EasyChair Preprint

№ 4160

Epirust: Towards a Framework for Large-Scale Agent-Based Epidemiological Simulations Using Rust Language

Jayanta Kshirsagar, Akshay Dewan and Harshal Hayatnagarkar

September 9, 2020

EpiRust: Towards A Framework For Large-scale Agent-based Epidemiological Simulations Using Rust Language

Jayanta K. Kshirsagar Akshay Dewan Harshal G. Hayatnagarkar

Engineering for Research, ThoughtWorks Technologies, Pune, India,
{jayantak,akshayde,harshalh}@thoughtworks.com

Abstract

To implement large-scale agent-based simulations, developers historically relied on C and C++ due to performance, while struggling to deal with tedious explicit memory management. This struggle translates into software defects and lower developer productivity. More recently, desire to harness multi-core systems via concurrent software complicates design and implementation when memory is shared among compute cores. When we faced this situation, we were looking for a system programming language as fast as C and C++ but without caveats around memory management. Between Go and Rust, we chose Rust language which guarantees safety in memory management even for concurrency, without a run-time or garbage collector. In this paper, we have shared our experience with Rust to build a framework named as EpiRust for agent-based epidemiological simulations. Our simulation experiments have shown some promising results for a population of a few million agents, using commodity-class hardware. Key outcomes of this whole exercise are that EpiRust could achieve following quality goals: 1. Robustness 2. Flexibility 3. Performance.

Keywords: agent-based modeling, agent-based simulations, epidemiology, large-scale simulations, Rust language.

1 Introduction

Pandemics are seen as one of the top global threats, and public health policy experts need tools to assess preparedness and response strategies. Large-scale agent-based epidemiological simulations are one such useful tool in their repertoire (Bisset et al., 2009).

To implement such an epidemiological model, developers can either choose to develop using agent-based modeling platforms such as Netlogo (Tisue and Wilensky, 2004) and GAMA (Taillandier et al., 2019) or using general-purpose libraries and frameworks such as Repast, Flame, and MASON (Rousset et al., 2016).

Precursor to EpiRust was our experience of developing a SEIR model using the GAMA platform. We found that the platform is useful for rapid prototyping and quick feedback on the functionality but performs slower for our needs. The GAMA-based simulations were consuming more time for more number agents in a non-linear fashion. For example, a simulation with 10000 agents would take more than 60 minutes using 128 CPU cores on a large server. We were discouraged with these results as our goal is to simulate a large city like *Pune, India* with more than three million population. These observations triggered us to look for effective alternatives for large scale simulations.

After surveying the literature, we started development of a new framework for large scale epidemiological simulations built on three intrinsic qualities namely, *robustness*, *flexibility*, and *performance*, and we named it as EpiRust.

1.1 Related Work

Developing and maintaining such large-scale simulations is a complex exercise, especially due to a trade-off between flexibility and performance (Rousset et al., 2016; Fujimoto et al., 2017; Chen et al., 2017).

We observed that for large-scale agent-based simulations including epidemiology models, only a few frameworks are designed for this purpose (Abar et al., 2017; Chen et al., 2017). The two important challenges to scale up and out are: 1. Partitioning and distributing the problem across computers, 2. Coordinating the computers to ensure the solution is valid (Yeom et al., 2014).

To develop such distributed systems, choosing a programming language can play an important role in performance, quality and evolution of software. Historically in scientific computing, system programming languages such as C and C++ have been the preferred languages for the superior performance they offer. However, developers using these languages also have low productivity in terms of development time and software quality, and according to (Phipps, 1999), both of these issues emerge from complexity of explicit memory management. In our search for a programming language to develop large-scale epidemiological models, we were looking for the languages that should be fast, stable, with a mature ecosystem of tools and libraries.

C and C++ are designed for system programming which means they can directly interface with underlying hardware. However, Java, C#, and Python languages target respective virtualized runtimes to support hardware-agnostic execution, resulting in lower performance. Julia language (which is designed for scientific computing), and Go language (which is designed for distributed scal-

able applications) adequately fit the bet for large-scale epidemiology simulations. However, both of these languages must rely upon some system programming language to access hardware resources. From a performance point of view, (Kouatchou, 2018) compares many of these languages together to get a sense of their relative strengths and weaknesses.

