *tick* and applied to each patient.

$$p_{ICU} = N_{patients} * C_{ICU} \tag{1}$$

Where:

- p_{ICU} is the probability of infection,
- $N_{patients}$ is the number of patients currently in the ICU, and
- C_{ICU} is the coefficient of proportionality.

C. Patient behavior

Patients implement a behavior that tries to resemble the general path that a person takes when entering an emergency room.

The system generates a configurable number of agents in a period of one year, taking as a reference a distribution that is scaled according to the number of agents to be created. For each day there is a probability p_i that a patient is infected.

The path that a patient takes inside the hospital once he is admitted is as follows:

1) He looks for a place in the waiting room.
2) He queues at the reception desk and waits his turn.
3) He goes to the reception desk, where he remains for a pre-determined time.
4) He looks for a place in the waiting room.
5) He queues at triage and waits for his turn.
6) He goes to triage, where he remains for a pre-determined time.
7) a) If his diagnosis is for a physician.
   i) He goes to the waiting room.
   ii) He queues at the assigned medical specialty.
   iii) Once it is his turn, he goes to the assigned physician.
   iv) Is seen for a period of time.
   b) If his diagnosis is for ICU admission.
   i) He requests a bed in the ICU.
   ii) He goes to the ICU, where he stays for the time dictated by the triage.
8) Goes to discharge.

There are several situations in which the patient heads for discharge early: if there is no room in the waiting room (step

1, 4 or 7(a)i); if he/she is not seen by the physician before the time limit established in the triage (step 7(a)iii), or if there are no beds available in the ICU (step 7(b)i).

D. Other characteristics

Medical personnel have a probability of being protected (through personal protective equipment for respiratory infections, such as a mask and gloves) and not participating in the contagion process. If they are not protected, when they are infected and show symptoms —after the incubation period—, they are replaced by healthy personnel.

Patients have a triage priority classification, which indicates the time limit for their medical attention. If patients are not seen before this time (because the less seriousness of their pathology allows it) they are given an appointment scheduled for the following days and they leave the hospital. Since the simulator models only the emergency room, these scheduled patients are not implemented. This difference in severity implies that each patient must "queue" in each medical specialty according to their severity, and not in a *FIFO* order as in reception and triage.

III. Repast for High Performance Computing

Repast HPC [7] is a high-performance alternative to Repast Simphony, it is implemented in C++, provides less functionality and lacks an IDE, but has an interface similar to that offered by Repast Simphony. The main focus of this alternative is cluster execution using the MPI standard.

A. Parallelism

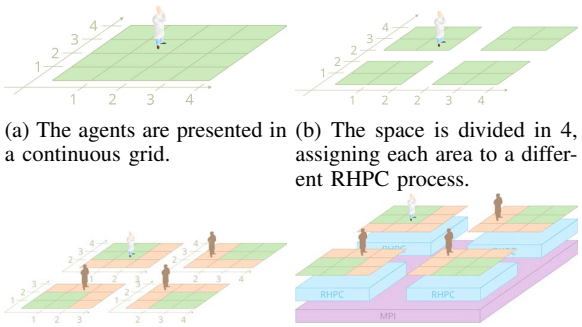
Repast HPC parallelizes the simulation *spatially*. The user defines a plane and assigns each agent a dynamic location. The framework takes care of assigning to each process an area of the plane, and consequently the agents that are in that area. To avoid discontinuities, agents that are close to the boundary of a process are copied to the adjacent process so that they are visible to nearby agents.

As an example, Fig. 2 is included, showing the division logic used by Repast HPC. The replicated agent, and the tiles external to each process, are shown in orange (Fig. 2c and 2d). While this implies that the same agent exists in two processes simultaneously, the *non-local* copies are read-only, cannot execute logic, and any changes in their internal state are not reflected in the original process. To maintain consistency, Repast HPC copies the agents in every simulation tick.

The space partitioning is determined by the user at the start of the simulation, and cannot be changed. For a given number of processes N, the user can request any combination of partitions along each axis, as long as the resulting number of partitions matches the number of processes N. For instance, while using 4 processes (as in Fig. 2), there are three possible *configurations*: 1x4, 4x1, and 2x2 (the one used in Fig. 2).

