DSim – A distributed message-passing mobsim implementation

Janek Laudan, Simunto GmbH, laudan@simunto.com
Christian Rakow, Simunto GmbH, rakow@simunto.com
Marcel Rieser, Simunto GmbH, rieser@simunto.com
Steffen Axer, Volkswagen AG, steffen.axer@volkswagen.de
Hannes Rewald, Volkswagen AG, hannes.rewald@volkswagen.de

Motivation

As Multi-Agent Transport Simulation (MATSim) (Horni et al., 2016) models continue to evolve, there is a growing demand to enhance their scale and level of detail. At the same time, Central Processing Unit (CPU) clock rates have plateaued for nearly two decades, while new processor generations increase parallel computing capabilities (Leiserson et al., 2020). Realizing performance improvements to accommodate computationally expensive simulations while maintaining fast execution times requires leveraging the parallel computing capabilities of modern computer hardware.

MATSim performs three main phases during each iteration: (1) the mobility simulation (mobsim), where synthetic persons execute their daily plans; (2) scoring, where executed plans are evaluated; and (3) replanning, where a fraction of the synthetic population invents new plans to be executed in the next iteration. Of the three phases outlined, scoring and replanning are relatively easy to scale onto parallel computing hardware, as the necessary computations are independent for each agent. In contrast, parallelizing the mobsim is not trivial, as synthetic persons and vehicles interact in the simulated reality of the mobsim.

The current mobsim implementation, QSim, uses a shared-memory algorithm to parallelize the simulation computation, effectively scaling up to eight processes (Dobler and Axhausen, 2011, Graur et al., 2021). Modern computing hardware features significantly more processors, requiring a new parallelization strategy. In our previous work, we have implemented a prototype featuring a message-passing approach for the mobsim (Laudan et al., 2025). The presented prototype is implemented in the programming language Rust, to provide compatibility with high-performance computing (HPC) infrastructure.

To make the findings from the prototype implementation of the distributed mobility simulation applicable to the MATSim framework, it is necessary to integrate them into the existing MATSim code-base, which is written in Java. Building on the ideas presented in Laudan et al. (2025) and Cetin et al. (2003), we present the message-passing based mobsim implementation **DSim**.

Methodology

The DSim implementation distributes the simulation work across available processes using domain decomposition. During this process, the MATSim network is divided into network partitions, which are assigned to a corresponding number of simulation processes. Each process executes the traffic simulation on one particular partition without interference from other processes. Neighbor partitions are connected by split links which start and end on different partitions. Vehicles that enter a split link are transmitted to the process that manages the downstream end of the link as a message. Similarly, capacity updates are communicated to the process managing the upstream end of the link in the form of a message. This architecture requires communication only between neighbor partitions in the simulation network, limiting the number of messages being sent during each time step.

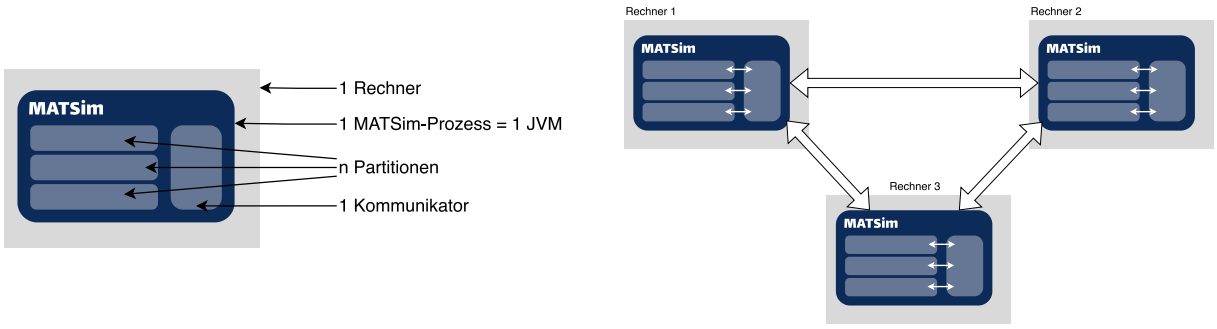

Figure 1: Overall architecture of DSim

The distribution of simulation work is implemented using a flexible architecture that allows executing the simulation on a single compute node, as well as distributing the simulation across multiple compute nodes. As shown in Figure 1, when executed on a single node, multiple simulation processes are executed on this node. The communication between simulation processes is managed by one communicator, which passes messages in-between processes. In the case of a distributed execution of the simulation shown on the right in Figure 1, multiple instances of the application are started, each containing multiple simulation processes. The communicator in each instance manages the inter-process communication by either dispatching messages to local processes, or by sending messages to the corresponding compute node that manages the receiving simulation process.

The DSim mobility simulation is compatible with existing infrastructure in the simulation framework, as it uses existing interfaces and extends them where necessary. Additionally, the simulation is backward compatible to the current event processing system, which allows re-using existing event handler code.

