LARGE-SCALE TRAFFIC SIMULATION FOR SMART CITY PLANNING WITH MARS

Julius Weyl
Ulfia A. Lenfers
Thomas Clemen

Hamburg University of Applied Sciences
Department of Computer Science
Berliner Tor 7
20099 Hamburg, Germany
{julius.weyl, ulfia.lenfers, thomas.clemen}
@haw-hamburg.de

Daniel Glake
Fabian Panse
Norbert Ritter

University of Hamburg
Department of Informatics
Vogt-Kölln-Straße 30
22527 Hamburg, Germany
{daniel.glake, panse, ritter}
@informatik.uni-hamburg.de

ABSTRACT

Understanding individual mobility in larger cities is an important success factor for future smart cities. Related simulation scenarios incorporate enormous numbers of agents, with the disadvantage of long run times. In order to provide large-scale and multimodal traffic simulations, we developed MARS V3. Adapting the Modeling and Simulation as a Service (MSaaS) paradigm, a seamless workflow can be provided to the modeling community. An integrated domain-specific language allows model descriptions without a technical overhead. For this study, selected parts of an individual-based traffic model of the City of Hamburg, Germany, were taken as an example. The entire workflow from model development, open data integration, simulation, and result analysis will be described and evaluated. Performance was measured for local and cloud-based simulation execution for up to one million agents. First results show that this concept can be utilized for building decision support systems for smart cities in the near future.

Keywords: agent-based, individual mobility, domain-specific-language, large-scale traffic scenario, MSaaS

1 INTRODUCTION

Individual mobility needs and demands are key issues in order to understand and manage traffic in larger cities (Grignard et al. 2018). Simulation plays a major role in solving transportation related issues in order to test planning and management scenarios before applying them to the real system (Bazzan and Klügl 2013). However, the vast amount of entities involved in daily traffic, e.g. humans, cars, buses especially requires new perspectives towards agent-based modeling and simulation (Waldrop 2018).

By applying the agent-based paradigm to the field of traffic simulations, a high levels of detail, i.e. microscopic simulations, can be achieved. This, in contrast to macroscopic simulations, allows us to gain insights about the non-linear dynamics of a traffic system itself (Schadschneider 2004).

Even though these types of simulations have different fields of applications, the detailed, microscopic ones have only been used in small scale scenarios while macroscopic simulations could cover large spatial extends. In the past, this has been a necessity due to scarce computational resources either through missing computation power, e.g. CPU and memory restrictions, or insufficient disk space to store extensive amounts

SummerSim-MSaaS, 2019 July 22-24, Berlin, Germany;

of result data. With powerful laptop computers and virtualization solutions available, this distinction is now fading, so that even large scenarios can be executed on a microscopic level of detail (Foytik et al. 2017; Weyl et al. 2018; Ramamohanarao et al. 2017). As a consequence, the complexity and extent of model descriptions and scenarios require a stronger research focus to the model definition process (Grignard et al. 2018).

We propose a seamless modeling and simulation workflow based on the MARS framework (Weyl et al. 2018). As an example, up to one million of individual car agents were were placed into the road network of the City of Hamburg, Germany. Experiments with varying numbers of agents were conducted in order to identify a threshold for moving from local simulation execution to a cloud-based one. The next section briefly presents the context of this work. Section 3 introduces into the MARS framework, whereas section 4 describes the overall workflow in detail. Section 5 and 6 present the experiments and their results. A discussion and a conclusion part finalize this paper.

2 RELATED WORK

Traffic simulation models can be classified based on their scale (Tchappi Haman et al. 2017). This includes both the covered spatial extent of the simulation as well as the level of detail. Microscopic simulations offer fine-grained insights because they include individual traffic participants like cars, pedestrians, bikes, and buses. In the field of microscopic modelling the agent-based approach remains the most popular one (Grignard et al. 2018, Kesting et al. 2008). Their biggest downside is the heavy resource requirements since the behavior of each entity has to be computed. Using the modeling and simulation as a service paradigm (MSaaS), this resource scarcity and missing analytical features can be mitigated by exploiting the features and elasticity of cloud computing. A public reference architecture are already proposed by (Hannay and van den Berg 2017), where besides necessary requirements, the system is separated by multiple top-level capabilities in a service oriented fashion. These cover aspects around *missions and operations*, *operational capabilities*, *user facing capabilities* as well as *backend capabilities*, each of them with further refinements. An outline by (Siegfried et al. 2014) goes one step further and concludes the decision from a more technical and orchestration perspective as well as propose also a set of requirements tackling the simulation execution in the near future.

Complementary to the detailed way of looking at things through microscopic simulations is the macroscopic approach. This type of simulations uses differential equations to depict the traffic as fluids or gases, thereby describing the flow of traffic (Schadschneider 2004). Using different levels of detail leads to distinct areas of application. Microscopic simulations traditionally have been used to examine limited spatial extends while macroscopic models were applied to large scenarios. Through the continued advances in computational power over the last decades, this clear distinction is melting away. While simulating districts, cities or countries was only possible through macroscopic simulations, the newly available resource abundance through cloud computing allows for a high level of detail even at these scales.

