Parallel and Distributed Agent-based Simulation of large-scale socio-technical Systems with loosely coupled Virtual Machines

Stefan Bosse

University of Bremen, Dept. Mathematics & Computer Science, Bremen, Germany

Abstract. Agent-based systems are inherently distributed and parallel by a distributed memory model, but agent-based simulation is often characterised by a shared memory model. This paper discusses the challenges of and solution for large-scale distributed agent-based simulation using virtual machines. Simulation of large-scale multi-agent systems with more than 10000 agents on a single processor node requires high computational times that can be far beyond the constraints set by the users, e.g., in real-time capable simulations. Parallel and distributed simulation involves the transformation of shared to a communication-based distributed memory model that can create a significant communication overhead. In this work, instead distributing an originally monolithic simulator with visualisation, a loosely coupled distributed agent process platform cluster network performing the agent processing for simulation is monitored by a visualisation and simulation control service. A typical use case of traffic simulation in smart city context is used for evaluation the performance of the proposed DSEJAMON architecture.

Keywords: Agent-based Computing, Agent-based Simulation, Distributed Simulation, Simulation of large-scale systems

1. Introduction and Overview

Agent models typically model behaviour and interaction that are used intensively in Agent-based simulation (ABS) and modelling (ABM) of complex systems providing a simplified mapping of real-world entity behaviour on a simplified simulation world. Parallel and distributed agent-based simulation (PDABS) aims at reducing the execution time through executing concurrently the agent models distributed on different processors, which is a first-level approach to improve the execution speed and is simplified by the agent memory model. Recent progress of distributed ABS is shown in [1] addressing geospatial systems (traffic). They achieved a speed-up about 10 by using macroscopic interaction models. Agent scheduling and distribution on processors are the main issues to be addressed among communication complexity arising in distributed systems. Load balancing is an important key methodology, too [12]. Agent communication can dominate inner agent computation.

There is an ongoing and increasing interest in the parallelisation of simulations, especially, but not limited to, for large-scale ABS [2]. Although, there is recent progress in utilisation of GPGPU systems for the exploitation of data path parallelism, exploitation of control path parallelism is not addressed and therefore considered in this work. The cellular automata computing model with neighbourhood communication is close to the agent model and can be processed partially and efficiently on GPU systems [3]. Hardware accelerators can be used to speed up special computations for simulation [4]. But in general, agents pose dynamic, irregular, short- and long-range communication that cannot be mapped efficiently on GPU-based processing systems. Distribution of simulations can base on two main principles: Distribution of the environment and distribution of agents [5]. In this work, both principles are addressed.

The processing of simulations using Virtual Machines (VM) with hardware and operating system abstractions shows significant benefits in flexibility and adaptivity required in most simulation scenarios. Therefore, in this work there is a focus of simulations performed on VMs and their parallelisation. Considering large-scale ABS, the large-scale distributed Web with millions of servers and billions of user is an attractive distributed machine for simulation.

The agent model itself poses inherent parallelism due its low degree of coupling to the processing platform and between agents. Interaction between agents commonly takes place with well-defined message-based communication. e.g., by using synchronised tuple spaces. Therefore, the agent model is an inherent parallel and distributed processing model relying on a distributed memory model (DMM) natively. But in simulation worlds a shared memory model (SMM) is often used for efficient and simplified agent interaction and communication. Typical examples for SM-based Multi-agent systems (MAS) are NetLogo [10] or SESAM [11]. Commonly, agent models used in simulation cannot be deployed in real computing environments. In addition to ABS there is Agent-based Computation (ABC), commonly involving totally different agent processing platforms (APP) and agent models.

The staring point of this work is an already existing unified agent model that can be used for ABS and ABC in real-word data processing environments, too [7]. Simulation of MAS is performed by using the same platform for ABS and ABC, the JavaScript Agent Machine (JAM) [9], which can be processed by any generic JavaScript (JS) VM like nodejs or by a Web browser (e.g., spidermonkey). Application of parallelisation to VM is difficult and is limited to some special cases. Most significant barrier for parallelisation in VMs is the automatic memory management (AMM) and garbage collection (GC) prohibiting SMM. Parallelisation is here considered as a synonym for distribution of computation and is no further distinguished in this work.