Software written using C and C++ typically offers better performance however suffers from memory safety and robustness issues due to explicit memory management and concurrency. After evaluating these options, we finally had two choices, Go and Rust language. Between these two, we chose Rust language that guarantees safe memory management without a garbage collector, and the resultant code is safe for concurrency as well (Klabnik and Nichols, 2018; Balasubramanian et al., 2017). Based on similar reasons, a recent work (Antelmi et al., 2019) also has chosen the Rust to implement agent-based simulation models.

1.2 The Rust Programming Language

Rust is a multi-paradigm programming language, which offers features such as zero-cost abstractions, memory safety without garbage collection and run-time overheads, and fearless concurrency.

Rust has a unique approach based on "Ownership and Borrow" principles to free the developer from the burden of manual memory management, an Achilles' Heel for many developers from the perspective of memory safety and robustness in C and C++ (Balasubramanian et al., 2017). Rust maintains only one reference to the allocated memory eliminating the need of garbage collection and resulting in faster execution performance. All the variables being non-mutable by default, Rust compiler forces a developer to write smaller and cleaner code. For flexibility and testability, Rust prefers composition over inheritance via traits.

In this paper, we have shared our rationale and learning of implementing a large-scale epidemiological simulation scaling up to a population of a few million agents and capable of running on commodity-class computer hardware in reasonable time. In section 2 discusses the current EpiRust model including population, geography, disease dynamics and interventions. In the following section 3, we have discussed the features of the Rust language, their relevance in the EpiRust implementation, and later the results of simulating baseline and lockdown scenarios for the city of Pune. Finally, we have concluded the paper with our observations and plans for the EpiRust framework.

2 EpiRust Model

The EpiRust model follows a minimalist approach for modeling an epidemic in a virtual society along three main aspects namely *population*, *geography*, and *disease*.

2.1 Population

An agent represents a person in a population. Agents are heterogeneous along two attributes namely *work status* (via *working* and *non-working*) and *transport preference* for office commute (via *public transport* and *private transport*). A fraction of the working population can be configured as the *essential workers* whose services are available even when the other agents have restrictions on their movements. Each agent follows a daily routine, prescribed by its demographics and other attributes, as shown in table 1. An agent could catch and spread infection while interacting with other agents based on how contact probabilities change due to their routine.

2.2 Geography

Factors such as geography, demographics, and transportation influence the spread and the speed of epidemics. The EpiRust model mimics geography of a minimalist city and represents it as a *grid* with various planned functional areas for homes, workplaces and transportation.

- **Home Area** Every agent is randomly assigned a home location during initialization. Collectively, these home locations represent a residential colony. Working agents commute to-and-fro their workplaces, whereas non-working agent movements are restricted to this area.
- **Public Transport Area** is used by working agents for home and work commute. Currently, the model assumes that agents using private transport reach directly to homes and workplaces.
- **Work Area** During work hours, a working agent occupies a dedicated work location in this area with restricted movements.

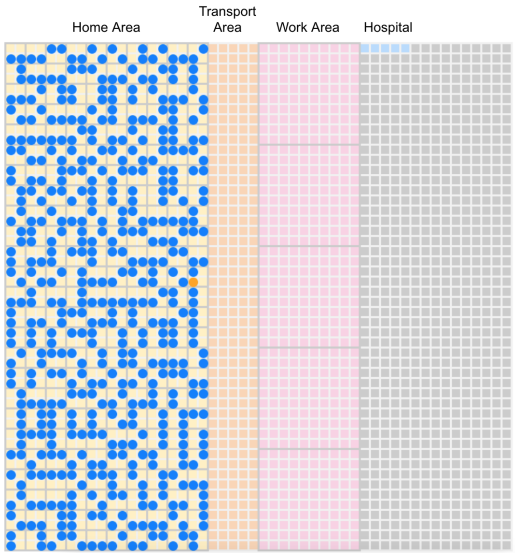


Figure 1. Grid-based Geography Depiction

Start Hour	End Hour	Start Place	End Place	Activity
0	8	Home	Home	No movement (Sleep, etc.)
8	9	Home	Home	Activity within home
9	10	Home	Transport	Go to public transport
10	11	Transport	Transport	Move within transport
11	12	Transport	Work	Go to work
12	19	Work	Work	Move within work
19	20	Work	Transport	Go to public transport
20	21	Transport	Home	Go to home
21	23	Home	Home	Activity within home

