Interactive Visualization of Agent-Based Pandemic Simulation in Web Browser

Michal Mrozek
Warsaw Univ. of Techn.
Warsaw, Poland
michal.mrozek2.stud@pw.edu.pl

Mikhail Sirenko
Delft Univ. of Techn.
Delft, the Netherlands
m.sirenko@tudelft.nl

Anna Foks-Ryznar
Space Research Centre (CBK PAN)
Warsaw, Poland
afoks@cbk.waw.pl

Bartosz Sawicki
Warsaw Univ. of Techn.
Warsaw, Poland
bartosz.sawicki@pw.edu.pl

2022 23rd International Conference on Computational Problems of Electrical Engineering (CPEE) | 979-8-3503-9625-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CPEE56060.2022.9919642

***Abstract*—The paper presents a web-based system designed to visualize a simulation of ca. 500,000 residents of a city during the COVID-19 outbreak. Each agent's infection status and mobility are presented in the time scale. The main effort was to ensure simplicity from the user's perspective and smooth operation on various client devices. The full range of software engineering problems is discussed, from efficient data storage to optimization of visualization methods.**

***Index Terms*—agent-based model, web application, covid-19 outbreak, big data**

I. Introduction

Data presentation is one of the main challenges of the *big data* era. Robust information systems collect and model real-world phenomena. However, if humans make decisions based on this, the data must be easily accessible and understandable. Web browsers are a common platform for IT systems to communicate with users. At the same time, web-based applications require meticulous design due to their multi-level and distributed nature.

The COVID-19 pandemic was a considerable threat that stimulated many scientific teams into action. One trend was geographically accurate modelling of the spread of the virus. An issue naturally associated with this type of problem is the presentation of the model results. It could be classified as a computer science challenge in software engineering.

A simple solution to this problem has been presented by the system CoronaModel developed by the University of Hohenheim [1]. It is also based on simulation data and allows changing the model's parameters in real-time. Since the data set includes less than 500 agents, it is possible to download all data at the start of the application. The map is static, i.e., it does not support zooming or panning. The authors created the visualization by directly drawing on an HTML canvas element.

A comprehensive list of 121 similar applications that visualize data from the COVID-19 pandemic has been created by Bernasconi and Grandi [2]. Their work reveals that none of the systems supports the one person (agent) level of detail. The aggregated data for provinces, states or entire countries are presented and animated.

This paper describes research for a system that interactively visualizes a time-varying infection situation at the level of individual virtual persons (agents). The developed open-source system has unique features that allow for animating large amounts of data in a web browser.

II. Agent-based model

Agent-based modelling (ABM) is a simulation modelling paradigm along with system dynamics (SD) and discrete-event simulation (DEVS). In its current form, ABM has originated from the work on segregation [3]. From there, it has spread across various disciplines, including epidemiology [4]. At the core of ABM lies the concept of emergent phenomenon (e.g. virus spread) generated from the interaction-behaviour of individual agents (e.g. people). Such a focus allowed scholars to take ABM on board for modelling the COVID-19 spread (see, e.g. [5]). The system of interest, in this case, can be a city, and agents represent its residents. Besides, such models typically have other city subsystems: locations that residents visit (schools, supermarkets), activities that they do (work, leisure), transport that they use (buses, trams) and epidemiology status (SEIR - Susceptible, Exposed, Infected, Recovered).

For this study, we use the HERoS model [6], which is an extension of the influenza model in Beijing [7]. As the case study, we select the city of The Hague, the Netherlands. We create a synthetic population of the city of 540,000 residents-agents from aggregated open data [8]. The resulting agents have a set of key attributes such as a home address, age, household size and occupation. Agent's activities depend on their social role. The Time Use Survey reports the average duration and frequency of different activities performed by an average Dutch person across a week. Based on this, we generate 10 schedules, one for each social group, for each day of the week. These schedules are combinations of essential activities, e.g. sleep, work, shopping, going out etc. Agent executes activities at the places of interest (POI) within data from OpenStreetMap. Depending on the activity, an agent visits one of 200,000 locations on a map which fall under different categories such as home, education, shopping, restaurants etc. To simulate the disease we use a modified SEIR-like model that accounts for symptomatic and asymptomatic transmission. The simulation model works on an open-source engine DSOL [9]. Under the hood, it uses a mix of DEVS and ABM simulation modelling paradigms. Such a mix allows agents to execute activities only at a given time and not every time tick. Therefore, the model is much faster than a conventional ABM: a model run of 2 months with 540,000 agents takes ≈ 1.5 hours on an average laptop.