The JAM platform supports already distributed loosely coupled platform networks, i.e., a set of nodes $\mathbb{N}=\{N_1,N_2,..,N_n\}$ is connected by an arbitrary communication graph $G=\langle\mathbb{N},\mathbb{C}\rangle$ that connects nodes by point-to-point communication channels $\mathbb{C}=\{c_{i,j}\}$. But there is no DMM, agents on different nodes are independent. JAM agents are capable to migrate between nodes (by code and data snapshot check-pointing and migration). This feature implements some kind of a distributed memory virtually, but without any central managing instance or group communication. Basically, an agent carries some isolated region of the distributed memory and memory access is only possible by agent communication (using TS/signals). JAM networks are inherently distributed by strict data and control decoupling, and there are no shared resources among the set of nodes. Up to here we have a well scaling distributed network. Indeed there is no upper bound limit of connected nodes.

The already existing Simulation Environment for JAM (SEJAM) [7] extends one physical JAM node with a visualisation layer and simulation control and enables simulation of real-world JAM agents situated in an artificial two-dimensional geo-spatial simulation world. Additionally, the JAM node of SEJAM can be connected to any other external JAM node providing real-world-in-the-loop simulation (i.e., agents from a real-world vehicle platform connected via the Internet can migrate into the simulation world and vice versa!). Virtualisation of JAM nodes enables simulation of JAM networks by SEJAM. In contrast to pure computational JAM networks the simulator couples its JAM nodes by shared memory (SM) tightly and is connected to all parts of the JAM node including direct agent access. Transforming this SM to a distributed memory (DM) architecture would cause significant Interprocess-Communication (IPC) costs by messaging limiting the speed-up.

In this work three main strategies are applied and evaluated to provide an almost linear scaling of the speed-up for large-scale distributed simulations:

1. Strict decoupling of visualisation and simulation control from computation (of agents and platforms);
2. Adding a Distributed Object Memory layer (DOM) to the existing JAM platform to enable distributed but coupled JAM node networks with distributed shared objects and virtualisation;
3. Mapping of simulation entities (virtual platforms and agents) on multiple coupled physical platforms by preserving spatial and communication context (environmental and agent distribution based on principle discussed in

[5]);

In [6] a basically similar approach was applied to large-scale agent systems using signal and message communication for node coupling, but limited to local agent interaction. The approach presented in this work poses no communication range limitations. To understand the challenges and pitfalls of different approaches a short introduction of the underlying agent model, its original JAM, and the simulator SEJAM is required, discussed in the next sections.

The novelty of this work is a the deployment of an unified agent platform for distributed computation and simulation supporting large-scale ABC/ABM/ABS based on an optimised virtualisation layer using distributed shared object memory. A simulation monitor providing visualisation and control is attached to the distributed JAM network. Different mapping methods are introduced and compared. Agents can migrate seamless between virtual and real world.

2. Parallel and Distributed Simulation

2.1 Parallel and Distributed Agent Processing Architecture

The core of the JAM agent processing platform is the Agent Input-Output System (AIOS). The AIOS provides a set of modules and operations that can be accessed by the agents. First part of virtualisation of the JavaScript code execution is a sandbox environment and the agent scheduler. Second part are virtual JAM nodes (vJAM). Each vJAM is a virtual isolation container with its own tuple space and agent process table. An agent can only interact with other agents via the AIOS by:

a. Anonymously storing and extracting data via a synchronised tuple space data base (generative communication);
b. Addressed sending of signal messages to other agents (mainly used by parent-children groups);
c. Agent control supporting creation and termination of agents including replication.

In this section three different JAM platforms modes are considered:

1. **pJAM**. A physical JAM node executed by a generic JS VM;
2. **vJAM**. A virtual JAM node providing virtualisation of a pJAM and attached to and executed by a pJAM node;
3. **dJAM**. A distributed JAM node part of a JAM cluster network providing coupling by Distributed Object Spaces (including a distributed simulation world).

