Adaptivity in distributed agent-based simulation: A generic load-balancing approach

Stig Bosmans[1], Toon Bogaerts[1], Wim Casteels[1], Siegfried Mercelis[1], Joachim Denil[2], and Peter Hellinckx[1]

[1] University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Sint-Pietersvliet 7, 2000 Antwerp, Belgium
{stig.bosmans, toon.bogaerts, wim.casteels, siegfried.mercelis, peter.hellinckx}@uantwerpen.be

[2] Flanders Make, University of Antwerp, Groenenborgerlaan 171, 2020 Antwerp, Belgium, joachim.denil@uantwerpen.be

Abstract. Distributed agent-based simulations often suffer from an imbalance in computational load, leading to a suboptimal use of resources. This happens when part of the computational resoures are waiting idle for another process to finish. Self-adaptive load-balancing algorithms have been developed to use these resources more optimally. These algorithms are typically implemented ad-hoc, making re-usability and maintenance difficult. In this work, we present a generic self-adaptive framework. This methodology is evaluated with the Acsim framework on two simulations: a micro-traffic simulation and a cellular automata simulation. For each of these scenarios a scalable and adaptive load-balancing algorithm is implemented, showing significant improvements in execution time of the simulation.

Keywords: Distributed agent-based Simulation · Adaptivity · MAPE-K · Dynamic load balancing.

1 Introduction

Although Agent-Based Simulation (ABS) is a relatively new simulation paradigm [15], it has been used as an effective tool in a wide range of research domains [1, 2, 4, 19]. The main characteristic in ABS is the concept of an agent, which is a self-contained autonomous entity, with the ability to interact with other agents and with the environment. These interactions can lead to complex emergent behavior [6]. Agent-Based Simulation is, therefore, one of the most powerful and natural tools to simulate emergent phenomena using a bottom-up approach.
ABS has been used to evaluate and analyze behavior of complex large-scale dynamic systems such as traffic systems [1] or complex Internet of Things systems such as smart city environments [4]. However, traditional monolithic ABS simulations quickly run into problems when the scale of the simulation increases. This is the case This becomes especially problematic when the application of these

simulations is time-critical. Therefore, reducing the computational cost and the run-time of these simulations is vital.
With this motivation, researchers have replaced the classic monolithic set-up by a distributed architecture. This can be achieved by partitioning the simulation into separate logical processes. This allows the simulation to be divided among multiple processors and servers, thus allowing to simulate larger systems and reducing the simulation run-time. This however also increases the complexity and may add inefficiencies such as the need for synchronization and slow remote communication between simulation partitions. Furthermore, the inherently dynamic aspect of agent-based simulation makes static partitioning inefficient because the computational load of each process changes during the simulation. This can lead to a significant waste of resources, for example, the simulation can start perfectly balanced, but over time the distribution of these agents can become highly imbalanced. Such distribution imbalance is often due to agents that change their locations, increase communications or change their internal load. A direct consequence of such imbalances is a significant increase in run-time. This is demonstrated in figure 1 for just two Logical Processes (LP's): the load of LP 1 increases, and as a consequence the Global Step Duration (GSD) increases like-wise. The figure also clearly demonstrates the corresponding waste of resources as the processing unit for LP 2 is underutilized most of the time. As stated by Long et. al. it is likely that such load imbalances occur in distributed agent-based simulations [14].

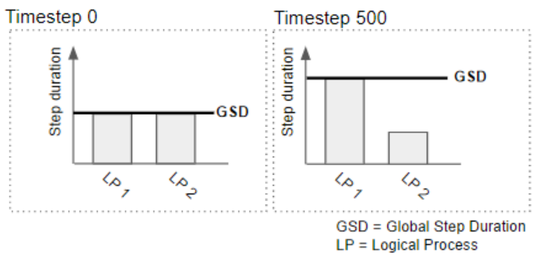

Fig. 1. Imbalance in step duration caused by a single LP

In this paper, we propose to organize the distribution adaptively by dynamically reacting to imbalances in computational load, synchronization load, and communication load. Most state-of-the-art load-balancing mechanisms are implemented in an ad-hoc manner, making them hard to reuse and maintain. The contribution of this paper is a generic framework to implement self-adaptivity in distributed agent-based simulators. We evaluate this method using two different implementations: a large-scale micro-traffic simulation with a graph-based environment and a cellular automata simulation with the Sugarscape model.
The second section of this paper discusses the concept of adaptivity and related

work. Section three presents the architecture of the distributed agent-based simulation framework Acsim, that will be used to evaluate the experiments. Section four presents the main principles of a MAPE-K loop and its implementation in Acsim. Section five presents the specific examples and the conclusions are drawn in section 6.