Agents and their definition of being autonomous entities, acting based on perceptions of their environment are the perfect templates to depict traffic participants. As before with the microscopic traffic simulations, most agent frameworks deal with scalability problems (Pawlaszczyk and Strassburger 2009). This either limits the number of agents that the frameworks can handle, the execution time for the simulation itself or the result data the framework is capable of producing. Another topic for traffic simulations is the data utilized for initialization purposes. For multi-agent systems with spatial relatedness, using of GIS data is considered of paramount importance (Dallmeyer et al. 2011). Not all frameworks allow to use this kind of data though.

New to this field is also the trend of 'OpenData' that is available to the public. Prominent examples where this kind of data has been used in traffic simulations are (Mcardle et al. 2014; Kickhöfer et al. 2016; Ziemke

and Nagel 2017; Ramamohanarao et al. 2017). In the case of (Ziemke and Nagel 2017), OpenStreetMap data was used to depict the streets of Berlin. A large variety of projects that deal with traffic simulation were described in literature, e.g. (Foytik et al. 2017), Matsim (Balmer et al. 2008), Transim (Strano et al. 2013) and Sumo (Krajzewicz et al. 2012) are prominent frameworks that have been used in the past (Bieker-Walz et al. 2015, Fernandes et al. 2013).

3 MARS FRAMEWORK AND TRAFFIC MODEL

The Multi-Agent Research and Simulation framework (MARS) provides an ecosystem for developing multi-agent simulations based on the Modeling and Simulation as a Service paradigm (Hüning et al. 2016, Parker and Epstein 2011). End users can create their own simulations in a variety of ways and then execute them directly on their machine or in the dedicated MARS cloud. The MARS system is a cloud-native framework that allows models to use the available server hardware, thereby scaling up the simulations. Results from simulations can be persisted to databases and files which allows for a wide variety of analysis including 3D visualization and visual analytics. After completing the third version of the framework (LIFE v3) we are now presenting the new capabilities by showcasing the workflow from creating a model all the way to running large-scale simulations.

The model used throughout this process is a microscopic simulation of Hamburgs street traffic. Agents in this model are cars which drive through the city. Each car-agent has individual driving parameters for acceleration, deceleration and top speed. A car-following model (*Intelligent Driver Model*) is used to give the agents a realistic driving behavior (Treiber et al. 2000). If the agent has enough space up front to drive, they do so while dynamically accelerating in every simulation step until they reach their top speed. If other road users get in the way, they adapt their speed so that they don't collide. In front of crossings the agents slow down, choose a random adjacent road and continue driving. In the beginning of the simulation, the agents are instantiated at randomly chosen intersection throughout the city. How such a model is build will be covered in the next section that describes the typical workflow.

4 WORKFLOW

Our approach to model creation and simulation execution consists of four basic steps as shown in figure 1. In the beginning of the model creation process, everything is being developed locally and probed under certain conditions, equivalent to the software development test-stage of checking for syntactic and semantic errors. Necessary for this step is a data preparation process that includes setting up the simulation parameters and adding the required input data like for example GIS files. If small-scale executions with an reduced amount of agents or a restricted spatial extend produce the expected results without throwing errors, the next step can be taken. This is the switch to running large-scale simulations where our approach enables the modeller to bundle the model and scenarios as one simulation unit which is then executed in the automatically provisioned cloud environment. The model then harvests the available resources of the cloud environment and persists all results into a appropriate data store (Hüning et al. 2016).

4.1 Model Development

Model development can be done in two ways, either through programming in C# or by using the MARS domain specific language (MARS-DSL). In both ways, the *first* step of the workflow is to build a conceptual model that clarifies the considered research question. This abstract form, usually a written description, is then translated into a well-structured and machine-readable MARS-DSL model. In that, the agent and layer entity types are created and fitted with individual attributes and data types. Alternatively it is possible to use