IV. Design and implementation

The system is designed to replicate the original model [1] while minimizing changes.



(a) The agents are presented in a continuous grid.

(b) The space is divided in 4, assigning each area to a different RHPC process.

(c) The original agent is replicated in the adjacent process as a read-only copy.

(d) Final state, showing the software stack. Each RHPC process can be in different machines.

Fig. 2: Space partitioning and agent distribution in Repast HPC. Since the framework uses distributed memory, the agents close to the border must be copied as "read-only" to the adjacent process.

A. Input file

Part of the objectives is to decouple the data from the code, allowing the reuse of the system for other hospitals without the need to modify the code. To achieve this, all relevant information is stored in a `.json` file:

- walls, chairs, doctors, entrance, exit;
- infection probabilities;
- triage levels, and maximum duration of each level;
- admission times;
- probabilities of infection, contamination, incubation times and cleaning of objects; and
- speed of patients, rate of entry, probability of pneumonia.

B. Multi-process

One of the difficulties in development is parallelism in distributed memory. Repast HPC provides agents as the only synchronization method, and lacks the functionality to synchronize global logic such as queues and shifts. This is why various aspects of the hospital —in particular those modeling shifts and queues— were modeled directly using MPI and not the framework.

Algorithm 1 contains the pseudo-code used by the patient when requesting a bed in the ICU. The `patient_logic` function is called once per tick, and depending on the instance the patient is in, its behavior changes. The patient first requests a place in the ICU, and then queries periodically until he gets an answer. Once he gets an answer, he checks if the answer was positive (there is a bed available) or not.

This polling architecture is used to model the queues for reception, triage and doctors, and chair and bed assignment.

C. Transmission logic

The transmission logic is modeled around the types of contagions: between two persons, between an object and a person, between a person and an object, and between a person

Algorithm 1 Sub-system usage example, the code represents a section of `patient_logic` function

```
if self.stage is REQUEST_BED_IN_ICU then
    icu.requestBed(self.id)
    self.stage ← WAIT_RESPONSE
else if self.stage is WAIT_RESPONSE then
    response ← icu.getResponse(self.id)
    if response then
        if response.bedAssigned then
            self.stage ← GOTO_ICU
        end if
    end if
    else
    self.stage ← GOTO_EXIT
else if self.stage is ... then
    ...                                  ▷ Rest of the states
end if
```

and the ICU environment. Two types of infection cycles are then defined: human and object, which model the previously mentioned cycles. Contagion in the ICU is modeled through an "environment" rather than an agent.

To model contagion, 2 probability parameters are defined (different for each cycle): probability of *contagion* and probability of *contamination*. The first parameter is used when the "victim" is a human, while the second is used when the victim is an object. Furthermore, each agent type can have different values for each parameter, allowing for more customization points in the model. The values for this parameters is shown in section V.

This scheme allows to generate any combination of interaction between any pair of agents. For example, in the case of a healthy person, both probabilities are 0. A contaminated object has a probability p_o of infecting a person, and a probability 0 of contaminating another object.

1) Contagion attempt: The "contagion attempt" used by both cycles boils down to:

1) Obtaining the probability of the offending cycle or environment (can be another patient, chair, bed). The probability is retrieved from the system, the value depends on the pair of agents involved.
2) Generating a random number in the range $[0, 1)$.
3) If the random number is less than the probability of contagion obtained in 1), the cycle is infected or contaminated.

For each healthy person and clean object currently in the simulation, the model iterates over the agents sorrounding said agent, and executes the contagion attempt logic between the two agents. The iteration is performed once per simulation tick.

D. Medical personnel and objects

In order to increase performance and provide greater flexibility to model the system, the medical staff is not logical. Shifts are administered by the queues and not by the agents representing the medical staff. In each turn, the condition of each physician is periodically observed, and if a sick physician is detected, he/she is replaced by a healthy one.

The objects (chairs and beds) were modeled outside Repast HPC, i.e., *they are not agents*, in order to increase the performance of the system. This is because Repast HPC synchronizes agents by proximity, and since objects interact with patients in the same location, it is unnecessary to synchronize them.

V. Calibration

The calibration of the system consists of adjusting parameters so that it resembles the values collected by Maccallini [1] in his project. In particular, the relevant metrics to match are Point Prevalence —percentage of infected patients— and the Point Prevalence in the Intensive Care Unit, —percentage of infected patients in the ICU—.