2 Adaptivity in Agent-Based Simulation

Adaptivity in agent-based simulations can be related to the notion of activity which was introduced by Muzy et. al. as a measure of the number of events occurring during a discrete event simulation [17]. As stated by Y. Van Tendeloo et. al. activity can be interpreted depending on the particular resource one wishes to focus on (time, memory, energy,..) [22]. Therefore both the communicational load and the computational load can be seen as types of 'activity'. For example, from a communicational load perspective, an agent has high activity if it generates many messages in a fixed time window. From the computational load perspective, an agent has high activity if its step duration takes a long time to process.

Given this definition of activity, we can go ahead and define adaptivity as the property of a distributed simulation framework to dynamically react to imbalances of activity with the aim to restore the balance and improve overall simulation run-time.

Adaptivity is typically implemented as a load balancing optimization problem based on global information [23] [5]. The activity is defined as a function of computational load, synchronization load and communication load. The disadvantage of these approaches is that they require global information to be stored or synchronized centrally and that the optimization algorithm is computationally intensive and thus less scalable. It is also possible to use heuristics that only require local information, making these solutions computationally much more efficient, but the obtained optimum might be local. For example, D'Angelo et. al. present in their work a range of heuristics that trigger agent migrations based on local and remote communication patterns [7] and Q. Long et. al. present a distributed load balancing algorithm based on partial local information [14].

But adaptivity is not constrained to solving load balancing problems only. In [9] and [3] the authors show that adaptivity can be used to dynamically switch abstraction levels of a single agent or a collection of agents. Switching to a higher abstraction level leads to a reduction in the computational load at the cost of losing accuracy.

Most of the related work rely on ad-hoc implementations of adaptivity. An exception is the work of Franceschini et. al. who are using a MAPE-K control loop to implement an automatic simulation abstraction solution [9]. In the following sections, we expand on this work and present the integration of a MAPE-K control loop in the Acsim distributed simulation framework. Furthermore, we show that MAPE-K can also be used effectively for adaptive load-balancing.

3 Distributed Simulation Architecture: Acsim

Acsim is a distributed Python-based agent-based framework, developed by the authors, inspired by Mesa [16]. It has been developed as a prototyping simulation framework. The goal of the framework is not to be a production-ready simulation framework but to allow for the validation of state-of-the-art techniques regarding simulation scalability. We hope that these techniques will eventually inspire production-ready distributed agent-based simulation frameworks.
One of the main motivations for the development of Acsim is the observation that there is an increasing need for large scale simulators in the context of Internet of Things (IoT) and Smart Traffic applications. Due to the increase of connectivity of smart devices and the availability of real-time data, simulation platforms provide the opportunity to simulate entire cities. Simulation technology enables the creation of a virtual testbed of large-scale IoT applications and allows for real-time simulation-based optimization. An application of this technology is, for example, a real-time city-wide and simulation-based traffic light optimization platform. But state-of-the-art simulators are limited in their scalability capabilities to support such technology. This is the challenge that Acsim tackles. Although Acsim focuses on large scale IoT and traffic simulation, it can also support other agent-based simulations.

Acsim relies on a conservative time-stepped synchronization mechanism. Where

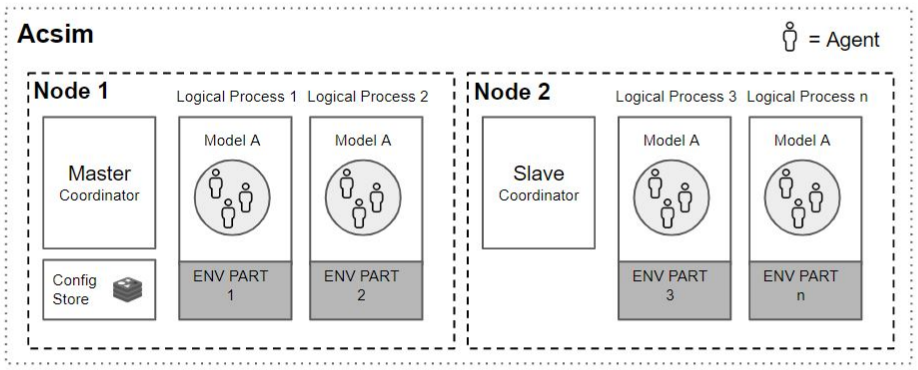

Fig. 2. Acsim - Distributed simulator architecture. Acsim contains a cluster of nodes and a node represents a physical device with one or more CPU cores, connected to other nodes via the network.