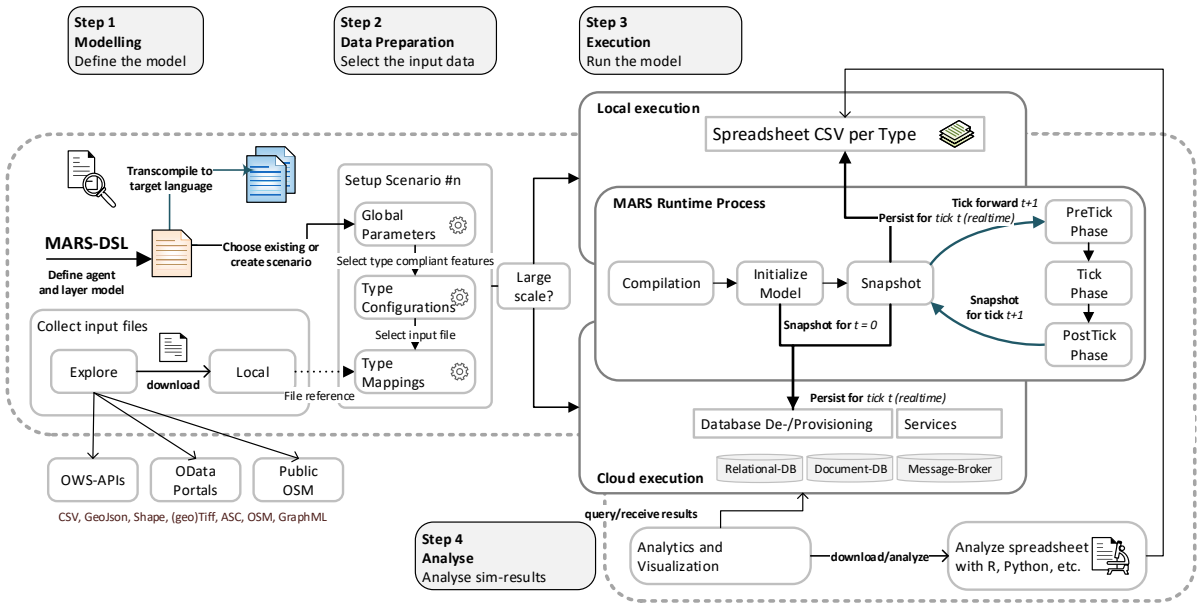

Figure 1: MARS V3 Workflow describing the modeling, execution and analysis phase.

an existing model from the core-library which equips the modeller with a set of useful modeling tools, i.e. an SI-unit converter-system or simple math calculation functionalities.

```
0  model SimpleRescueModel
1  layer RoadLayer
2
3  agent Pedestrian on RoadLayer {
4      external observe var NextPosition : Tuple<real,real>
5      external var Unconscious : bool
6      external val IndividualWalkVelocity : real
7      //...
8      def Fast() =>
9          if(Gender == GenderType.Male)
10             return Constants.MaleVelocity*Time.Delta
11         else return Constants.FemaleVelocity*Time.Delta
12 }
```

Listing 1: Example type definition with input and output parameters.

```
0  NextPedestrian := nearest Pedestrian
1      where [he => return he.IsUnconscious]
2  val pos = #(xcor,ycor)
3  if(Math.Distance(pos,NextPedestrian.Position) < 1)
4      NextPedestrian.Help
5  else move me Fast to NextPedestrian
6
7  //...
```

Listing 2: Example expression language for agent query and movement.

The MARS-DSL is a full featured *external* language, including concepts of individual based modeling for Multi-Agent systems. Modelers use agent-specific modeling paradigms when defining the agent behaviour based on the widely accepted *sense-reason-act* pattern. The conceptual model with its identified agent types, involved attributes and simulation environment, including spatial (i.e. satellite image) or temporal (i.e. time-series) data can directly be integrated in the description. With focus on the agent logic and their interaction, the language provides a set of expressions for handling exploration of the environment as well as various movement actions. A similar and more established general purpose language is SARL (Rodriguez et al. 2014), which besides the agent type itself also considers a multi-level holonic decomposition, where one agent is a sub part of another. For general purpose systems the SARL Language is recommended but for the case of individual-based simulation models, this language is not as tailored to the domain as the MARS-DSL. When it comes to performing complex agent movements, querying other entities and working with spatial and temporal data, the MARS-DSL provides a more suitable modelling environment.

Besides these describing elements, the language comes with a type-system to infer non-explicit defined types from each point in the model and which is used within the subsequently ahead-of-time (AOT) compilation process. A more detailed introduction of this process has already been given in (Glake et al. 2017). An example type definition with the MARS-DSL is shown in Listing 4.1, where a simple agent type can be defined by creating an instance of the agent-meta type and an assignment on which layer environment this instance shall live. Furthermore the agent input and output parameters can be defined similar to field definitions of regular object-oriented languages. A short example of the proposed expressions is shown in listing 4.1, where adapting the individual-based approach, the agent itself makes simple *nearest* query for another agent-type.

The language tries to bridge the gap between the high complex programming concepts of existing run-time languages, like C#, Java, Python or C/C++ and the well understood agent modeling paradigm, extended with concepts to query spatial and temporal data from layer data source. The AOT compilation step involves this through using a tree-transformation to generate coarse-grained code snippets, which are then combined into a cross-platform code block. Later these blocks are compiled into platform specific byte code for an optimized local or cloud execution.

4.2 Data Preparation

The second step in the workflow is the description of the desired scenario and input data that should be used to initialize the model. For data collection purposes we have developed an OpenData discovery and integration tool which uses the public OData endpoints as well as public OWS services, to query and transform publicly available raster and vector files into MARS compliant input. Instead of proprietary formats, open industry standards like (GeoTiff, ASCII Grid, simple Bitmaps and GDAL supported formats) are being used.

To start a simulation, a model and respective scenario is needed. Therefore, the modeller can specify an own scenario description within MARS-DSL, slightly decoupled from the model description and only holding references by the conceptual model and respecting type name. The description is concrete directly writable JSON or YAML document and control the global configuration for the simulating time resolution, i.e. x steps or 1 seconds for the traffic model and the time interval as real-time or an amount of steps to simulate.

4.3 Model Execution

After the model has been prepared, the simulation runtime executes the model according to the given scenario description. Each of the input files is collected and the containing data assigned to the respective layer or agent instance. From the technical perspective, the agents get mapped to corresponding initialization file entries and are then created. Each table T is then associated with an equivalent agent model type A. *External* and thus required attributes of A are mapped by exactly one column of the input table. If no input file is available, also a constant attribute value can be specified, which is then valid for all instances of the type. If there are no *external* attributes defined, this step is skipped. The mapping only includes *value-types* and not *reference-types*, so that e.g. interaction references between agent entities need to be resolved in the model itself or by another intermediate step.