The original JavaScript Agent Machine (JAM) is a portable processing platform for reactive state-based JavaScript agents [7]. JAM is programmed entirely in JavaScript as well as the agents. This enables the deployment of the platform on arbitrary computing devices including WEB browsers and the integration in any existing software just requiring an embedded JavaScript interpreter. JS execution is strictly single-threaded with only one control flow preventing any parallelisation on JS level except asynchronous IO operations that can be scheduled in parallel. One major feature of JAM agents is their mobility. An agent can migrate at any time its snapshot to any other JAM node preserving the code, the data, and control state. The distributed dJAM node is sub-classified in tightly coupled pJAM nodes (dpJAM) sharing most of their data structures, and in coupled vJAM nodes (dvJAM) sharing only a sub-set of platform data structures. Each pJAM node has a scheduler with a scheduling loop iteration over the set of associated vJAM nodes. All agents of a vJAM node are proecssed sequentielly by executing the next activity of an agent. Any remote operations (always asynchronous operations) are queued and proecssed after the first run of the scheudling loop finished (two-phase processing).

2.2 Distributed Shared Objects

Virtual nodes can be clustered to a distributed virtual node by sharing:

1. Tuple Space (copy-on-demand by RPC with event notifications);
2. Process Table (replicated on all distributed nodes, strict consistency required);
3. Auxiliary objects by higher software layers managed by a single central instance, e.g., the graphics layer of simulator (P2P RPC)
4. Agent data (Direct Agent Access, copy-on-demand by RPC, no consistency required).

Shared objects are managed by a virtual node object manager (OM) by using point-to-point communication and FIFO message queues. Access of shared objects involves multi-cast group communication emulated by single point-to-point messages. To satisfy consistency of data objects, group communication (implemented by multiple point-to-point messages) must be totally ordered. For this reason, one node of the group must act as a sequencer.

Shared objects are modified by:

1. A one-phase update protocol;
2. A two-phase primary-copy update protocol;
3. An invalidation protocol; or
4. By a P2P RPC protocol.

using point-to-point messaging provided by DAMP and managed by the sequencer to broadcast the messages to all group members. Tuple space output store operation are not replicated to all group member tuple spaces (copy-on-demand). Centralised stored and handled objects like the modification of the graphical simulation world is performed directly by RPC messages. Tuple space communication provides no guarantee for completion and reliability. For example, agents can concurrently try to remove a tuple, but only one agent will succeed.

Some shared objects are updated lazily and directly by the group members without using the broadcast sequencer, i.e., data consistency is not guaranteed, but not being relevant in this case. To support distributed object memory and group communication, the AMP is extended by a **Distributed AMP** (DAMP) using either P2P RPC or multi-cast messages with a sequencer with the following (low-level) messages supporting distributed shared objects (DSO), based on a ordered multi-cast message implementation using point-to-point messages [8]: `JOIN, B-REQUEST, BROADCAST, RETRANS, PHASE1, PHASE2, ACK`, supporting one- and two-phase object update and invalidation including lazy updates. Finally, DAMP supports (high-level) access to a distributed region-segmented simulation world (via the DSOM, mainly shapes, resources, platforms): `ADD, ASK, REM, SEARCH, MIGRATE`. DAMP is commonly handled on separate communication ports spawning a JAM platform network.

2.3 Distributed Shared Object Manager (DSOM)

The object manager (OM) is responsible for the underlying object sharing management and messaging. But the original JAM architecture and API would require a significant redesign of the entire JAM architecture to support DMM via the OM. Since DMM is only required for simulation and JAM should be deployed in simulation and computer networks without modification, a distribution shared object manager (DSOM) connects to the JAM code on demand (i.e., if JAM is part of a simulator or simulation network). The DSOM extends and exchanges parts of the JAM code to connect to the JAM supporting DMM and vJAM migration. To reduce communication, a log-based approach with P2P RPC communication is used to update a global object map. The log contains changes of distributed objects. Agents typically access spatially bounded data only.

2.4 Simulation Environment for JAM

The core of the distributed simulation environment is the already existing Simulation Environment for JAM (SEJAM) [9], which consists basically of a physical JAM node (pJAM) with a visualisation and simulation control layer. In SEJAM; each JAM agent is associated with a visual shape visible in a two-dimensional simulation world, shown in Fig. 2 (left). The simulator extends the AIOS of JAM with additional APIs, beside visualisation, most important, a NetLogo-like shared memory model enabling spatial and pattern search and modification (i.e., direct agent access), and access of data storage (e.g., SQL data bases).