time is collectively progressed after the completion of each individual agent step. The architecture of the simulator is displayed in Figure 2. Acsim consists of three main building blocks: 1) Agent: represents an entity at its highest granularity, an agent contains a state, can adapt its state at each time-step and has the possibility to interact with other agents using message-passing and interact with the environment. 2) Model: a model serves as a container for a specific type of agent and is responsible for the initialization of all agents of this type. For example, a class of car agents will be part of a car model. This car model will initialize all cars, generate routes and collects car-related logging information. 3) Logical Processes: Acsim consists of multiple sub-simulator or LP's. Each LP manages a part of the environment and a collection of agents that are located in this partial environment. It runs a dedicated process and is responsible for low-level simulation tasks such as handling agent migrations, managing message-passing between local and remote agents, collecting logs and initiating agent steps. An agent step is a discrete step forward in time. Only as part of a step can an agent adapt its state or communicate with other agents and the environment. The global synchronization is managed by the master coordinator. The coordinator orders all LP's to execute the next step. Furthermore, the coordinator collects and stores logs generated by the LP's. Finally, Acsim has extensive monitoring capabilities, enabling an in-depth analysis of local and global simulator performance.

4 MAPE-K as a generic framework for adaptivity

Due to the ever-increasing complexity of computing infrastructure, a shift to self-managing systems is observed in the field of software development. In 2005, IBM introduced MAPE-K loops to deal with this complexity [10]. Measure Analyze Plan Execute - Knowledge (MAPE-K) loops are closed feedback loops which can handle the complexities of self-adaptivity. More recently, [11] described templates on how to utilize MAPE-K control loops to different distributed applications. The implementation of most adaptive optimization strategies in a simulation is ad-hoc and cannot be reused efficiently. We propose the application of a MAPE-K control loop as a generic solution that will allow existing adaptivity strategies to be efficiently implemented and maintained.
As mentioned above, the Acsim framework is step-based which results in the simulation being as fast as the slowest simulator in the distribution. There is no guarantee that this local optimization leads to a global optimum. The overhead of calculating the global optimum, at a master node, increases with the scale of the simulation. Because of the varying load-distribution over time, the global optimum shifts and a new optimization iteration is needed. Our approach focuses on a distributed solution to partitioning/merging environments. Our approach is generic, each simulator can easily implement its specific logic as part of the MAPE-K framework implemented in Acsim. Execution of the MAPE-K loop is handled by the Acsim framework. Next, we will go in-depth on the structure of the MAPE-K framework integrated into Acsim:

1. **Monitor**: During this phase, logs are retrieved from each subpart of the Acsim framework regarding the model, simulator and environment. When a MAPE-K iteration starts these logs are stored to the shared knowledge. This

knowledge base is located at the master node. To enhance the scalability, only low compute algorithms are used at the master level.

2. **Analyze**: This has access to the shared knowledge base to identify bottlenecks and flag optimization opportunities. These identifications do not provide a solution but an indication of the performance of a certain entity in the framework.
3. **Plan**: This step collects all flags and generates an optimization plan without execution. There could be multiple optimization plans in a single MAPE-K loop.
4. **Execute**: This phase of the loop runs distributed after receiving an optimization request from the planning phase. This phase has the highest computation requirement in the loop. The optimization algorithms used can vary from each application. When a local optimization is complete, a synchronization message is sent to all relevant entities involved in the optimization.
5. **Knowledge**: This part is shared between the first three steps of the loop. The execute step does not need the knowledge base as it only executes the plans created during the previous step. During each iteration, the knowledge can be expanded to store relevant information for future MAPE-K loops.

Each simulation will have access to the simulation logs, these are stored in the knowledge class. The MAPE-K framework implemented in Acsim allows easy implementation of the phases and allow for reuseable, maintaineable and application-specific adaptivity behavior. The loop can be executed both locally and centrally. Also a hierarchy of multiple loops, affecting each other is supported by the framework.