TABLE I: Calibration results for the point prevalence.

Data set	Point Prevalence (%)		
	Reference	Maccallini	Actual
Calibration	1.190	1.244	1.156
Validation	1.220	1.713	1.284

TABLE II: Calibration results for the ICU point prevalence.

Data set	Point Prevalence in ICU (%)		
	Reference	Maccallini	Actual
Calibration	3.370	3.398	3.348
Validation	3.180	3.203	3.543

As can be seen in Table I and II, the results are closer to the reference values than in the original version of the simulation, so they are taken as valid. With these values we proceed to the system performance analysis, which is the main objective of the project.

The parameters used to obtain the previous results are shown in Table III. As explained in previous sections, the probabilities are splitted in two groups, depending on the victim (in this table labeled as *target*). The following table contains every possible combination of disease transmission, where *source* is an infected person or a contaminated object, and *target* is a healthy person or clean object.

TABLE III: Parameters used in the new Simulator in order to obtain the same results as Maccallini [1].

Target	Source		
	Person	Chair	Bed
Person	4.9×10^{-2}	3×10^{-5}	7×10^{-8}
Chair	4×10^{-4}	0	0
Bed	3×10^{-5}	0	0

VI. Performance analysis

First of all, it is worth remembering that the parallelism in Repast HPC is given in terms of a spatial partition of the

hospital plane. The framework takes as parameters the number of partitions to be performed along each dimension (x and y). This implies that the use of, for example, 4 processes, requires the partition of the space in 1 of 3 possible combinations:

- $(1, 4)$: the hospital is partitioned into 4 horizontal "strips", i.e., the y-axis is partitioned.
- $(4, 1)$: the hospital is partitioned into 4 vertical zones of equal size, i.e., the x-axis is partitioned.
- $(2, 2)$: the hospital is partitioned into 4 equal quadrants.

The "local" benchmarks are performed in a tuned virtual machine, which offers the same performance as a bare-metal operating system. The host is a AMD Zen2 8c16t CPU; each guest has 4vCPUs, 8GB of RAM and runs Debian 11.

First, we run the simulation in Repast Simphony (the original version), whose average performance is 187 s. Then, the new simulation (implemented in Repast HPC) is run in a single node with the data and parameters obtained in the calibration. Table IV shows the results obtained; the Time column is obtained from the median of 10 runs for each configuration.

TABLE IV: System performance using the calibration data set.

Processes	Configuration (x, y)	Time (s)
1	(1, 1)	200
2	(1, 2)	209
	(2, 1)	334
3	(1, 3)	393
	(3, 1)	426
4	(1, 4)	359
	(2, 2)	312
	(4, 1)	390

The data in Table IV are plotted in Fig. 4, sorted by time. As can be seen, the performance is far from being an improvement.

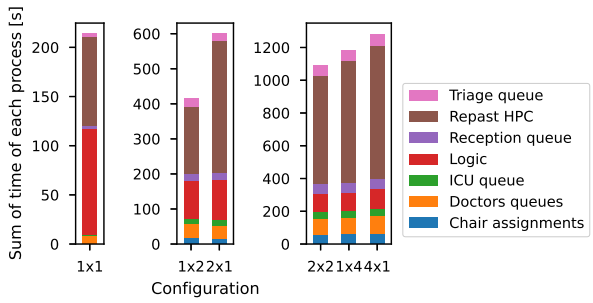


Fig. 3: Time distribution in one, two, and four processes simulation. For each configuration, an internal measurement of the time spent in each section of the simulation is performed. Then, the individual times of each processes are added.

Checkpoints are introduced into the code in order to measure the distribution of the time taken for the simulation. A graph generated from such data is shown in Fig. 3 (first plot). It can be seen that even in single-process execution, where there should be no synchronization, about 40 % of the execution time is consumed by the framework.