A local execution will be performed concurrently on one or multiple processors. Each of them executes the actual simulation *tick* for each agent. Internal, the *tick* is projected to the logical *simulation step* for the specified Δt or the corresponding real time e.g. 2019.01.01 12:05:01, calculated by the *start-time point* within the scenario. These configuration are necessary for models with unit subjected equations, such as velocity calculations. Within one simulation step, the agent follows his internal logic of operators and

statements, as described in 4.1 section. For traffic simulations these operators are mapped to a so called Spatial Graph Environment (SGE) (Weyl et al. 2018), which is an in-memory graph data structure to depict road system. Besides well-known graph algorithms, it comprises high-frequency movement and explore semantics for other containing entities, i.e. agents exploring the environment or moving along a road. Internally the SGE uses an index-based query for each operator to get predecessors and successors, to query entities in front or behind or to perform collision tests between entities, by checking overlapping lines. These queries are lane-precise, so that merges and branches of lanes are considered in the query execution.

At the end of each simulation step, a snapshot of the current state gets persisted either into a database or to a set of spreadsheet CSV files. It is possible to specify which attributes of the agents are relevant for the own research question and therefore should be persisted through the model description. Once the execution finished, the modeller can use tools like R, Python etc. analyse the results. Each agent key-frame k_{mi} stands for *one* agent m at tick i, representing one data row in the output table or a document/message in the cloud deployment. This keyframe is associated with the current agent position p_i for the tick i and optional the corresponding realtime projection $r(i)$ as well as a set of recorded primitive agent attributes $\{a_{1i}, ..., a_{ni}\}$, specified by MARS-DSL or the scenario.

When the amount of agents, the spatial extent of the considered environment or the internal logic needs to increase and be more complex, able to profit from emergent effects or other heavy-weight scenarios, i.e. resulting traffic-jams for huge events, or the modelling of incoming and outbound traffic flow of the city, a more suitable environment is needed. On the one hand obviously, the model execution performs better through vertical CPU and Memory scalability, proposed by the cloud infrastructure. On the other hand this also profits from the massive multi-tenant storage infrastructure, taking place to get each individual result and layer output of all ticks be persisted and available for subsequently or later data analytic tasks. Due to this identified problems we established an *automatic* cloud deployment pipeline, where the locally specified scenario and first model probes can directly be deployed into this environment. The logical deployment view is shown in figure 2.

The deployment components used during the modeling phase are the Modeling and Decision Support Tool (MDSS), as well as the MARS runtime, running already locally on the scientists's machine. The MARS Runner Service (MRS), the WebAnalytics Board (WA), the WebGL Visualization (WGLV), the the Messaging system (MS) and database layer running in a Cloud environment. All components are active throughout all life cycle phases and the scientist develops his model using the MARS-DSL within the MDSS tool. During the cloud model execution he can monitor the running simulation instances and in contrast to the established business workflow domains in the eScience workflow domain there is typically only one role responsible for the modeling, deployment and execution of a simulation or application. Therefore, just like in many eScience systems, we integrated the modeling, deployment and monitoring functionalities in one tool (Sonntag and Karastoyanova 2010). The WA and the WGLV provide realtime results of running and already finished simulation executions.

The MRS endpoint is the basic piece of the cloud interface needed to provision the cloud execution middleware. Instead of provisioning the whole execution middleware in one step, we follow a two-step bootstrapping process. In the first step (step 1), the MRS creates a job descriptor which contains information about the run. These are information's about the sim-run name, the user (tenant) triggering this deployment and a retry counter which can be used together with a start time, to build this deployment as a batch-job. Before applying this specification, the MRS takes the input model and scenario configuration, package them together into one container image and persist them into a container registry. In the second step (step 2), the MRS applies the job descriptor loads the packaged image from the registry and schedules the such called *sim-container* to exactly one cluster node. The *sim-container* can now be accessed via the specified name. At this point the MARS runtime, which was also used during the local development, assumes the remaining responsibility.