SEJAM supports physical and computational agents (introduced in Sec. [Agent-based Modelling, Computation, and Simulation]). A physical agent is always bound to its own vJAM and cannot migrate to another vJAM. Computational agents (processed in simulation and computer networks) can migrate between different vJAM nodes on different pJAM nodes. Note that SEJAM processes the JAM node and the simulation layer by one JS VM in one computational process. There are basically three distribution strategies that could be applied (see Fig. 1):

a. Each simulation vJAM is moved to a remote pJAM node;
b. One simulation vJAM node is distributed across multiple remote pJAM nodes;
c. Spatial regions of simulation vJAM nodes are mapped on single pJAM nodes.

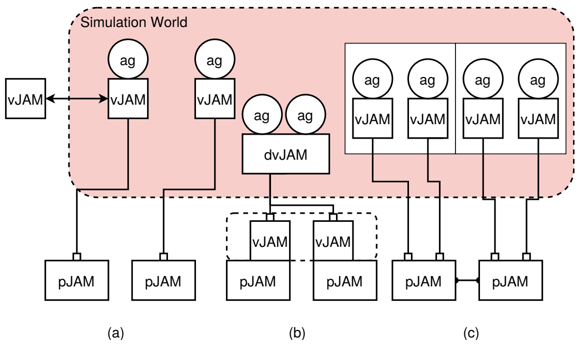


Fig. 1. Distribution strategies: (a) Each virtual JAM node is assigned to a different physical node (b) A distributed virtual JAM node (dvJAM) is created by distributing agent processing on different physical nodes (c) in Regions of agents with their vJAM nodes are distributed on different pJAM nodes by preserving local context

Beside the computational parallelisation of the agent processing, the agent-simulator interface (e.g., operations that modify the agent shape visualisation and simulation worlds) and extended APIs provided by the simulator have to be considered in the parallel and distributed simulator architecture. The extended NetLogo-like API for physical agents maps a shared memory model on the multi-agent system, i.e., agents can access other agent data and control state directly (Direct Agent Access, DAA). All DAA operations must be forwarded to remote JAM nodes by using DAMP primitives and the OM, too. The distribution of DAA (i.e., basically the powerful NetLogo *ask* operation) is a challenge and non-trivial, and can result in high computational costs. Grouping of vJAM nodes with a spatial context (region of the simulation world) on the same pJAM node can reduce communication significantly.

2.5 Distributed Simulation and Monitoring

In this work, parallelisation is always implemented by message-based communicating processes, i.e., parallelisation is an equivalent of distribution. The unified distribution approach does not distinguish between (computer) local and remote network operations. To reduce communication complexity and costs a new approach is proposed using a distributed JAM network cluster for simulation without a tightly coupled simulator, shown in Fig. 2. The simulation and visualisation layer is only loosely coupled and acts as a monitor and a controller. Since physical agents situated in a graphical simulation world can change the simulation world, a backing-store architecture with a change log is proposed. Each pJAM node represents a region of the simulation world with a backing-store layer that maps the visual world with visual shapes representing agents, virtual platforms, and resources (with data). Additionally, agents can get information from the backing-store, i.e., resources and their constraints, e.g., streets of a city map. The simulator will periodically collect changes to update the real visual representation of the world and vice versa via P2P RPC messages. Each region can access any other region, too.