Table 1. 24-hour Routine of Working Agents

2.3 Disease Dynamics

The current disease dynamics is based on the Mordecai SEIR model (Childs et al., 2020), and is specialized for COVID-19 similar to (Snehal Shekatkar et al., 2020). The model is shown in figure 2, and the disease progression over time is rendered in figure 3.

1. **Susceptible** All agents are initialized in the susceptible compartment.

2. **Exposed** A susceptible agent could get exposed to an infection if it comes in contact with an infectious agent. An exposed agent does not infect other susceptible agents immediately. It stays in this state for an incubation period specific to a disease. After this period, the agent state would switch to the infected compartment. While initializing the simulation, a fraction of the population can be configured as being in the exposed compartment.

3. **Infected** In this state, an agent could infect other susceptible agents with some probability. This probability is specific to a disease, and can be configured by a user. An infected agent could either be symptomatic or asymptomatic. Symptomatic agents could later be mild or severely infected. On the other hand, the asymptomatic agents would not exhibit any symptoms but still could infect others.

4. **Removed** The recovered and the deceased agents would belong to this state. A SEIR model assumes that the recovered agents would not catch an infection again in their lifetime. The death rate is an input parameter to decide an agent's recovery.

2.4 Interventions

Interventions play a key role in devising policies to reduce and eventually to stop the spread of an epidemic. An intervention can be specific to demographic groups within a population or their actions. EpiRust has support for three interventions namely isolation via *hospitalization*, *lockdown* of the city, and *mass vaccination*.

- **Hospitalization** intervention helps to isolate and quarantine agents with severe infection. Hospitalization is limited by the number of hospital beds, but this limit can be specified via configuration parameters.

- **Lockdown** would restrict citizen movements. Agents would be confined in their homes and only essential workers would be allowed to travel.

- **Mass Vaccination** strategy vaccinates user-defined percentage of randomly chosen susceptible agents. Once vaccinated, these agents would never catch disease in their lifetime. This intervention can be triggered by a configurable threshold.

2.5 Simulation Loop

At the heart of the simulation, there are two simple loops: The outer one controls the discrete time steps (or clock), and the inner loop updates the state of each agent at each time step.

```
for hour in 1..n do
    if can_intervene then
        Apply lockdown;
    end
    for each agent do
        Move agent on the grid;
        Update infection state;
    end
    if number_of_infected == 0 then
        Stop simulation;
    end
end
```

Algorithm 1: Simplified Simulation Loop

Here, updating an agent after another in a sequential execution leads to path dependent outcomes. To solve this issue, EpiRust uses a double buffering technique (Cosenza et al., 2018; de Aledo Marugán et al., 2018). The technique employs a read-only buffer for referring to the current state of agents, and a write-only buffer for updating their next state.

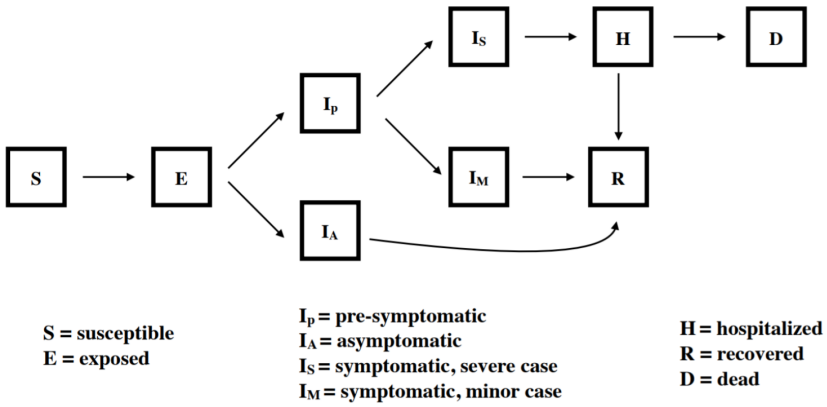


Figure 2. The Mordecai SEIR Model (Childs et al., 2020)