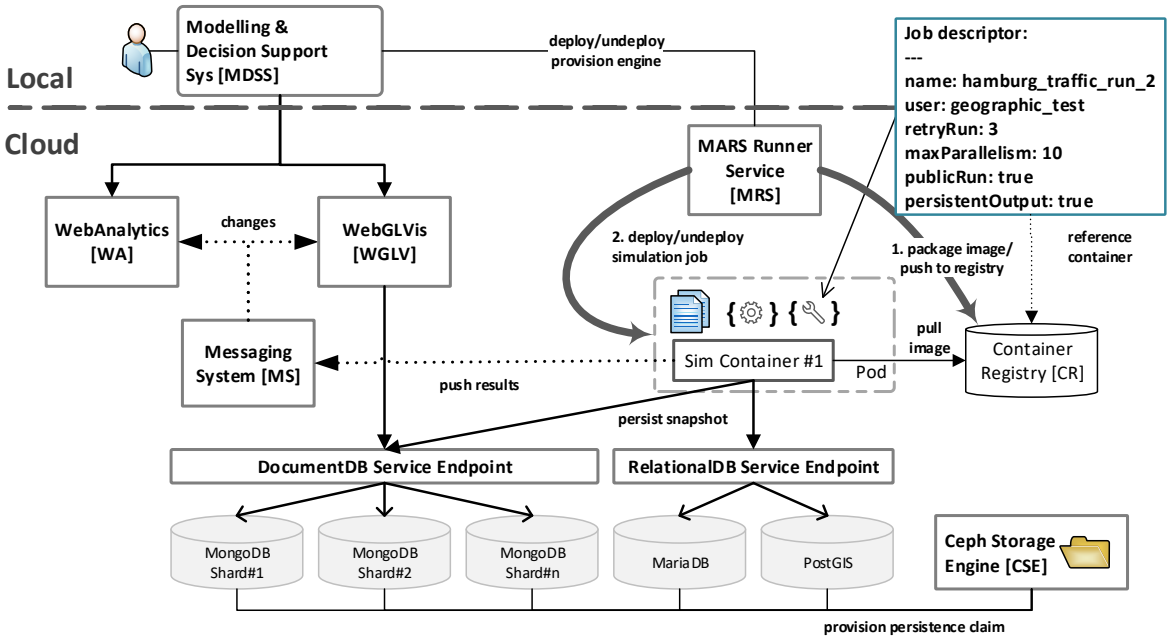

Figure 2: MARS V3 Cloud deployment view.

Instead of writing the keyframes into entity-type separated CSV files, the runtime now uses the underlying storage engine. Each database node is already deployed or will be provisioned by the cluster and associated with their persistent volumes. This contains the underlying durable file-system and database files what is managed by a distributed Ceph file system (Weil et al. 2006). The database service schedules the queries with an integrated load balancer. In addition to the regular output, each update will be pushed into the Messaging system what informs each registered WA or WGLV participant. This decoupling facilitates it to integrate other services as well.

After all, finished simulations will be marked as completed and also running simulations can be stopped by the MRS endpoint by simply forwarding a stop message to the internal MARS runtime in the *sim-container*. The simulation loop will be interrupted, remaining results will be persisted and all acquired resources released.

5 EXPERIMENTS

The multi-agent traffic model was executed at multiple scales to assess the limits of local execution and to find out when it is advisable to switch to simulating in the cloud. Results from performing the experiments are presented in the next section while the classification takes place in the DISCUSSION section. Setting for the simulations was *Altona*, one of the biggest districts of Hamburg. It is located in the cities west and spans an area of 77.5 km^2 with the total road length equating to 545.884 km. The graph that represents this part of town is comprised of 16.743 nodes and 43.825 edges. During the experiments, different amounts of agents were simulated until the local execution-time exceeded reasonable limits. This limit is reached once 1 simulated second takes longer than 1 second in real time. To compensate for varying tick durations, this requirement was changed so that the simulation of 1h of traffic mustn't exceed 1h in real time. The simulation scenario was run with $10^3, 10^4, 10^5$, and 10^6 car agents. All simulation series were deployed

locally and in the cloud environment. The results from multiple runs were noted and averaged afterwards. Simulation time was one hour with a $\Delta T = 1$ sec., i. e. 3600 ticks per run.

The experiments were conducted using two different hardware configurations. The computer used for local execution is a MacBook Pro (late 2016) with a quad core $2,6GHz$ Intel Core i7 processor, 16 Gigabyte of memory and $512Gb$ of SSD storage. At the time of performing the experiments, macOS 10.14.3 Mojave was installed. The simulations carried out in the cloud environment were run on SuperMicro SuperServer 6029TP-HTR TwinPro2 machines and Mac Pro's from the end of 2013. Former are configured with two 10 core Intel Xeon Gold 5115 processors per node, running at $2,4GHz$, $192GB$ memory and a SSD with $1,6TB$. The latter have a $3,5GHz$ Intel Xeon 6 core processor, $64GB$ of memory and $200GB$ of disk space available. For all cloud machines Core OS is being used as operating system on top of which a Kubernetes cluster is running, orchestrating the simulation pods and equally distributing them to the machines, described above. All simulations in the cloud are executed inside Docker containers that use the Dotnet 2.1 runtime as base.

6 RESULTS

Table 1 shows the results from the local and cloud simulating runs. All simulations were executed four times and the average running-time was calculated. The first group of experiments with 10^3 agents took, on average, $43.5s$ locally and $37.61s$ in the cloud which gives a ratio of 0.86. Simulating 10^4 agents took $8m.54s$ on the laptop and $8m.1s$ in the cloud, resulting in a 0.9 ratio. Experiment group number three with 10^5 agents took $1h,22m,14s$ locally and $1h,12m,59s$ in the cloud (ratio of 0.89). The last simulations with 1 million car-agents took $12h,20m,29s$ on the laptop and $10h,57m,26s$ in the cloud which results in a ratio of 0.88.

Figure 3 shows the plots of average run-time measurements from tick 0 to 3600, grouped by an interval of 10 ticks. For each time line it shows fluctuations in the local execution, emphasize with measured peaks between tick 1400 and 2500 for 10^3 and 10^4 agents as well as an initialization overhead only for the 10^3 scenario at tick 0. The increasing deviation of the local- to the cloud time shows a benefit of up to 14%. This starts to throttle when it comes to run scenarios with 10^7 agents, where the deflection has the maximum at tick 2240 with an average difference of 1854 *milliseconds*. This benefits shrinks to 1208 *milliseconds* for 10^5.