The key model outcomes are the number of agents in each of the 12 states: susceptible, exposed, etc. These outcomes have two dimensions: temporal and spatial. Therefore, we are interested in visualizing agents over time and on the map.

III. Visualization Requirements

The results of modelling were initially presented on an interactive web-based dashboard to visualize on graphs and maps the number and location of active COVID-19 cases, hospitalizations, intensive care unit admissions, deaths and recoveries over time. The interactive maps allow the user to compare the pandemic situation on selected dates and in two data aggregation modes, i.e. in city districts and neighborhoods. However, only an animated visualization of individual agents moving around the city can provide an idea of the magnitude and rate of virus spreading, as well as highlight the locations where transmission of the virus spreads more easily.

The interactive animation combined with a web-based application provide the public health authorities with an useful and easy accessible tool to visualize and explore possible impacts of the outbreak as it can unfold. This approach has the advantages of giving a dynamic perspective, accessibility to wide audience and interactivity.

The system, as a web-based GIS application, should meet the requirements related to effective map design and ergonomics of human-system interaction. In this domain the usability, which is associated with such attributes as effectiveness, efficiency and satisfaction [10], can been achieved by i.a.:

- simplicity and clarity – a map should be easy to read in all available scales within the zooming feature, the interactive elements and settings should be targeted,
- purposefulness – the symbology and colours should be meaningful and intuitive (consistent with the map message).

IV. System Architecture

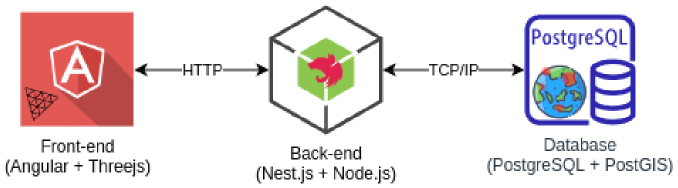

Fig. 1. System architecture diagram

The developed system is a classical web application, as presented on Fig. 1. The solution is hosted on server that has the Node.js environment and the PostgreSQL database with the PostGiS extension installed. For ordinary users focused on the analysis of the simulation model, the application is available from a web browser and does not require the installation of any special extensions or software.

The system is built of three main components: front-end, back-end and database. They share one code repository[1]. Since the client application should be run by a browser, the obvious choice is to use the JavaScript programming language. However, its basic feature of loose typing generates problems when developing larger projects. Fortunately there is a superset language called TypeScript, which introduces strict code typing into the syntax.

Visualization is provided by the Three.js library, which uses a low-level WebGL API specifically designed to display graphics in a web browser. Due to its popularity and modularity, many extensions have been created to solve common problems when developing web applications. In the developed system, maps are powered by MapLibre, a community fork of the popular and once open-sourced Mapbox, a library that allows to add aesthetic and functional maps to an application. The connection between a map and Three.js is achieved with Three-box plugin, which automatically synchronizes cameras and converts between geographic and Cartesian coordinate systems.

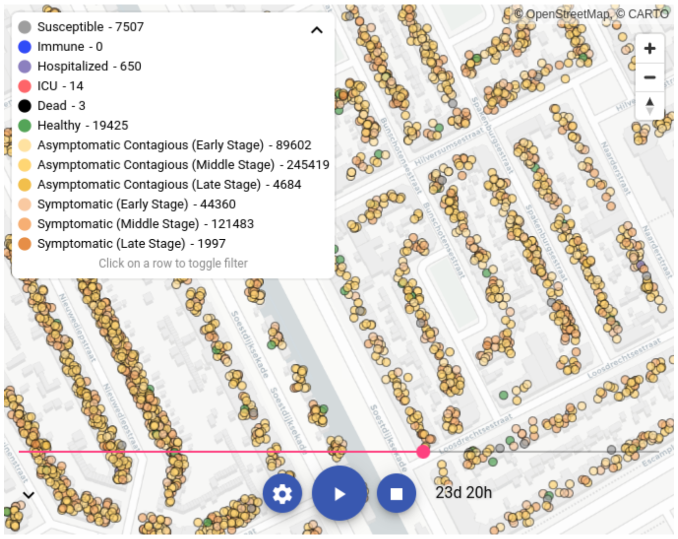
● Susceptible - 7507
● Immune - 0
● Hospitalized - 650
● ICU - 14
● Dead - 3
● Healthy - 19425
● Asymptomatic Contagious (Early Stage) - 89602
● Asymptomatic Contagious (Middle Stage) - 245419
● Asymptomatic Contagious (Late Stage) - 4684
● Symptomatic (Early Stage) - 44360
● Symptomatic (Middle Stage) - 121483
● Symptomatic (Late Stage) - 1997
Click on a row to toggle filter