5 Motivating Examples

In the previous sections we introduced the concept of adaptivity and how we can implement it generically in the Acsim framework using the MAPE-K framework. In this section we validate this approach on two different agent-based simulations. In both scenarios we implement a novel activity load balancing heuristics. As stated in section 4, we differentiate compared to classical adaptive load balancing algorithms by making sure the heuristics are not performed centrally but at the level of a LP to ensure scalability. In the experiments our aim is to improve the global step duration GSD of the entire simulation. We can express it as follows: $GSD = max_i(LSD^i)$, where LSD^i is the local step duration of LP i. In other words, the global step duration is always equal to the worst LP step duration. The reason for this is that Acsim relies on a conservative time-stepped synchronization algorithm, as discussed in section 3. In the examples below the goal is to improve the activity balance with each MAPE-K iteration. To gain insight in how LP's are performing, we distinguish the different contributions to the step duration (as discussed in detail in [7]): the Model Computation Cost (MCC), the Remote Communication Cost (RCC), the Local Communication Cost (LCC) and the Model Synchronisation Cost (MSC). The weight of each contribution is application-specific. When an imbalance occurs, for each variable a different optimization strategy could be used. When optimizing on a local level, each LP calculates their cost balance using only local information.

5.1 Adaptive local optimization of compute cost - A micro-traffic example

In this example we perform a micro-traffic simulation of a 20km by 20km urban area where cars are making random trips. Each car is an agent, managing its state and adapting its acceleration based on speed regulation and the acceleration of leading cars. The implemented models are based on the Intelligent Driver Model [21], which is a state-of-the-art car following model and the lane-changing model MOBIL (Minimizing Overall Braking Induced By Lane Changing) [13]. This implementation leads to both realistic local behavior and realistic emerging behavior. All cars comply to standard traffic regulations and priority rules.
The environment is represented by a directed graph datastructure. Edges are roads (with single or more lanes) and nodes are junctions. A car agent can interact with the environment by requesting where nearby cars are located. Car agents can also interact with each-other to request acceleration and related information or with traffic light agents to request the state of a traffic light.
During initialization the environment is partitioned based on the number of available cores. The partitioning algorithm is a multilevel recursive algorithm for multi-constraint graph partitioning as presented [12]. It attempts to balance node cost of the graph partitions and minimizes edge cut. A single LP will manage a single environment partition and the agents located in this partition. When agents leave the environment partition they will migrate to a simulator that manages one of the neighboring partitions. At the edges of a partition, car agents require state information of agents that are located in the neighbouring partition. Therefore, we include a synchronization mechanism. This mechanism broadcasts the state of an agent, located at a border area, to neighboring partitions after each state update. In this scenario the cost of a step depends on two activity parameters: Model Computation Cost, MCC and Model Synchronization Cost MSC. In the remainder of this section we elaborate on how we can dynamically load balance these activity parameters using local information only in order to reduce the global step duration.
Optimization algorithm: A significant amount of research has been done in the context of distributed micro-traffic simulation. The load balancing problem is one of the most discussed problems within this context. As stated in [18] it is necessary for all simulation processes to consume similar amount of computing power in order to run at the same speed and the communication among the processes should be minimal. Ramamohanar et. al. [20] introduce a spatial workload balancing approach where they partition the environment in grids. As pointed out by the authors, this approach is static, and unable to react to changes in computational load introduced by agent migrations. Instead, other work, such as Xu et al.'s work that presents an adaptive graph partitioning approach [23]. They essentially execute the graph partitioning algorithm multiple times, on the

entire traffic network, when imbalances are detected. The problem with this approach is that the algorithm runs on the entire network, making a distributed approach difficult.
To solve this problem, we propose a heuristic-based approach, powered by the MAPE-K framework presented in the previous section, that is able to run in a distributed way. The global idea of the algorithm is that we keep track of activity using an activity graph. For example, assuming car agent computational load is homogeneous, we keep track of the number of cars located on the incoming edges of a node. When imbalances are detected between neighboring environment partitions we allow an overloaded partition to migrate a collection of its border nodes and edges to a neighboring, less occupied partition. This is visualized in figure 3. The amount of nodes and edges that gets migrated depends on the amount of activity that needs to be transferred in order to reestablish the activity balance.