Table 1: Running times for simulations in the cloud and locally with different amounts of agents.

Amount of agents	avg local sim duration	avg cloud sim duration	ratio
1000	43.5s	37.61	0.86
10000	533.75s	480.89s	0.9
100000	4934.42s	4379.49s	0.88
1000000	44429.82s	39446.76s	0.88

7 DISCUSSION

The difference in running time between local simulation and execution times in the cloud environment is up to 14% for the used traffic model. Throughout the range of simulation with 1000 agents to 1 million agents, the ratio varies between 10% and 14%. For large runs with 1 million agents this saves up to $1h$ and $23m$ of time when compared to the local execution. This was to simulate one hour of traffic.

Technically, the two ways of executing the simulation are not fully comparable. However, by this study we wanted to analyze whether the naive approach of deploying the execution engine onto the Kubernetes cloud would bring any advantages.

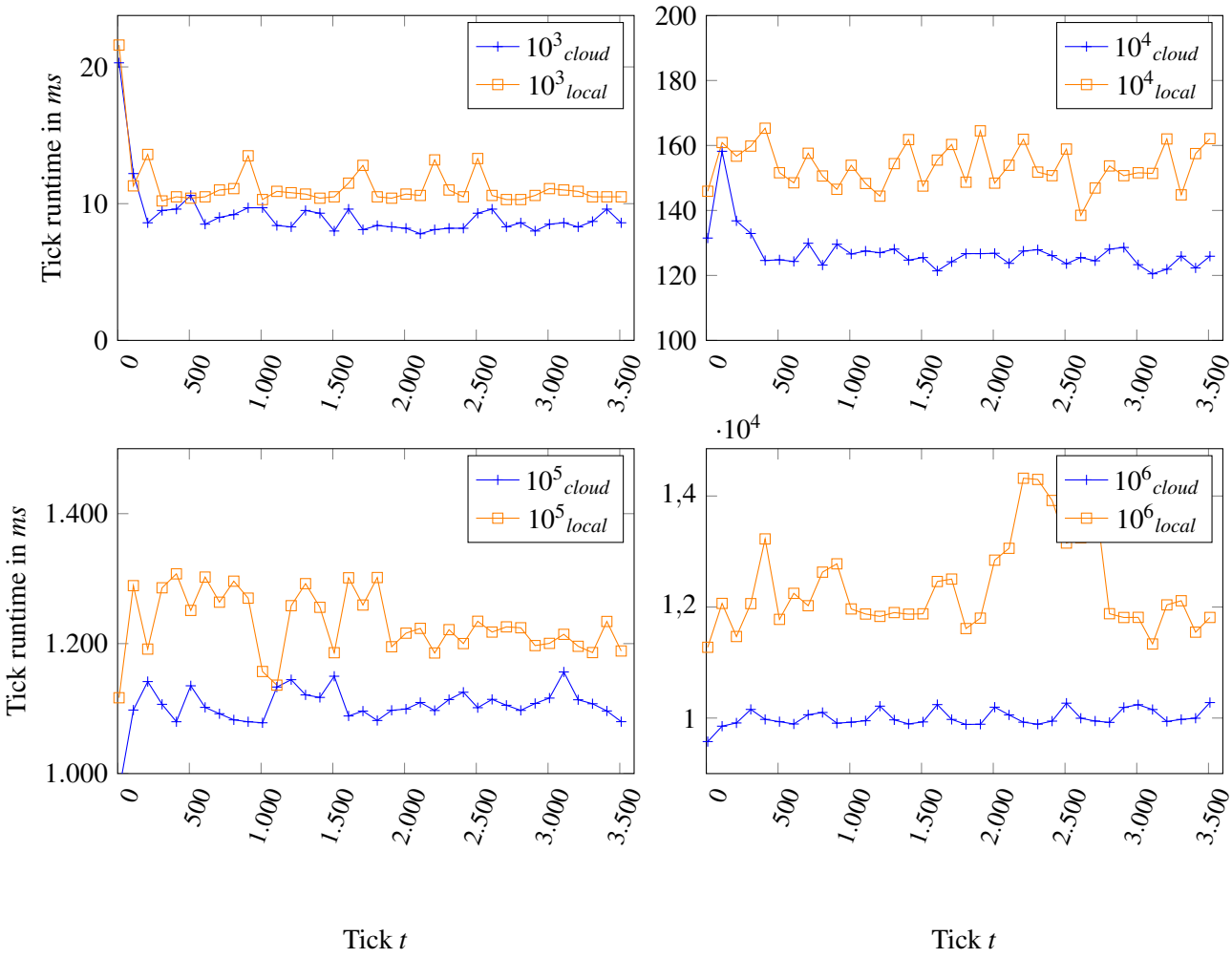

Figure 3: Comparison of *local* and *cloud* based execution for different agent amounts.