Fig. 2. Screenshot of front-end web-based application

As per the requirements, the front-end application has been designed with the simplest possible user experience in mind. Fig. 2 presents screenshot of an interactive map limited geographically to the region where the simulation was performed is displayed on the whole screen. Zooming in, zooming out, or changing position is possible using the mouse. The upper left part presents a legend with the agents' disease statistics, which are updated during simulation time. Filtering by the disease phase can be selected for visualization by clicking on its label. At the bottom a progress bar allows to change the current animation time. Four buttons have been placed below,

[1]The code has been open-sourced and is available at Github: https://github.com/Michsior14/covis.

minimize progress bar, open settings, start/stop animation, and reset animation to initial state.

V. Serving data

The system's greatest challenge is processing massive amounts of data. The 180-day simulation tracks the hourly location and properties of 534,638 agents. As one can easily count this is over 2 billion 312 million records. The transfer medium between the HERoS model and the visualization system is a 350 GB compressed csv file. Working with that much information at once in a browser is nearly impossible using today's technology. No client machine is able to keep in memory a few hundred gigabytes and additionally animate the movement of agents on the map. So only the part of the data needed at a given moment is displayed. Two parameters are taken to effectively limit the amount of data: location and simulation time. However, filtering the csv file is also not among the fastest solutions. Consequently, we need to use a database filtering data using its latitude and longitude. Two databases were choosen to be analysed [11], [12] to meet the requirements: an open source object relational PostgreSQL with the PostGiS extension and document based NoSQL datastore MongoDB. Both solutions are scalable and popular for geospatial data.

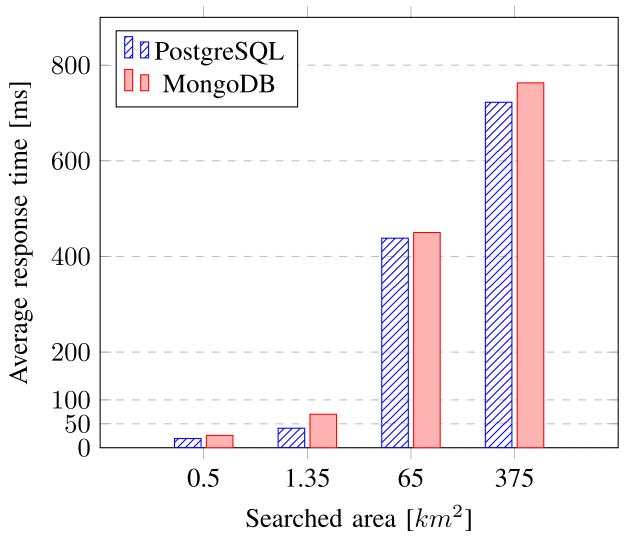

Fig. 3. Average response time of PostgreSQL and MongoDB

In order to determine which solution performs better in the described use-case, tests were performed on a portion of the data consisting of 1 million records. The official Docker containers of MongoDB version 5.0.6 and PostgreSQL version 14.2 with PostGiS version 3.2.1 were used as the test environment. The tests were performed after a warm-up phase, which consisted of executing various queries based on the created indexes. This ensures that more realistic results are returned, as filling the cache influences the speed of subsequent queries. Fig. 3 illustrates the average response times from 10 trials on both databases. As it can be observed PostgreSQL performed slightly better for all area sizes. For this reason, and by deciding to use a separate table to store agent properties (to reduce overall size of the database), PostgreSQL along with PostGiS was chosen.