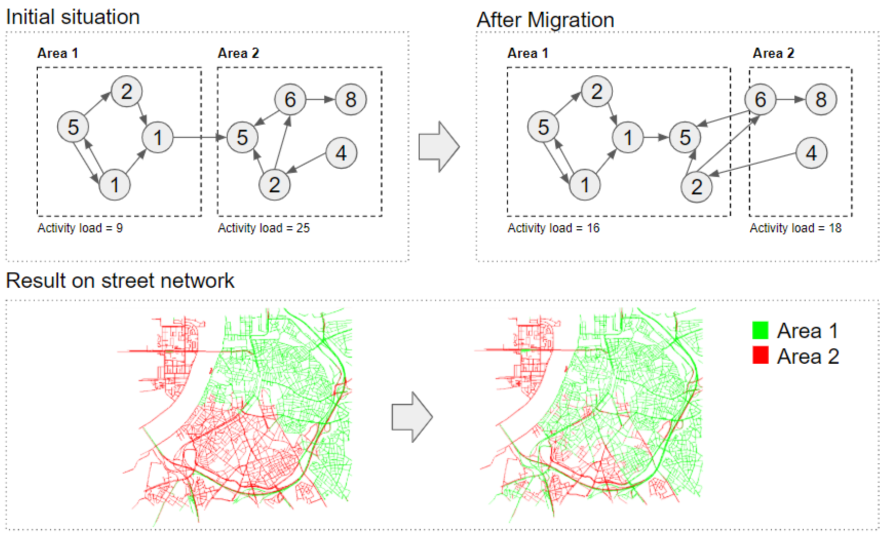

Fig. 3. Heuristic: Load balancing using local activity graphs

Experiment: The implementation of the computational load balance algorithm in the MAPE-K framework is explained below:

1. **Monitor**: We keep track of the global step duration (GSD) and the local step durations of the simulators (LSD^i).
2. **Analyze**: The average LSD is calculated. When one of the LSD exceeds the average by 20% or more, the algorithm evaluates if part of the computational

activity can be offloaded to the neighbors (this is achieved by migrating nodes, edges and agents). If this is the case a ‘migration flag‘ is set.

3. **Plan**: When a migration flag is found, a plan of execution will be created. This plan orders the overloaded area to migrate a given amount of activity to one of its neighboring areas that has been selected in the Analyze step.
4. **Execute**: The overloaded area will calculate which nodes it can offload. Consequently, both the originating area and the destination area will update their graph datastructure accordingly.

We ran an experiment to test this implementation. In the experiment we randomly generate trips in a city center. We introduced an initial imbalance of 1/10. This could be a realistic scenario when people are leaving a residential area to an industrial area in the morning. We expect the algorithm to restore this imbalance over time. Thirty runs of this experiment were executed, the average and standard error are displayed in figure 4. In both graphs we compare a non-adaptive approach with an adaptive approach. The MAPE-K optimization is performed at time-step 250. We observe a significant reduction of step duration when the optimization occurs.

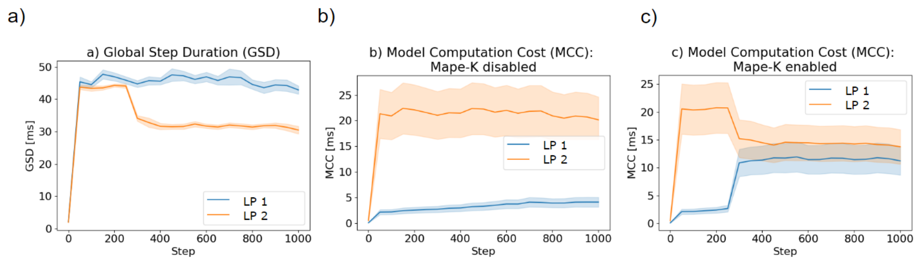

Fig. 4. Results - with and without MAPE-K adaptive optimization, micro-traffic simulation

Balancing Synchronization Cost: As explained in the introduction, the step duration not only depends on Model Computation Cost (MCC). It also depends on Model Synchronization Cost (MSC). The impact largely depends on the scenario. When there is a large amount of traffic at the border areas of environment partitions, the MSC will be significant and cannot be ignored. Therefore, further optimization will be required. We propose a technique that can be explored in future work. The general idea is that we can measure the synchronization cost based on the amount of agents located in a border area. When an imbalance in synchronization cost is observed between areas, we can simulate the synchronization cost after incremental expansion of the graph. This is similar to incremental expansion demonstrated in figure 3. When the synchronization cost of the incremental expansion is lower than the initial cost, we can perform a migration of nodes and edges.