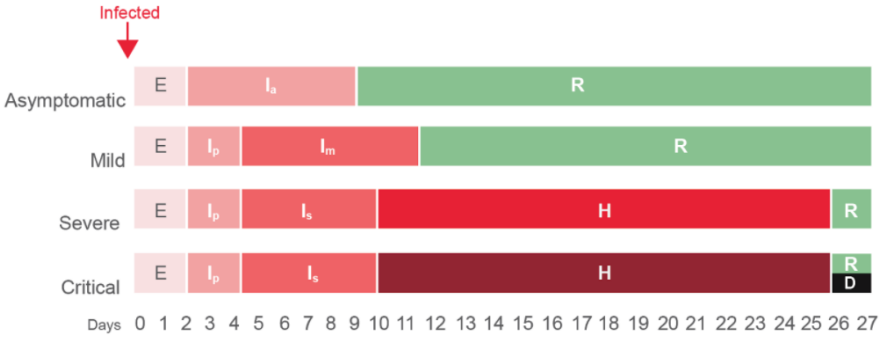


Figure 3. Disease Dynamics (Snehal Shekatkar et al., 2020)

3 EpiRust Implementation

As we discussed earlier, EpiRust has been designed with three goals: Robustness, flexibility, and performance. Rust contributes to robustness of memory management and concurrent access via its features of borrow checking and lifetime management (Klabnik and Nichols, 2018). In the rest of this section, we have discussed the implementation towards flexibility and performance goals.

3.1 System Requirements

3.1.1 Model Correctness

To verify the model, a basic grid visualization is developed to observe the agents during each time step of the simulation. Along with the grid visualization, we followed *test-driven development* (*TDD* for short) to detect defects and similar regression caused by code changes. TDD is part of *Agile Software Development* practices, where a developer writes unit test cases before writing the functional code, and runs these test cases after implementing a change (Beck, 2003). This approach helps in verifying functional correctness, especially in case of the frequent code changes. Cargo (Klabnik and Nichols, 2018), Rust's build and package manager gives an ability to run unit tests and performance benchmark tests. Unit tests rely upon *Mocks* which mimics structure and behavior of code beyond the scope of the current unit. We found that creating *mocks* for Rust structures is not convenient for developers.

3.1.2 Flexibility

A model is flexible if a user can simulate different variations of the simulation model, either by specifying configuration parameters or disease-specific model extensions.

EpiRust uses the Rust constructs such as *mod*ules for modularity, *struct*ures for entity modeling, *trait*s to define behavioral contracts for later composition with entities (something similar to inheritance in other languages). These constructs have contributed to separate disease dynamics from other components. This separation enables the necessary *flexibility* to attach any SEIR-based disease dynamics to the simulation without significant changes in the rest of the source code. A user can alter disease dynamics by altering configuration parameters for not only COVID19 but also other SEIR diseases such as smallpox. For example, to model smallpox disease dynamics, a user can choose to not specify asymptomatic and mild symptomatic transmission rates in the configuration parameters.

The user can simulate multiple scenarios without changing source code via configurable input parameters. A subset of these parameters are discussed below.

General Parameters

- **grid_size** size of the square grid to represent the city.