For small-scale settings, the local development process is feasible up to scenarios with 10.000 - 50.000 agents. Once over this threshold, the total simulation duration exceeds the simulated time. For example the simulations with 100.000 agents took 1 hour and 22 minutes on average while simulating one hour of traffic. Predictive scenarios where the simulation has to run fast enough to depict future conditions aren't possible anymore at that scale. Even though the execution times in the cloud were shorter, they only allow to cut down the running time by up to 10 percent.

Simulations of 1 million agents are not advisable on personal computers. Since running times exceed one day, the batch processing in the cloud offers more convenience. Even if the simulation takes multiple weeks, the process can be left running in the cloud where long running times don't interfere. Not using the own personal computer for a week is just not feasible. In context of larger amounts of agents the cloud-based execution also benefits from the underlying data-management. The fluctuations over time indicate that the system is busy with rescheduling of concurrent processes and their management of data-frames. While this overhead is noticeable in run with lower amount of agents, the run-time deviation can be neglected in larger scenarios.

8 CONCLUSION

MARS V3 is an agent-based modeling and simulation framework that provides planning and decision-support capabilities to city planners and political stakeholders on a massive scale. This study compares performance measures of a local execution with a cloud deployment in order to develop recommendations for its usage. Additionally, we presented a comprehensive workflow from model development to the analysis of results.

The performance results indicated that with the presented twofold architecture it is plausible to develop very complex models locally, moving to a cloud-based execution afterwards when it comes to serious scenarios and that require persisting large simulation results for a later analysis.

However, the current architecture uses a single-node execution principle to run the equivalent scenarios and models from the same perspective and just scales the run-time vertically, but also getting benefits from persisting the result into the target databases. In terms of the local, single-node testing environment the boundary of limited amounts of processors still needs to be overcome i.e. by integrating GPU-based processing in manifold ways. This can be done by for specific accelerations of environment *move* and *explore* tasks. In cases of the cloud-based execution the run-time reduction will benefit from a horizontal scalability of the SGE graph, what is still a hard problem in terms of graph-databases.

Future work includes enhancing the model by multi-modal transport features and adaptive human behavior. Furthermore, the simulation results will be validated by field data obtained through public authorities and public transport organizations. Additionally, the presented approach will also be adapted to other cities.

ACKNOWLEDGMENTS

This study was partly funded by the ahoi.digital initiative, City of Hamburg, Germany (SmartOpenHamburg project).

REFERENCES

Balmer, M., K. Meister, M. Rieser, K. Nagel, and K. W. Axhausen. 2008. “Agent-based simulation of travel demand Structure and computational performance of MATSim-T”. In *2nd Conference on Innovations in Travel Modeling*. Portland.

Bazzan, A. L., and F. Klügl. 2013. “A review on agent-based technology for traffic and transportation”. *Knowledge Engineering Review* (29), pp. 375–43.

Bieker-Walz, L., D. Krajzewicz, A. Morra, C. Michelacci, and F. Cartolano. 2015. “Traffic Simulation for All: A Real World Traffic Scenario from the City of Bologna”. *Lecture Notes in Control and Information Sciences* vol. 13, pp. 47–60.

Dallmeyer, J., A. D. Lattner, and I. J. Timm. 2011. “From GIS to Mixed Traffic Simulation in Urban Scenarios”. *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pp. 134–143.

Fernandes, R., F. Vieira, and M. Ferreira. 2013. “Parallel Microscopic Simulation of Metropolitan-scale Traffic”. In *Proceedings of the 46th Annual Simulation Symposium*, ANSS 13, pp. 10:1–10:8. San Diego, CA, USA, Society for Computer Simulation International.

Foytik, P., C. Jordan, and R. M. Robinson. 2017. “Exploring Simulation Based Dynamic Traffic Assignment With A Large-Scale Microscopic Traffic Simulation Model”. In *ANSS ’17 Proceedings of the 50th Annual Simulation Symposium*. Virginia Beach, USA.

Glake, D., J. Weyl, C. Dohmen, C. Hüning, and T. Clemen. 2017. "Modeling through model transformation with MARS 2.0". In *Proceedings of the Agent-Directed Simulation Symposium*, pp. 2. Society for Computer Simulation International.

Grignard, A., L. Alonso, P. Taillandier, B. Gaudou, T. Nguyen-Huu, W. Gruel, and K. Larson. 2018. "The Impact of New Mobility Modes on a City: A Generic Approach Using ABM.". In *Unifying Themes in Complex Systems IX. ICCS 2018. Springer Proceedings in Complexity.*, edited by A. Morales, C. Gershenson, D. Braha, A. Minai, and B.-Y. Y., Springer, Cham.

Hannay, J. E., and T. van den Berg. 2017. "The NATO MSG-136 reference architecture for M&S as a service". *M&S Technologies and Standards for Enabling Alliance Interoperability and Pervasive M&S Applications (STO-MPMSG-149), Lisbon*, pp. 20–21.

Hüning, C., M. Adebahr, T. Thiel-Clemen, J. Dalski, U. A. Lenfers, L. Grundmann, J. Dybulla, and G. A. Kiker. 2016. "Modeling & Simulation as a Service with the Massive Multi-Agent System MARS". ADS '16, pp. 8. San Diego, CA, USA, Proceedings of the 2016 Spring Simulation Multiconference.