In conclusion, combined with the computational cost balancing heuristic we expect it to lead to a further reduction in step duration. The proposed heuristics will improve upon sub-optimal scenarios where imbalances are observed in neighboring areas, in a scalable and computationally efficient manner. But, it will not be able to reach a global optimization.

5.2 Adaptive local optimization of communication cost - A cellular automata example

In this example we use the agent-based simulation Sugarscape [8] with a cellular automata environment. These simulations typically lead to emergent behaviour and can be used in, for example, biology [2]. In Sugarscape, sugar is grown in each cell of the environment at a certain rate and the goal of the agents is to survive by collecting enough sugar. If an agent cannot satisfy his metabolism, he is replaced by a randomly initiated agent at a random vacant position. The agents are characterised by a metabolic rate and range of sight. At each step they search for sugar by looking in the four perpendicular directions and move one step towards the cell with the highest sugar level, collecting the sugar at their new location. The environment regrows sugar at each step in the cells according to a fixed rate until a maximal sugar level is reached. The model computation cost (MCC) the agent is relatively small but instead the step duration depends mainly on the Local Communicaton Cost (LCC) and the Remote Communication Cost (RCC) (with RCC being significantly more expensive). The RCC is the result of agents that are close to the edge of a simulator and searches within the next simulator for sugar. This consists of two messages that are send between the simulators: one to ask for the amount of sugar on the cells of interest and one with the corresponding answer. Our optimization algorithm keeps track of a communication based activity graph, where imbalances in LCC and RCC between simulators are monitored and dynamically improved by migrating parts of the environment. The MAPE-K cycle is implemented as follows:

1. **Monitor**: The local and remote (both contributions due to received and send messages) compute time for each cell is logged.
2. **Analyze**: Bottleneck simulators are identified by comparing their Local Step Duration (LSD) to the Global Step Duration (GSD).
3. **Plan**: A plan is created to partition certain sections of the simulator's environment to restore imbalances that might have manifested over time. Each section is evaluated using its logged LCC and RCC. During partitioning, the algorithm can decide to migrate a section which will switch LCC to RCC (and possibly vice versa), and as a consequence reduce overall cost.
4. **Execute**: Each simulator executes his part of the established plan and possibly migrates parts of its environment to another simulation process.

To evaluate the performance of the adaptive approach, we ran 30 random initiated Sugarscape simulations and compared them to 30 non-adaptive simulations. Both simulations are initiated with four LP's managing each a quarter

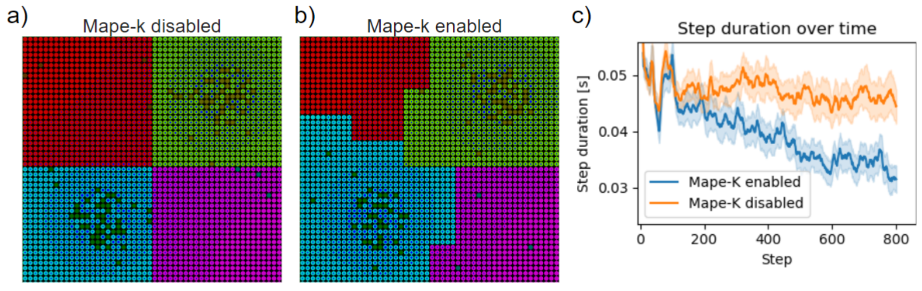

Fig. 5. Results - with and without MAPE-K adaptive optimization, Sugarscape.

of the environment. The simulation stops after 800 steps, the MAPE-k framework is executed every 50th step. The results are illustrated in Figure 5. Image a represent the initiated environment divided over the four LP's. Figure Image b shows the same environment where the borders are optimized based on the current activity. Finally, c illustrates the average step duration and the standard error. From these results we can see that once the random activity of the agents is replaced with emerging behaviour, the adaptive approach improves performance.

6 Conclusion

In this work we presented a MAPE-K loop as a generic and effective framework to implement a self-adaptive distribution for agent-based simulations. We evaluated this framework by implementing two examples: a distributed traffic simulation and a sugarscape simulation. We showed that MAPE-K can be used effictively in multiple simulations to implement adaptivity. Furthermore, the results of the proposed adaptive load-balancing heuristics show a significant reduction in computational cost while being executed decentralized. In future work we will further optimize the presented heuristics and perform an empirical comparison with MAPE-K implementations of state-of-the-art load-balancing techniques. Furthermore, we want to explore the benefits of a hybrid decentralized and centralized adaptive load balancing approach in micro traffic simulation. In this hybrid scenario we envision two MAPE-K loops: 1) a decentralized heuristic, as proposed in this paper, and 2) a centralized load balancing algorithm that is able to find global optimum, as proposed in the related work of section.