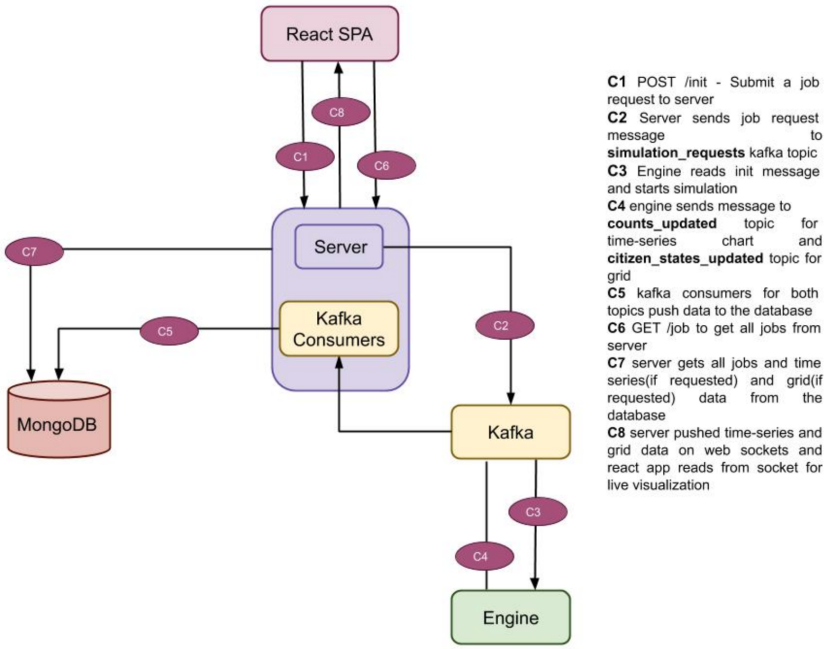


Figure 4. System Architecture

- **number_of_agents** the population in the city.
- **working_percentage** the percentage of working population to total population.
- **public_transport_percentage** the percentage of the population taking public transport to work.
- **hospital_beds_percentage** the number of hospital beds per 100 people.
- **starting_infections** the seed infected population.

Disease Dynamics Parameters

- **percentage_asymptomatic_population** specifies the percentage of population will not show any symptoms when infected.
- **percentage_severe_infected_population** Fraction of the infected population with severe symptoms would need hospitalization.
- **transmission_start_day** Day after catching an infection when an infected person may spread the disease.
- **transmission_rate** is the probability of a susceptible agent catching an infection when contacted by an infected agent.
- **asymptomatic_last_day** is the last day for an asymptomatic agent to recover completely from an infection.
- **last_day** is the last day of the disease when an agent either recovers completely or dies of a severe infection.

Intervention Parameters

- **Lockdown.at_number_of_infections** triggers the lockdown intervention when the total number of infected agents matches to the parameter value.
- **Lockdown.essential_workers** is the fraction of the working population which would perform their work routine even during lockdown.
- **Vaccination.at_hour** is the time step to trigger vaccination.
- **Vaccination.percent** is a fraction of the population to be randomly vaccinated.

The EpiRust disease dynamics model works as a finite state machine (FSM) with the states as the SEIR compartments. For state transitions, FSM first matches the current state and then moves to the next state. To ease this implementation, Rust offers a construct *Pattern* for matching numbers, strings, *enum*s, and regular expressions, etc.

3.1.3 Scalability

While representing millions of agents with their movements and disease states along with the geographical details CPU and memory consumption are two important factors to optimize. Choosing appropriate data structures and algorithms can help to optimally manage these resources. For example, when the model used a dense matrix for representing the grid-based geography for agent locations, it consumed a large amount of memory throughout the simulation for even smaller agent populations. An efficient alternative is spatial hashing technique which

stores only the locations where agents are placed. Switching from the dense matrix to the spatial hashing reduced memory usage to a fraction of its original.

3.1.4 Performance

While spatial hashing helped to reduce memory usage, it kept CPU time consumption at the same level of dense matrix. For the hashmap data structure, the average-case time complexity for retrieval and insertion is $O(1)$. After evaluating multiple implementations of hashmap data structure in the Rust ecosystem, EpiRust uses *FNVHash* (Crichton, 2020) for better performance. In addition, we observed that the random number generator supplied by the Rust standard library is adequate to avoid collisions during agent placement.

3.2 Architecture

EpiRust has multiple smaller components as depicted in figure 4. **Engine** is the simulation engine written in *Rust* and discussed in this paper. The engine is invoked via command line inputs including configuration files in the JSON format. **Server** is a web server and the central component which interfaces with other components such as the engine and the user interface. It invokes the engine using its command line interface. It sends simulation requests to the engine and reads the engine output. The output is later stored in a database so as to consume it later. The server uses *Apache Kafka* for communication with the engine. Kafka is a distributed streaming platform used here for mediation between the server and the engine. **MongoDB** stores simulation-generated data for later consumption for visualization and debugging. **React SPA** is the web user interface to specify model parameters and visualize the simulation results. It has been developed as a single page application using *React JS*.