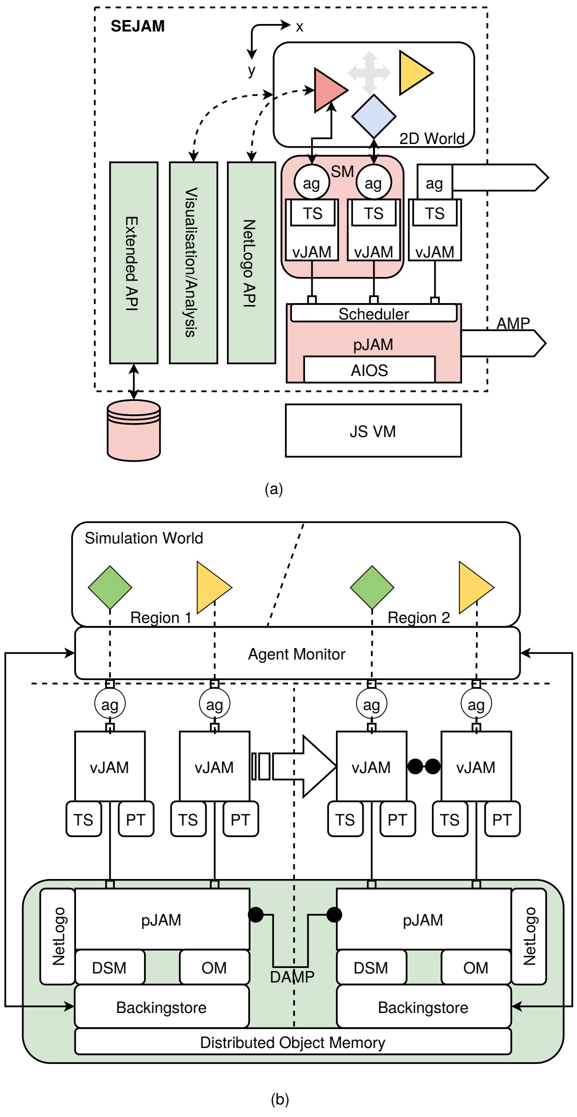


Fig. 2. (a) Original monolithic SEJAM simulator with one pJAM and multiple vJAM nodes (b) Distributed simulation network consisting of coupled and extended pJAM networks. The simulator monitors the agents and its visual backing-store

Each pJAM is extended by typical simulation APIs, most of all the NetLogo API. Depending on the NetLogo operation and the involved set of entities (agents, resources), the operation can be performed by local computation and data, or by additional communication with remote pJAM nodes via DAMP. The location and current pJAM node association of physical agents is represented by a distributed object table (with a local copy on each pJAM node with a one-phase update protocol).

Physical agents can migrate from one region to another requiring the migration of the entire associated vJAM, but a vJAM is just an administrative data structure including the tuple spaces (significant payload). All simulator-pJAM/vJAM and vJAM-vJAM IPC is performed using the DAMP. The vJAM nodes can be linked (ad-hoc and dynamically) via AMP, too. If the nodes are on the same pJAM node this link can be virtual (direct object and message passing), If the vJAM nodes are located on different pJAM nodes the virtual link must be replaced by a communication channel via DAMP.

There is still one central pJAM node part of the DSEJAMON architecture providing a multi-cast sequencer and a central simulation clock controlling single-stepping of all nodes synchronously. The synchronous stepping of all regions is important to reduce multi-cast messaging and synchronisation between regions. A simulation step is atomic. A logical world agent is used to control the simulation, accessing data bases, computing and distributing sensors, monitoring agents, and finally providing sensor input to the physical agents via tuple spaces. This requires remote tuple space operation requests implemented by DAMP via P2P messages. The simulation world is distributed in regions with a local backing store and log-based modification tracking avoiding expensive synchronised distributed shared memory. Nodes are connected via UDP (remote and local IPC) or UNIX socket (local IPC only) channels.

3. Evaluation and Use-Case Traffic Simulation

The last section considered only the number of messages to implement various distribution approaches. Network latency, data bandwidth, and message volume have a significant effect on the real speed-up that can be achieved. The largest contributions to communication costs are vJAM migration and remote simulation world access. To evaluate the new distributed simulation environment, a typical use-case with a large number of computational intensive agents is investigated. The main question is the scaling and overall performance (speed-up) of the distributed parallel simulation compared with the classical sequential single-instance simulation.

There are some major parameters to be considered:

1. The distribution of sensor data and monitoring of the simulation world via the backing store and change log;
2. The communication time of agents with the distributed simulation world (remote set operations) and vJAM migration;
3. The communication time for shared objects;
4. Scaling efficiency of the underlying communication and computer architecture (e.g., w.r.t. memory architecture);
5. The impact of distribution on memory management (garbage collection) of the JS VM (reducing memory pressure).