Although it seems to be quite a simple task, importing data to the database is another a challenge. The data from HERoS is stored in a CVS file, which needs to be processed during the migration to fit the database scheme. Additionally, data even chunked might not fit in the program's memory if it hasn't been enough allocated beforehand. A simple import using the built-in migrations in TypeORM while handy during development turned out to be relatively slow. For 1 million records such a process takes in average 3.5 minutes. To speed things up an SQL script was created using native PostgreSQL functions. It first reads the entire file using the COPY FROM command to a temporary table, and then distributes the records to target tables changing at the same time the values to the appropriate format. Only after they are completely moved, indexes are created and planner is updated to determine the most efficient way to execute a query in the future. On the same 1 million data set as before this approach is up 45% faster and takes in average 2 minutes. However, it also has its drawbacks, such as it requires 2 times more disk space during import compared to the first solution because of the temporary table.

VI. Visualization performance

Dynamic control of levels of detail were introduced to manage large scale of date and improve clarity of presentation. Depending on the hardware of the user's device it is possible to select low, medium and high level of detail. The map zoom was correlated with the maximum number of agents returned by the database. This was implemented using a modulo function called on the agent ID. Chosen values are shown in Table I, for instance if the zoom equals 15 and detail level is set to low then the agent is returned only if agent ID % 200 = 0. The modulo divisors assigned to the zoom were defined manually based on number of agents in the system and observations of phone, tablet and computer performance while visualizing different city neighborhoods. This approach ensures that the filtering will be stable and at subsequent simulation hours we will receive the same units as long as they are still in the searched area.

TABLE I
Divisors for map zoom and level of detail for total number of 500,000 agents

Map zoom level (n)	Divisor for the detail level		
	low	medium	high
$n < 12$	1500	1000	500
$12 \leq n < 13$	600	500	250
$13 \leq n < 15$	400	300	150
$15 \leq n < 16$	200	100	50
$16 \leq n$	5	2	1

It is common in simulation data for different agents to be in exactly the same geographic location at one time. In extreme cases, there may even be a few thousand agents in the same position. This situation makes it hard to analyze the results and look for outbreaks. As a remedy, three different strategies for displaying points on the map has been studied. The simplest, Normal strategy displays the positions exactly as declared. The Random strategy adds a random offset to the coordinates of every point. Unfortunately, by using the built-in Math.random() function, the agents move to a random location at each successive simulation hour, even though the real position does not change. This strongly affects animation performance, which drops almost by half on average. The Hashed strategy (selected by default) is an improved version of the Random strategy. By using the mulberry32 pseudo-random number algorithm [13] created by T. Ettinger, the aforementioned effect of agents moving between consecutive time moments is eliminated.

Fig. 4 shows the distribution of device performance in one of Hague's higher density residential areas for different levels of detail and location strategies.

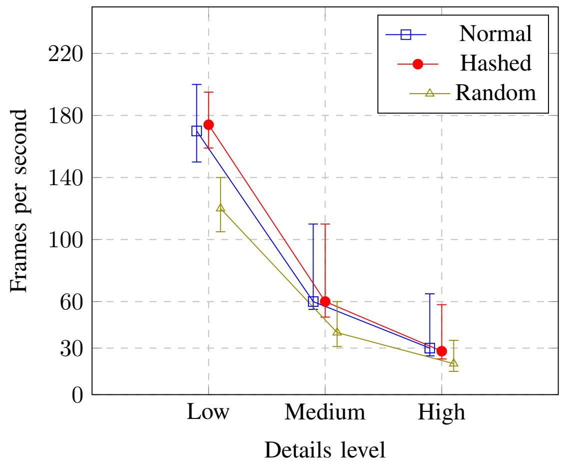

Fig. 4. Visualization performance in correlation to details level and points strategy in dense residential area (280 Hz display)

To enable fast animation of points, data are pre-loaded forward, i.e., n hours are loaded at a time for each hour of animation. This is particularly important for animation speeds of less than 1 second. Assuming that the average server response for an area larger than a square kilometer is about 300ms under ideal conditions then an animation rate of half a second may begin to stutter. If the speed of 1/4 second would be chosen, then for sure one would have to wait for loading the data from the next hour, because the animation would be executed faster than the data were arrived. In the advanced settings section, the user can select between 1 and 10 hours to be pre-loaded. For animations faster than a second, this value automatically changes to 5 if it was lower.