4 Scenarios and Results

When EpiRust development had started, COVID-19 had begun to unfold in *Wuhan, China*. In order to get a better understanding of COVID-19 spread in real life, we decided to run it for the city of *Pune, India*. The city is spread over an area approximately *331 square km* in the Western India, and has a population of more than three million people.

To represent Pune city, we provided the following configuration parameters as rendered in table 2.

To get a better understanding of the impact of various non-pharmaceutical interventions on spread of a disease, we simulated two different scenarios namely *Baseline* and *Intervention*. As a model is probabilistic, we ran each of these scenarios 5 times to ensure statistical stability.

4.1 Baseline Scenario

The baseline scenario depicts business as usual even in a pandemic scenario. All agents will follow their routing regardless of the disease spread and all of the interventions like lockdown, increasing hospital bed capacity, vaccination etc. would not be in place.

Parameter	Value
number_of_agents	3137224
grid_size	5660
working_percentage	70%
public_transport_percentage	80%
Lockdown_at_number_of_infections	100
essential_workers_population	10%
hospital_beds_percentage	0.003%
starting_infections	32645

Table 2. Input Parameters

4.2 Intervention Scenario

Lockdown can be a good intervention strategy but has its own costs. To balance cost versus benefits requires trying various what-if scenarios. EpiRust lockdown intervention can be triggered for a configurable threshold of number of infections. During lockdown intervention, the essential workers would still be working and performing their daily routine as discussed in 2.1, and could potentially acquire and spread infections. A fraction of the population could choose to defy the lockdown intervention. This non-compliant population could be modeled by adding their percentage to the essential workers. Mass vaccination as an intervention has been turned off for these simulations.

4.3 Results

We used Rust compiler version 1.44.1 for all these benchmarks, and each scenario of this stochastic model has been run for at least five times to ensure statistical stability. So, the epicurves charts 5, 6(a) and 6(b) render mean and variance over time.

Compared to a similar model implemented using the GAMA platform, a single-threaded EpiRust simulation now completes within 30 seconds (a speedup of more than 100x). For Pune city simulations with more than 3 million population, EpiRust takes close to 150 minutes on single core of commodity class hardware for baseline scenario and 50 minutes for lockdown scenario.

For every agent in the simulation, the number of interactions with neighboring agents and its associated time complexity are order of $O(n^3)$. We observed that the Rust's *HashMap* based implementation has a smaller memory footprint, just over twenty megabytes for a million agents, and hence computer memory is less of a concern here. The performance across processors varies due to clock frequency as well as internal architecture of the processor.

While comparing multiple simulation scenarios for Pune city, we found that lockdown is an effective strategy to reduce the total infections in the society. Based on our observations (figures 6(a), 6(b)) lockdown slows down the rate at which infections grow and imposing lockdown