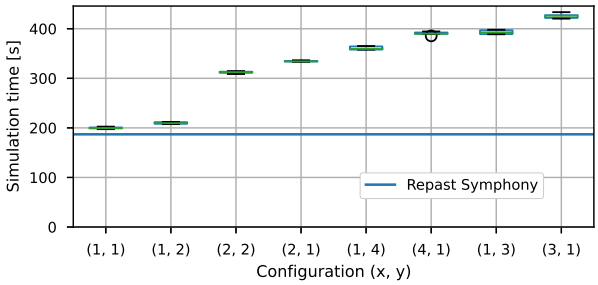
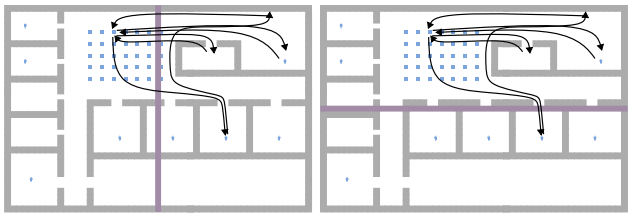


Fig. 4: System performance using calibration data set (65 713 annual patients). The blue line (Repast Symphony) is the performance of the original simulation developed by Maccallini [1].

In addition, internal metrics are collected from the executions in two processes (see Fig. 3, second plot), which have between them a large difference in performance. The execution $(1, 2)$ takes 209 s while the configuration $(2, 1)$ takes 334 s, 60 % more.



(a) If the x-axis is divided (configuration 2x1), the patient switches processes several times.

(b) If the division is performed in the y-axis (configuration 1x2), the worst case scenario involves only 2 process switches.

Fig. 5: The diagram shows the space partition (purple line) and one possible path of a patient. Space partitioning has an influence in performance, since the number of movements between processes is determined by the agent path and the map distribution.

The change in performance between one configuration and another can be understood by analyzing the partitioning of the hospital (lengthwise or widthwise) and the distribution of the hospital areas. If partitioned vertically (Fig. 5a), the patient "crosses" the hospital meridian several times, generating a synchronization between the two processes running the hospital. This does not occur if it is partitioned horizontally (Fig. 5b), leaving the entrance, chairs, reception and triage in the same process.

Furthermore, Fig. 6 shows the time distribution inside each process, for both configurations (1x2 and 2x1). This measurements confirm the increased synchronization time (in brown, Repast HPC) between processes when using an improper

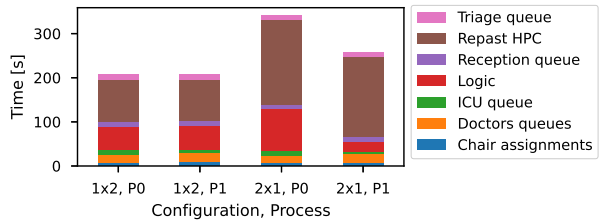


Fig. 6: The internal distribution of each process in both 1x2 and 2x1 configuration is measured and plotted. The analysis shows the increased synchronization time (in brown) when using a "bad" partition.

partition. Not only the synchronization increases, but also the logic becomes unbalanced, plotted in red in the same figure.

A similar situation occurs in the case of 4 processes (Fig. 3, third plot), although on a smaller scale, since the performance in all cases is poor (due to the above mentioned). The best case (which is still bad compared to the original) is when the simulation is distributed equally. On the other hand, if one axis is "over-partitioned", the performance worsens, as the agents "cross" too many processes to reach their destination.

A. *Varying the number of agents*

In addition to the original simulation, the responses of both the original version (implemented in Repast Simphony by Maccallini [1]) and the new parallel version are tested by varying the number of agents. The behavior is summarized in Fig. 7. As detailed in the legend, the execution time is measured for all combinations of up to 4 processes, varying the number of patients between 0 and 70 000. There are two key points that can be drawn from the graph:

- The performance of the framework without patients, and consequently, almost without agents (there is still the staff), is considerably poor.
- As the number of agents increases, the difference between single-process execution and the best multi-process configurations is reduced.

Furthermore, Fig. 7 shows a consistent performance degradation in performance for each patient increment.

This leads one to believe that Repast HPC ——despite having considerable overhead—— is geared towards considerably larger space simulations with larger numbers of agents. Other models that make use of Repast HPC, simulate from 100 000 concurrent agents [11], up to 3 000 000 [12].

Since the performance of the original sequential version of the simulation starts to approach the new parallel version at 60 000 annual patients, an extended test was performed. The extended test (Fig. 8), measures the performance while varying the annual patients from 60 000 to 120 000.