Traffic simulation typically involves a large number of agents representing vehicles, drivers, and passengers. To get reasonable results, MAS with more than 10000 agents have to be simulated. An agent represents a vehicle/driver group with its own vJAM. Even with simplified agent behaviour models the processing on one processor is a challenge. An already performed traffic simulation from [9] was used and compared with the parallelised simulator. In this simulation independent driver agents should learn long-range navigation by computationally intensive reinforcement learning. A typical simulation run with the original one-process/pJAM SEJAM requires 10 minutes / 10000 simulation steps. The distributed simulation maps regions of the artificial city with thousands of vJAM nodes on remote pJAM nodes, each associated with a region of the simulation world, shown in Fig. 3. The pJAM nodes are operating in single-step mode controlled by a clock of the central simulation node. Each region can access other regions pJAM nodes directly via P2P RPC communication via DAMP. Physical agents can access all regions of the world via DAMP requests. Each region contains shapes representing agents and resources spatially organised by an r-tree for fast search. Some resources like streets are distributed across multiple regions.

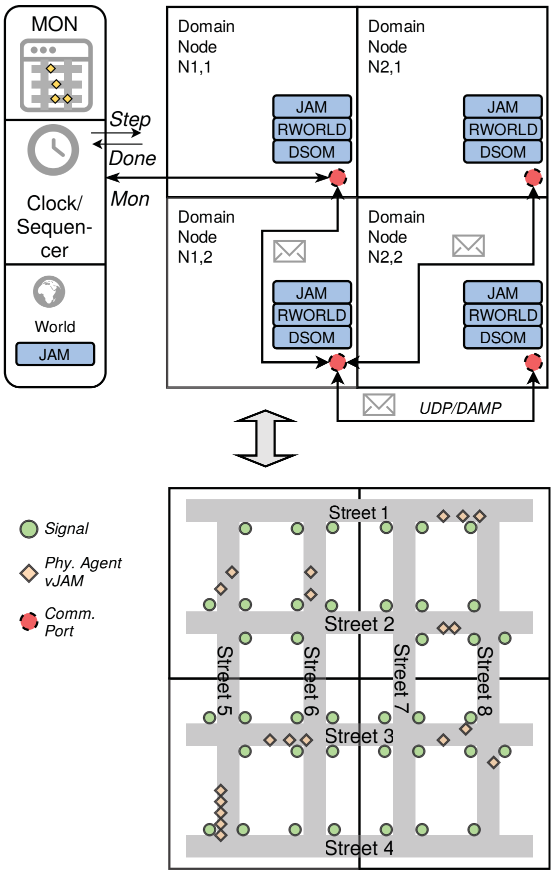
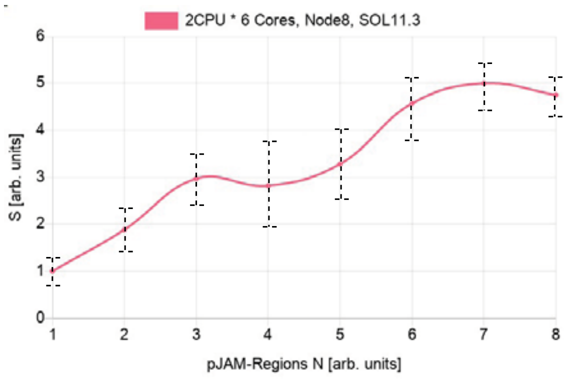
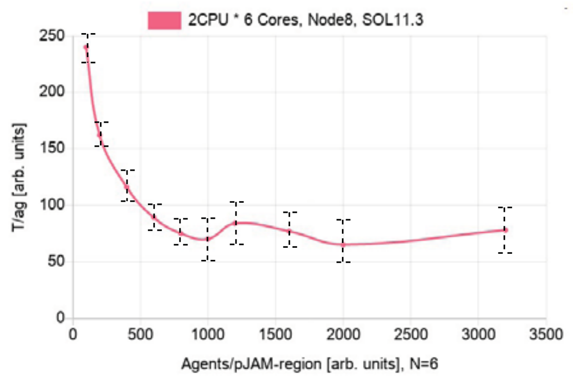


Fig. 3. Example segmentation of the traffic simulation world (art. city with streets, signals, and vehicle agents) that is partitioned into four regions, each associated to a vJAM node (DOM: Distributed Shared Object Manager).

Currently a speed-up up to 5 with a scaling of about 70% can be achieved with a typical communication overhead depending on computer architecture and the network bandwidth and latency. The parallel distributed simulation was evaluated on a two CPU / 6 Core per CPU workstation.