References

1. Balmer, M., Cetin, N., Nagel, K., Raney, B.: Towards truly agent-based traffic and mobility simulations. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. IEEE (2004)

2. Baradaran, S., Maleknasr, N., Setayeshi, S., Akbari, M.E.: Prediction of lung cells oncogenic transformation for induced radon progeny alpha particles using sugarscape cellular automata. Iranian journal of cancer prevention **7**(1), 40 (2014)
3. Bosmans, S., Mercelis, S., Hellinckx, P., Denil, J.: Reducing computational cost of large-scale simulations using opportunistic model approximation. In: SpringSim 19
4. Bosmans, S., Mercelis, S., Hellinckx, P., Denil, J.: Towards evaluating emergent behavior of iot using large scale simulation techniques (wip). In: Springsim 2018
5. Boukerche, A.: An adaptive partitioning algorithm for distributed discrete event simulation systems. Journal of Parallel and Distributed Computing **62**(9) (2002)
6. Chan, W.K.V., Son, Y.J., Macal, C.M.: Agent-based simulation tutorial-simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. In: Proceedings of the 2010 winter simulation conference. pp. 135–150. IEEE (2010)
7. D'Angelo, G.: The simulation model partitioning problem: an adaptive solution based on self-clustering. Simulation Modelling Practice and Theory **70**, 1–20 (2017)
8. Epstein, J.M., Axtell, R.: Growing artificial societies: social science from the bottom up. Brookings Institution Press (1996)
9. Franceschini, R., Challenger, M., Cicchetti, A., Denil, J., Vangheluwe, H.: Challenges for automation in adaptive abstraction. In: 2019 ACM/IEEE 22nd int. Conference on MDE Languages and Systems Companion (MODELS-C). IEEE (2019)
10. IBM: An architectural blueprint for autonomic computing. (2006)
11. Iglesia, D.G.D.L., Weyns, D.: Mape-k formal templates to rigorously design behaviors for self-adaptive systems. ACM TAAS **10**(3), 1–31 (2015)
12. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing. pp. 28–28. IEEE (1998)
13. Kesting, A., Treiber, M., Helbing, D.: General lane-changing model mobil for car-following models. Transportation Research Record **1999**(1), 86–94 (2007)
14. Long, Q., Lin, J., Sun, Z.: Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations. Simulation Modelling Practice and Theory **19**(4), 1021–1034 (2011)
15. Macal, C.M., North, M.J.: Tutorial on agent-based modeling and simulation. In: Proceedings of the Winter Simulation Conference, 2005. pp. 14–pp. IEEE (2005)
16. Masad, D., Kazil, J.: Mesa: an agent-based modeling framework. In: 14th PYTHON in Science Conference. pp. 53–60 (2015)
17. Muzy, A., Touraille, L., Vangheluwe, H., Michel, O., Traoré, M.K., Hill, D.R.: Activity regions for the specification of discrete event systems. In: Proceedings of the 2010 Spring Simulation Multiconference. pp. 1–7 (2010)
18. Potuzak, T.: Distributed traffic simulation and the reduction of inter-process communication using traffic flow characteristics transfer. In: Tenth International Conference on Computer Modeling and Simulation. pp. 525–530. IEEE (2008)
19. Raberto, M., Cincotti, S., Focardi, S.M., Marchesi, M.: Agent-based simulation of a financial market. Statistical Mechanics and its Applications **299**(1-2) (2001)
20. Ramamohanarao, K., et. al.: Smarts: Scalable microscopic adaptive road traffic simulator. ACM TIST **8**(2), 1–22 (2016)
21. Treiber, M., Kesting, A.: Traffic flow dynamics. Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg (2013)
22. Van Tendeloo, Y., Vangheluwe, H.: Activity in pythonpdevs. In: ITM Web of Conferences.-Place of publication unknown. vol. 3, p. 01002 (2014)
23. Xu, Y., Cai, W., Aydt, H., Lees, M.: Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic sim. In: WinterSim. IEEE (2014)