The number of patients exceeds the capacity of the hospital designed by Maccallini [1], which can be observed by the increased number of derivations to other hospitals (Fig 8, green line). This implies a maximum number of concurrent agents

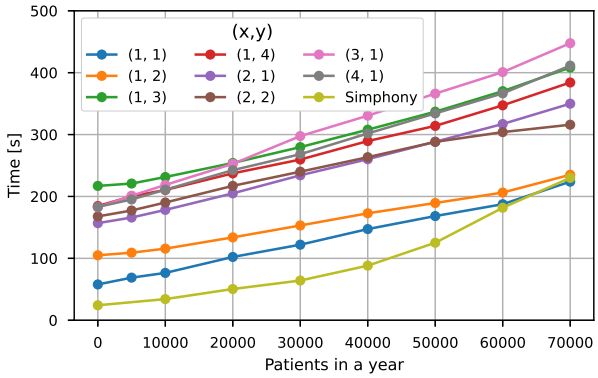


Fig. 7: System performance while varying number agents (x-axis), of processes, and configuration (each line). The olive line is the original simulation in Repast Simphony. All measurements where performed in the same environment described in Performance Analysis.

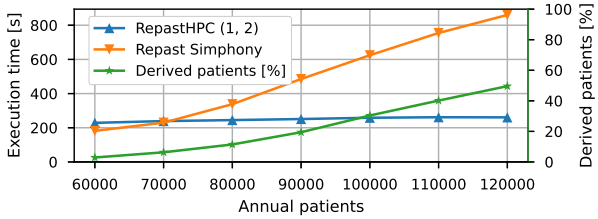


Fig. 8: Simulation response while varying the number of patients. The test is performed in the original simulation and a two-process simulation (1, 2) in Repast HPC. Since the hospital was designed for 65 000 annual patients, the number of derivations due to saturation increases significantly after this point.

is reached, due to patients arriving at the hospital and been immediately derived due to lack of free chairs. Nevertheless, a favorable scenario can be observed, since the new parallel simulator (blue line in Fig. 8) offers a better performance when the number of annual patients exceeds 70 000.

B. *Cloud Performance*

Due to the growing popularity of cloud computing services, we chose to run the simulation on Google Cloud Platform, to observe how the simulation responds when run in an unpredictable environment such as cloud.

The instance used is `n2-standard-2`. Each node contains 2 vCPUs (Intel Xeon), 8GB of RAM, and Ubuntu 20.04.

Each node executes a single MPI process, this means that, for a 2 process simulation, the communication is performed via network. Due to the random nature of instance allocation, it is impossible to determine is said network relies on a physical or virtual communication path.

TABLE V: System performance in Google Compute Platform.

Configuration (x, y)	Patients	Time (s)
(1, 1)	0	62
(1, 1)	25 000	152
(1, 1)	50 000	237
(1, 1)	70 000	281
(1, 2)	0	14 971
(1, 2)	25 000	14 161

As can be seen, the system performance when using 2 nodes, which *communicate via SSH* (as documented in the manual [13]), is 2 orders of magnitude worse. In order to determine the origin of the performance loss, the simulation is measured on two virtual machines running on the same machine, this minimizes the network factor since the network interface used by the Virtual Machines are not limited by a physical medium. But again, a considerable performance impact is found, which is detailed in Table VI.

TABLE VI: System performance in VirtualBox.

Configuration (x, y)	Patients	Time (s)
(1, 2)	0	4379
(1, 2)	65 713	9148

Although the performance does not worsen as much as in the cloud case, it exceeds what is expected. To complete the analysis of the situation, the internal metrics of the execution in virtual machines are extracted. This analysis is shown in Fig. 9, where the relative increase of the sub-systems implemented outside Repast HPC can be observed. From this graph, it can be determined that Repast HPC, in addition to being oriented to the execution of a large number of agents, was optimized to minimize the impact when using network communication.