(a)



(b)

Fig. 4. Results of the evaluation of the parallel traffic simulation. (a) Measured speed-up with respect to number of regions/pJAM nodes (b) Scaling of computation time with respect to agents/region with 6 pJAM regions (summarising all nodes and 10000 simulation steps)

Fig. 4 shows some preliminary results with respect to the measured speed-up of the traffic simulation for 10000 simulation steps of the traffic simulation (the agent populations was about 1000 agents) and the scaling of the speed-up with respect to the number of agents per region (and pJAM node). A reasonable speed-up can only be achieved if there is a significant number of agents per region with a significant amount of computational load. Communication was composed of message transfers between region-region and region-controller nodes.

4. Conclusion

Distribution and parallelisation of large-scale ABS is still a challenge due to the increasing communication complexity that can annihilate a possible speed-up (below 1). A originally shared-memory simulator could not be distributed efficiently by separating visualisation and agent computation. Another approach starting with an isolated distributed JAM network mapping simulation world agents and their vJAM nodes on multiple pJAM nodes by preserving spatial constraints (zones) could achieve a significant speed-up by moderate communication costs. Originally a computational multi-agent system can be efficiently mapped on a distributed processing network, but simulation overlaps a shared memory model hard to be distributed efficiently afterwards.

The performance of the proposed distributed simulation architecture still depends on the micro- and macro-scale modelling. Long-range communication and interaction of agents (including agent search and DAA operations) can increase the communication overhead significantly and lower the speed-up depending on the communication bandwidth. With increasing number of remote pJAM nodes (each associated with a spatial region of the simulation world) the scaling efficiency decrease further, again due to increasing communication between regions and more migrations of agents/vJAM nodes. The scaling for large networks has to be further investigated and WEB browsers as pJAM nodes must be considered in future work.

5. References

[1] M. Mastio, et al. *Distributed agent-based traffic simulations*, IEEE Intelligent Transportation Systems Magazine 10.1, pp. 145-156, 2018

[2] B. G. Aaby, K. S. Perumalla, S. K. Seal, *Efficient Simulation of Agent-Based Models on Multi-GPU and Multi-Core Clusters*, in SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain, 2010

[3] P. Richmond, S. Coakley, D. M. Romano, *A High Performance Agent Based Modelling Framework on Graphics Card Hardware with CUDA*, in Proc. of the AAMAS 2009

[4] J. Xiao, P. Andelfinger, D. Eckhoff, W. Cai, A. Knoll, *A Survey on Agent-based Simulation using Hardware Accelerators*, 2018.

[5] O. Rihawi, Y. Secq, P. Mathieu, (2014). Effective Distribution of Large Scale Situated Agent-based Simulations.In Proceedings of the 6th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART, 312-319, 2014 , ESEO, Angers, Loire Valley, France

[6] D. Šišlák, P. Volf , M. Pěchouček (2009) Distributed Platform for Large-Scale Agent-Based Simulations. In: Dignum F., Bradshaw J., Silverman B., van Doesburg W. (eds) Agents for Games and Simulations. AGS 2009. Lecture Notes in Computer Science, vol 5920. Springer

[7] S. Bosse, U. Engel, *Real-time Human-in-the-loop Simulation with Mobile Agents, Chat Bots, and Crowd Sensing for Smart Cities*, Sensors (MDPI), 2019, doi: 10.3390/s19204356

[8] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, *An efficient reliable broadcast protocol*, SIGOPS Oper. Syst. Rev., 1989.

[9] S. Bosse, *Self-adaptive Traffic and Logistics Flow Control using Learning Agents and Ubiquitous Sensors*, Proc. of the 5th International Conference on System-Integrated Intelligence Conference, 11.11-13.11.2020, Bremen, Germany, 2020

[10] S. Tisue, U. Wilensky, *Netlogo: A simple environment for modeling complexity*. International conference on complex systems. Vol. 21. 2004.

[11] F. Klügl, R. Herrler, M. Fehler, *SeSAm: implementation of agent-based simulation using visual programming*. Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. 2006

[12] G. Cordasco, C. Spagnuolo and V. Scarano, *Work Partitioning on Parallel and Distributed Agent-Based Simulation*, in 2017 IEEE International Parallel and Distributed Processing Symposium: Workshops (IPDPSW), Lake Buena Vista, FL, 2017 pp. 1472-1481. doi: 10.1109/IPDPSW.2017.87