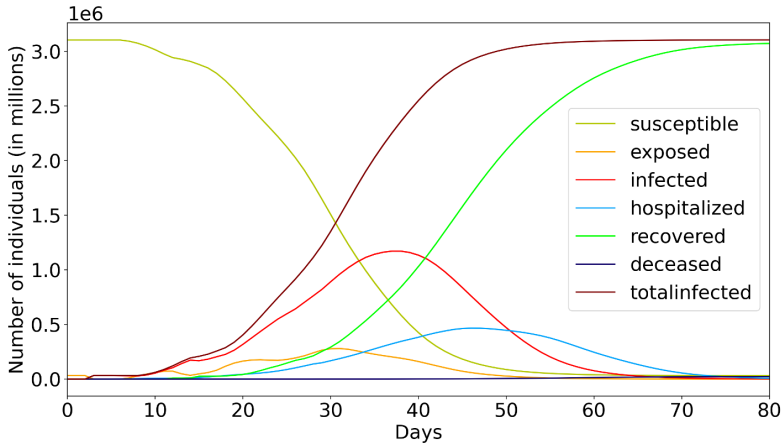
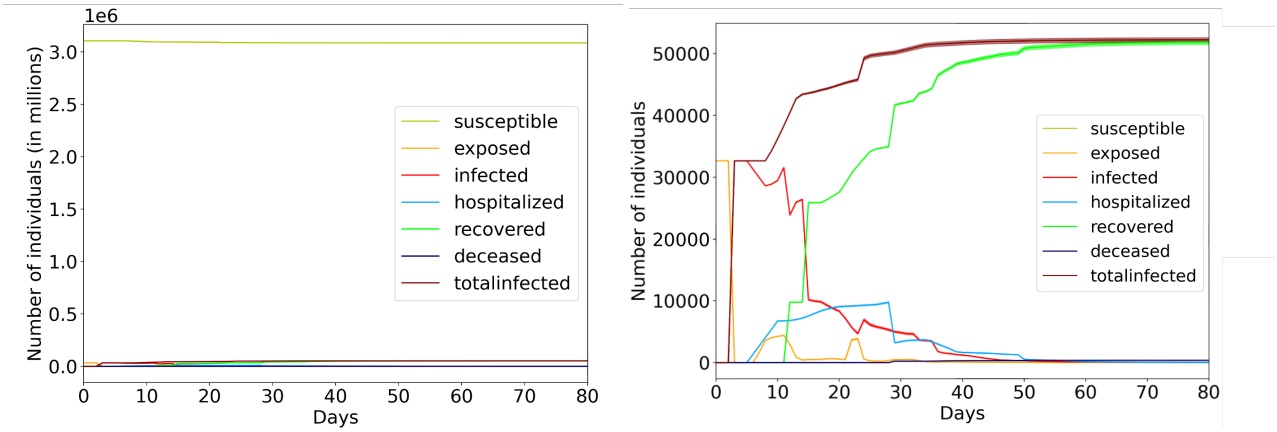


Figure 5. Pune Baseline Scenario



(a) Lockdown

(b) Lockdown (without visualizing Susceptible Population)

Figure 6. Lockdown Scenarios for Pune Simulations

reduces stress upon health care facilities like hospitals.

4.4 Reproducing the results

EpiRust source code is released in open source. To reproduce results of this paper, the EpiRust framework and the model used in this paper can be found here: `https://github.com/thoughtworks/epirust/tree/sims2020`. In cloned Git repository, the configuration files used for baseline and lockdown scenarios can be found at `epirust/engine/config/high_baseline.json` and `epirust/engine/config/high_lockdown.json` respectively.

5 Conclusion and Future Work

Based on our current exploration, we found Rust useful in writing large-scale epidemiological models, even without employing some advanced features such as parallelism and concurrency. Rust's preference for composition over inheritance helps in building flexible models. We found that Rust has a steeper learning curve than other programming languages, especially to interpret the compiler errors around borrow-checking. However, we trust that once compiled the generated executable is robust and guaranteed to be memory safe. As a bonus, we found the ownership principle guarantees efficient and memory safe code even for concurrency. The Rust package manager *Cargo* handles library dependencies and their installation transparently. Rust's borrow checker expects the programmer to specify the lifetime of the function parameters to check the reference validity. This can make writing extensible code to be tricky. We observed that writing unit tests for complex nested structures is difficult due to limited support for mocking.

Using the Rust and its ecosystem, we believe our success is moderate in achieving our goals of performance, flexibility, and robustness qualities. At the same time, this lays the foundation of our future work on this framework. For example, to improve performance, we would like to harness multiple cores across computers. We also intend to explore features for flexibility around dynamic and user-defined disease models. We acknowledge the impact of demographics on simulation results, and will work towards supporting synthetic population. To ease setup and deployment, we are working on a Docker image and

will be shared soon.

6 Acknowledgements

The authors would like to thank Dr. Gautam Menon for his guidance, especially for the Mordecai SEIR Disease Model. The authors would like to acknowledge and thank all, who contributed to the EpiRust development including the ThoughtWorks EpiRust team.

References

Sameera Abar, Georgios K Theodoropoulos, Pierre Lemarinier, and Gregory MP O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33, 2017.