Kesting, A., M. Treiber, and D. Helbing. 2008. "Agents for Traffic Simulation". *Multi-Agent Systems: Simulation and Applications*, pp. 325–356.

Krajzewicz, D., J. Erdmann, M. Behrisch, and Bieker Laura. 2012. "Recent Development and Applications of SUMO - Simulation of Urban MObility". *International Journal On Advances in Systems and Measurements*.

Parker, J., and J. M. Epstein. 2011. "A Distributed Platform for Global-Scale Agent-Based Models of Disease Transmission". *ACM Trans. Model. Comput. Simul.* vol. 22 (1), pp. 2:1–2:25.

Pawlaszczyk, D., and S. Strassburger. 2009. "Scalability in Distributed Simulations of Agent-Based Models". In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. Rossetti, R. Hill, A. D. Johansson, and R. Ingalls.

Rodriguez, S., N. Gaud, and S. Galland. 2014. "SARL: a general-purpose agent-oriented programming language". In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Volume 3, pp. 103–110. IEEE.

Schadschneider, A. 2004. "Physik des Straßenverkehrs". *Institute for Theoretical Physics University of Cologne*.

Siegfried, R., A. Diehl, S. Y. Diallo, M. Bertschik, G. Herrmann, and M. Rother. 2014. "Outline of a Service-Based Reference Architecture for Effective and Efficient Use of Modelling and Simulation". *NATO Modelling & Simulation Group (NMSG) Multi-Workshop, Paper 18, Washington D.C., USA*.

Sonntag, M., and D. Karastoyanova. 2010. "Next generation interactive scientific experimenting based on the workflow technology". In *Proceedings of the 21st IASTED International Conference on Modelling and Simulation (MS 2010)*.

Strano, E., M. Viana, L. da Fontoura Costa, A. Cardillo, S. Porta, and V. Latora. 2013. "Urban street networks, a comparative analysis of ten European cities". *Environment and Planning B: Planning and Design* vol. 40 (6), pp. 1071–1086.

Tchappi Haman, I., V. C. Kamla, S. Galland, and J. C. Kamgang. 2017. "Towards an Multilevel Agent-based Model for Traffic Simulation". *Procedia Computer Science* vol. 109, pp. 887–892.

Treiber, M., A. Hennecke, and D. Helbing. 2000. "Congested Traffic States in Empirical Observations and Microscopic Simulations". *Physical Review E 62*.

Waldrop, M. M. 2018. "Free agents". *Science* vol. 360 (6385), pp. 144–147.

Weil, S. A., S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn. 2006. "Ceph: A scalable, high-performance distributed file system". In *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320. USENIX Association.

Weyl, J., D. Glake, and T. Clemen. 2018. "Agent-based Traffic Simulation at City Scale with MARS". In *Proceedings of the Agent-Directed Simulation Symposium*, ADS '18, pp. 2:1–2:9. San Diego, CA, USA, Society for Computer Simulation International.

Ziemke, D., and K. Nagel. 2017. "Development of a fully synthetic and open scenario for agent-based transport simulations – The MATSim Open Berlin Scenario". *VSP Working Paper 17-12, TU Berlin, Transport Systems Planning and Transport Telematics*, pp. 1–11.

AUTHOR BIOGRAPHIES

JULIUS WEYL is a master's student in computer science and a research associate at the University of Applied Sciences in Hamburg, Germany. His research in the MARS group focuses on large-scale, agent-based simulations of individual mobility in urban scenarios and socio-ecological modeling. His email address is julius.weyl@haw-hamburg.de.

DANIEL GLAKE is a PhD candidate at the University of Hamburg and has a Master degree in computer science by the University of Applied Science in Hamburg. His research focus lies on polyglot data management, language- and compiler-engineering, including their transformation and more efficient data structures for simulation execution or general purpose applications. His email address is daniel.glake@uni-hamburg.de.

THOMAS CLEMEN holds a position as a full professor at the University of Applied Sciences in Hamburg, Germany. His teaching activities predominantly cover topics within information management and data science, whereas he is focusing on modeling and simulation of dynamic, complex and self-organizing systems in his interdisciplinary research. His email address is thomas.clemen@haw-hamburg.de.

ULFIA A. LENFERS is a PhD candidate at the Hamburg University of Applied Sciences in cooperation with the CAU University Kiel. She has a Diploma degree in geography and a Master of Science in interdisciplinary environmental science. Her research focus is the analysis and modeling of complex social-ecological and urban-geographical systems of various space-time scales. Within the MARS group it is her focus to build the bridge between geography and computer sciences. Her email address is ulfia.lenfers@haw-hamburg.de.

NORBERT RITTER is a full professor at the Department of Informatics, Faculty of Mathematics, Informatics and Natural Sciences, University of Hamburg, heading the Databases and Information Systems group. His research focus areas are modern database technology, esp. NoSQL-Systems, and scalable infrastructures for big data management and cloud data management. His email address is ritter@informatik.uni-hamburg.de.

FABIAN PANSE is a PhD at the University of Hamburg. His research and teaching activities cover several topics in the field of databases and information systems, whereas he is focusing on data cleaning, data integration and uncertain data management. His email address is panse@informatik.uni-hamburg.de.