To optimize memory usage, the application uses a limited number of materials representing an agents. There are exactly as many materials as phases of the disease and they are shared by entities. One material occupies only about 3.6 KB of memory. Each of the materials is declared using GL shaders, giving full control over how the agents are drawn directly by the GPU. Thanks to this, it is possible to effectively and easily visualize an agent as a circle with a border and additional transparency without significant degradation of animation performance.

VII. Conclusions

The developed system allows to effectively present the behavior and health status of 500,000 residents of the city in a web browser environment. It was achieved by combining mechanisms such as: automatic change of detail level, special GPU shaders, application of hash functions for agent spread. Processing of over 2 billions of data records were the main challenge. However experiments have shown that modern database engines with GIS extensions could efficiently process in a such scale of the problem.

As a future direction, a direct connection between the visualization system and software that models resident behavior would be beneficial. This would reduce the problem of data transmission and alignment, and provide the opportunity to dynamically change the parameters of the influenza model.

Acknowledgment

This work has received support from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101003606 (HERoS).

References

[1] B. Vermeulen, M. Müller, and A. Pyka, "Social network metric-based interventions? experiments with an agent-based model of the covid-19 pandemic in a metropolitan region," *Journal of Artificial Societies and Social Simulation*, vol. 24, no. 3, p. 6, 2021. Application available at: https://inno.uni-hohenheim.de/corona-modell.

[2] A. Bernasconi and S. Grandi, "A conceptual model for geo-online exploratory data visualization: The case of the covid-19 pandemic," *Information*, vol. 12, no. 2, 2021.

[3] T. C. Schelling, "Dynamic models of segregation," *Journal of mathematical sociology*, vol. 1, no. 2, pp. 143–186, 1971.

[4] E. Hunter, B. Mac Namee, and J. D. Kelleher, "A taxonomy for agent-based models in human infectious disease epidemiology," *Journal of Artificial Societies and Social Simulation*, vol. 20, no. 3, 2017.

[5] B. Faucher, R. Assab, J. Roux, D. Levy-Bruhl, C. Tran Kiem, S. Cauchemez, L. Zanetti, V. Colizza, P.-Y. Boëlle, and C. Poletto, "Agent-based modelling of reactive vaccination of workplaces and schools against covid-19," *Nature communications*, vol. 13, no. 1, pp. 1–11, 2022.

[6] Sirenko, M., Rui Yap, J., Sarva, S., Verbraeck, A., Comes, T., "D2.1 local behavioural model and recommendations for local covid-19." https://www.heros-project.eu/output/deliverables/. Accessed 30/05/2022.

[7] M. Zhang, A. Verbraeck, R. Meng, B. Chen, and X. Qiu, "Modeling spatial contacts for epidemic prediction in a large-scale artificial city," *Journal of Artificial Societies and Social Simulation*, vol. 19, no. 4, 2016.

[8] Y. Ge, R. Meng, Z. Cao, X. Qiu, and K. Huang, "Virtual city: An individual-based digital environment for human mobility and interactive behavior," *Simulation*, vol. 90, no. 8, pp. 917–935, 2014.

[9] Delft University of Technology, "Dsol manual." https://simulation.tudelft.nl/dsol/manual/. Accessed 30/05/2022.

[10] ISO 9241-11:2018, "Ergonomics of human-system interaction – part 11: Usability: Definitions and concepts," standard, International Organization for Standardization, 2018.

[11] M. Pietroń, "Analysis of performance of selected geospatial analyses implemented on the basis of relational and nosql databases," *Polish Cartographical Review*, vol. 51, pp. 167–179, 10 2019.

[12] Makris, A., Tserpes, K., Spiliopoulos, G. et al., "Mongodb vs postgresql: A comparative study on performance aspects," *Geoinformatica*, vol. 25, p. 243–268, 2021.

[13] A. Scott, M. MacDonald, and S. Powers, *JavaScript Cookbook*, ch. 6. O'Reilly, 3 ed., 2016.