Alessia Antelmi, Gennaro Cordasco, Matteo D'Auria, Daniele De Vinco, Alberto Negro, and Carmine Spagnuolo. On Evaluating Rust as a Programming Language for the Future of Massive Agent-Based Simulations. In *Asian Simulation Conference*, pages 15–28. Springer, 2019.

Abhiram Balasubramanian, Marek S. Baranowski, Anton Burtsev, Aurojit Panda, Zvonimir Rakamaric, and Leonid Ryzhyk. System programming in rust: Beyond safety. *Operating Systems Review*, 51:94–99, 2017.

Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

Keith R Bisset, Xizhou Feng, Madhav Marathe, and Shrirang Yardi. Modeling interaction between individuals, social networks and public policy to support public health epidemiology. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 2020–2031. IEEE, 2009.

Jiangzhuo Chen, Bryan Lewis, Achla Marathe, Madhav Marathe, Samarth Swarup, and Anil KS Vullikanti. Individual and collective behavior in public health epidemiology. In *Handbook of statistics*, volume 36, pages 329–365. Elsevier, 2017.

M. L. Childs, M. Kain, D. Kirk, M. Harris, L. Couper, N. Nova, I. Delwel, J. Ritchie, and E. Mordecai. The impact of long-term non-pharmaceutical interventions on covid-19 epidemic dynamics and control. *medRxiv*, 2020.

Biagio Cosenza, Nikita Popov, Ben H. H. Juurlink, Paul Richmond, Mozhgan Kabiri Chimeh, Carmine Spagnuolo, Gennaro Cordasco, and Vittorio Scarano. Openabl: A domain-specific language for parallel and distributed agent-based simulations. In *Euro-Par*, 2018.

Alex Crichton. fnv hash. `https://crates.io/crates/fnv/1.0.7`, 2020. URL `https://crates.io/crates/fnv/1.0.7`.

Pablo González de Aledo Marugán, Andrey Vladimirov, Marco Manca, Jerry Baugh, Ryo Asai, Marcus Kaiser, and Roman Bauer. An optimization approach for agent-based computational models of biological development. *Adv. Eng. Softw.*, 121:262–275, 2018.

Richard M Fujimoto, Christopher Carothers, Alois Ferscha, David Jefferson, Margaret Loper, Madhav Marathe, and Simon JE Taylor. Computational challenges in modeling & simulation of complex systems. In *2017 Winter Simulation Conference (WSC)*, pages 431–445. IEEE, 2017.

Steve Klabnik and Carol Nichols. *The Rust Programming Language*. Mozilla Research, 2018.

J Kouatchou. Basic comparison of python, julia, matlab, idl and java (2018 edition). *Modeling Guru–National Aeronautics and Space Administration, USA*, 2018.

Geoffrey Phipps. Comparing observed bug and productivity rates for Java and C++. *Software: Practice and Experience*, 29(4):345–358, April 1999. ISSN 0038-0644, 1097-024X.

Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review*, 22:27–46, November 2016. ISSN 15740137.

Snehal Shekatkar, Bhalchandra Pujari, Mihir Arjunwadkar, Dhiraj Kumar Hazra, Pinaki Chaudhuri, Sitabhra Sinha, Gautam I Menon, Anupama Sharma, and Vishwesha Guttal. Indscisim a state-level epidemiological model for india, 2020. Ongoing Study at `https://indscicov.in/indscisim`.

Patrick Taillandier, Arnaud Grignard, Nicolas Marilleau, Damien Philippon, Quang-Nghi Huynh, Benoit Gaudou, and Alexis Drogoul. Participatory modeling and simulation with the gama platform. *Journal of Artificial Societies and Social Simulation*, 22(2):3, 2019. ISSN 1460-7425. URL `l`.

Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004.

Jae-Seung Yeom, Abhinav Bhatele, Keith R. Bisset, Eric J. Bohm, Abhishek Gupta, Laxmikant V. Kalé, Madhav V. Marathe, Dimitrios S. Nikolopoulos, Martin Schulz, and Lukasz Wesolowski. Overcoming the scalability challenges of epidemic simulations on blue waters. *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 755–764, 2014.