Results

The scalability of the DSim implementation is tested on a computing cluster using an Intel® Xeon® Platinum 9242 Processor with a clock frequency of 3.8 GHz and a core count of 48. Figure 2 shows the real-time ratio (RTR) to compute the Open Berlin Scenario using the default QSim and the new DSim mobsim implementation.

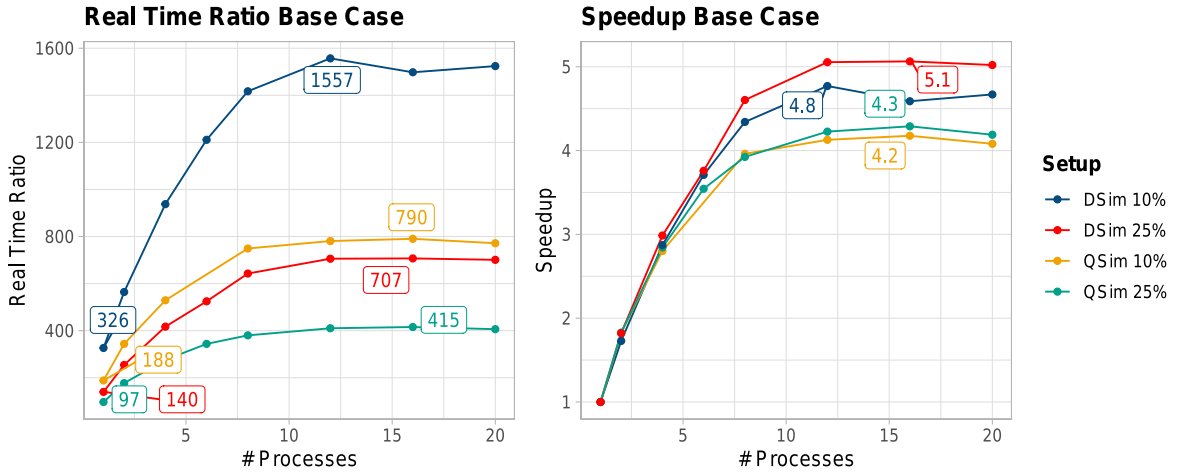

Figure 2: Real-time ratio for computing the Open Berlin Scenario with the QSim and with the DSim implementation on the left; Comparison of relative speedups on the right

For both, the 10% and the 25% sample size, the new implementation achieves two times higher RTR compared to the standard QSim. That said, the scalability of the new implementation indicated by the relative speedups shown on the right in Figure 2 is only slightly better compared to the QSim. Detailed benchmarks show that the traffic simulation of the DSim implementation scales very well. Still, further speedups are limited by the existing events processing system, which is not yet adapted for distributed execution of the simulation. In particular, the scoring functionality must be adapted to allow for better scalability on larger process counts.

Similarly, it is shown how extensions like the demand responsive transit (DRT) contrib can be included into the new message-passing architecture. The runtime investigations also include a DRT simulation and identify the optimizer module as a bottleneck for scalability. The current DRT optimizer implementations assume full access to the entire simulation domain at any time, which is not a good match for a distributed simulation architecture. Further research is needed on how to re-formulate this problem to match the newly introduced architecture.

Conclusions

The presented distributed computing architecture for MATSim provides minor speedup improvements. This is partly because the new architecture reuses much of the existing functionality, which has not yet been fully optimized for parallel execution. Further optimizations of these components are already planned and should mitigate the identified bottlenecks in scalability. Even in its current state, the new message-passing architecture allows for horizontal scaling of MATSim scenarios onto multiple compute nodes, making it possible to simulate larger models than before.

References

Cetin, N., Burri, A., Nagel, K., 2003. A large-scale agent-based traffic microsimulation based on queue model, in: IN PROCEEDINGS OF SWISS TRANSPORT RESEARCH CONFERENCE (STRC), MONTE VERITA, CH.

Dobler, C., Axhausen, K.W., 2011. Design and implementation of a parallel queue-based traffic flow simulation.. https://doi.org/10.3929/ETHZ-B-000040273

Graur, D., Bruno, R., Bischoff, J., Rieser, M., Scherr, W., Hoefler, T., Alonso, G., 2021. Hermes: Enabling efficient large-scale simulation in MATSim. Procedia Comput. Sci. 184, 635–641.. https://doi.org/10.1016/j.procs.2021.03.079

Horni, A., Nagel, K., Axhausen, K.W., 2016. The Multi-Agent Transport Simulation Matsim. Ubiquity Press.. https://doi.org/10.5334/baw

Laudan, J., Heinrich, P., Nagel, K., 2025. High-performance mobility simulation: Implementation of a parallel distributed message-passing algorithm for MATSim. Information (Basel) 16, 116.. https://doi.org/10.3390/info16020116

Leiserson, C.E., Thompson, N.C., Emer, J.S., Kuszmaul, B.C., Lampson, B.W., Sanchez, D., Schardl, T.B., 2020. There's plenty of room at the Top: What will drive computer performance after Moore's law?. Science 368.. https://doi.org/10.1126/science.aam9744