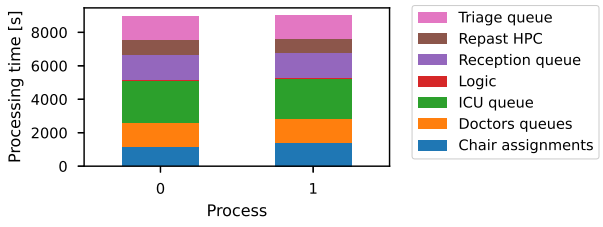


Fig. 9: Time distribution in two processes simulation in a simulated cluster. The experiment is conducted using two virtual machines, which communicate via virtual network.

VII. Conclusion and Future Work

Although no performance improvement was obtained, the research provides an overview of Repast HPC and the challenges that a distributed memory framework presents when implementing small simulations.

With respect to Repast HPC, it is difficult to justify the exclusive use of distributed memory with the abundance of multi-core processors. In particular, for this simulation, the high degree of mobility of the agents and the reduced space where they are moved has an important impact on the performance of the system.

A noteworthy point of the data obtained and useful for future work is the overhead in single-process execution. If it is considered that the framework takes $40\,\%$ of the execution time even when it has no communication with other instances, it can be assumed that using a framework with lower overhead and working on shared memory has the potential to considerably improve performance.

Due to the simplicity of the model, an alternative to increase performance is to implement a solution with no framework, using conventional APIs such as `pthreads` or OpenMP. Since Repast Simphony allows the usage of parallelism, it can be also be considered for future works.

References

[1] L. Maccallini, D. O. Encinas, and F. Romero, "An approach to the modeling and simulation of intra-hospital diseases," *Journal of Computer Science and Technology*, vol. 21, no. 2, p. e14, Oct. 2021. [Online]. Available: https://journal.info.unlp.edu.ar/JCST/article/view/1827

[2] "Departamento de estadísticas e información de salud de chile." last accessed 13 Feb 2022. [Online]. Available: https://deis.minsal.cl/#estadisticas

[3] "Estudio de las prevalencias de infecciones nosocomiales en españa (epine)," last accessed 13 Feb 2022. [Online]. Available: https://epine.es/api/documento-publico/2019%20EPINE%20Informe%20Espa%C3%B1a%2027112019.pdf/reports-esp

[4] "Módulos de principios de epidemiología para el control de enfermedades (mopece), segunda edición revisada, salud y enfermedad en la población, organización panamericana de la salud." last accessed 13 Feb 2022. [Online]. Available: https://www.paho.org/col/dmdocuments/MOPECE2.pdf

[5] N. Gomez Cruz, *Simulación basada en agentes: Una metodología para el estudio de sistemas complejos*. Colombia: Universidad de Antioquia, Jan. 2018, pp. 230–268.

[6] M. J. North, N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, p. 3, Mar 2013.

[7] N. Collier and M. North, "Parallel agent-based simulation with repast for high performance computing," *SIMULATION*, vol. 89, no. 10, pp. 1215–1235, 2013.

[8] K. P. Fennelly, "Particle sizes of infectious aerosols: implications for infection control," *The Lancet Respiratory Medicine*, vol. 8, no. 9, pp. 914–924, Sep. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2213260020303234

[9] W. Chen, N. Zhang, J. Wei, H.-L. Yen, and Y. Li, "Short-range airborne route dominates exposure of respiratory infection during close contact," *Building and Environment*, vol. 176, p. 106859, Jun. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360132320302183

[10] C. Blazejewski, M.-J. Guerry, S. Preau, A. Durocher, and S. Nseir, "New methods to clean ICU rooms," *Infectious Disorders Drug Targets*, vol. 11, no. 4, pp. 365–375, Aug. 2011.

[11] N. Huynh, P. Perez, M. Berryman, and J. Barthélemy, "Simulating transport and land use interdependencies for strategic urban planning—an agent based modelling approach," *Systems*, vol. 3, no. 4, pp. 177–210, 2015. [Online]. Available: https://www.mdpi.com/2079-8954/3/4/177

[12] C. M. Macal, N. T. Collier, J. Ozik, E. R. Tatara, and J. T. Murphy, "Chisim: An agent-based simulation model of social interactions in a large urban area," in *2018 Winter Simulation Conference (WSC)*, 2018, pp. 810–820.

[13] "Mpich installer's guide," Available at: https://www.mpich.org/static/downloads/4.0.2/mpich-4.0.2-installguide.pdf, last accessed 27 Jul 2022.