This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Debner, Anton; Hyyppä, Matias; Hanhiova, Jussi; Hirvisalo, Vesa

Scalability of a Machine Learning Environment for Autonomous Driving Research

Published in:

Proceedings - 2019 IEEE 17th International Conference on Industrial Informatics, INDIN 2019

DOI:

[10.1109/INDIN41052.2019.8972278](https://doi.org/10.1109/INDIN41052.2019.8972278)

Published: 01/01/2019

Document Version

Peer reviewed version

Please cite the original version:

Debner, A., Hyyppä, M., Hanhiova, J., & Hirvisalo, V. (2019). Scalability of a Machine Learning Environment for Autonomous Driving Research. In *Proceedings - 2019 IEEE 17th International Conference on Industrial Informatics, INDIN 2019* (pp. 687-692). [8972278] (IEEE International Conference on Industrial Informatics (INDIN)). IEEE. <https://doi.org/10.1109/INDIN41052.2019.8972278>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Scalability of a Machine Learning Environment for Autonomous Driving Research

1st Anton Debner
Department of Computer Science
Aalto University
Espoo, Finland
anton.debner@aalto.fi

2nd Matias Hyyppä
Department of Computer Science
Aalto University
Espoo, Finland
juho.hyyppa@aalto.fi

3rd Jussi Hanhiova
Department of Computer Science
Aalto University
Espoo, Finland
jussi.hanhiova@aalto.fi

4th Vesa Hirvisalo
Department of Computer Science
Aalto University
Espoo, Finland
vesa.hirvisalo@aalto.fi

Abstract—We study scalability of machine learning environments in the context of mixed collaborative driving. Mixed collaborative driving includes both human controlled vehicles and vehicles controlled by AI (Artificial Intelligence) that share the physical road resources (*e.g.*, intersections and roundabouts). Many such driving situations cannot be easily created nor replicated in the real life. Therefore, development and testing of AI systems is often done with simulators.

Machine learning environments must maintain a real-time understanding of their traffic situation. Scaling of the machine environment to multiple distributed nodes is required to support larger number of participating vehicles. Our experimental environment consists of the CARLA simulator, custom AI implemented with the TensorFlow framework, and a corner case search subsystem. With the corner case search subsystem we can automatically evaluate the AI in different driving scenarios. In this paper, we present how scaling of the environment to multiple distributed nodes affects its performance.

Index Terms—Simulation, Hybrid Systems, New Control Applications

I. INTRODUCTION

In this paper, we address scalability of machine learning environments that are based on distributed real-time traffic simulation. The recent development of AI is enabling many new applications including autonomous driving. The AI systems for autonomous driving typically process sensor data in real-time to maintain an understanding of their ever-changing traffic situation [3], [10], [15].

Autonomous driving requires robotic drivers that have their AI systems on-board the vehicles and smart mobility support services deployed on the road infrastructure, *e.g.*, in the Fog computational nodes [13]. Many driving situations require the

Fig. 1. AI systems can use object detection CNNs to estimate the state of their surroundings. Additional AI logic is required to, *e.g.*, distinguish between moving and stationary vehicles and do predictions. The picture of the left presents a real life view and picture on the right presents a simulated view. Both have CNN-based detections overlaid on the images. Our AI system works similarly for both simulated or real scenes.

participating vehicles to share the physical road resources, *e.g.*, driving through a roundabout in rush hours. Accurate environment recognition, low-latency decision making, and driving rules are required to handle the situations. Additional challenge is introduced when both robotic and human drivers need to interact in the driving situations.

Developing AI systems that are capable of interacting with humans requires special development and testing environments. We see photorealistic simulation (see Figure 1) as a possible solution for developing mixed multi-AI-human driving scenarios. In our approach, both robotic and human drivers can take part in a shared simulated model world.

The main requirement for such a system is that we can scale real-time sharing of the model state and the sensor data rendering to multiple clients. We need to run the simulation in real-time to make evaluation of human reactions possible, and to be able to evaluate real autonomous driving software on real hardware platforms.

In addition to scaling the simulation, special care needs to be taken into timing of computation and communication. Realistic driving situations using shared road resources require that all the participating entities receive a consistent view to the model state (*i.e.*, the shared world).

Our research problem relates to the fundamental problems with distributed real-time computation with a shared state. State sharing is required to allow multiple parties in a simulation. State sharing adds computation and communication latencies leading to uncertainty in the timing of events. Clock synchronization is required to compare when different entities take their actions and to reason about the performance and behavior of the AI systems.

There exists several open-source autonomous driving simulators, such as TORCS [16], AirSim [14] and CARLA [5]. We have used the CARLA simulator as the central component of our environment. The CARLA simulator is able to simulate traffic situations and produce photorealistic video streams suitable for machine learning systems. CARLA uses the Unreal Engine 4 game engine [6] as its basis. Video stream data produced by the simulator can be consumed by AI clients that use, *e.g.*, TensorFlow [7] based CNN (Convolutional Neural Network) inference as part of their decision making. The simulated entities receive sensor data from the virtual world using CARLA client interface and make vehicle control commands based on the perceived environment. Machine learning models have been shown to be able to perform similarly in simulated and real environments [2], [4].

In this paper, our contribution is to present a way to distribute the heavy scenery rendering load to multiple distributed computing nodes in a scalable way. Our corner case search system is able to handle the distributed computation related timing challenges, thus enabling AI system evaluation also in the time domain.

The structure of this paper is the following. We begin our presentation by describing our environment and the architecture of the simulator that is the central component of the environment. After that, we describe our distributed computation and how we approach the many timing problems that are inherently present. We continue our presentation by describing the setup of our scalability experimentation and our results with the setup. We end the paper with our conclusions following a short discussion of our work.

II. MACHINE LEARNING ENVIRONMENT

Using our environment, we can evaluate the behavior of real autonomous driving software and hardware, as-well-as human driver reactions and behavior. The traffic environment and the vehicles for the drivers (both robotic and human drivers) are simulated, and the simulation operates in real-time.

Figure 2 illustrates the usage of the environment. The drivers connect to the clients created by the simulator. Each of the simulator clients act as an interface to a vehicle in the simulated world. On the left-hand side, there is an example of a human client setup. On the right-hand side, there is an example of a typical hardware platform for a robotic driver. Robotic drivers

Fig. 2. Shared state simulation allows multiple drivers to drive in the same model world. Drivers can be human (left: typical driving station set-up) or they can be robotic drivers (right: NVIDIA Jetson TX2 that includes a typical on-board hardware for running autonomous driving software).

Fig. 3. An overview to the machine learning environment. The simulation system consist of the CARLA server, multiple clients, the corner case search subsystem and a simulator driver. The related on-board and off-board subsystems communicate with the simulation.

can receive their view into the simulated world via a network connection or by looking at the same display as the human drivers.

For understanding the driver performance, our system compares the actions of the driver against the ground-truth given by the simulator. Especially for the robotic drivers, the system can compare their AI perception of the world directly against the simulator ground-truth.

The simulator ground-truth enables the operation of our corner case search subsystem, which is illustrated in Figure 3. Using the comparison, we can spot situations in which the AI systems are not working properly, *i.e.*, their estimate of the world state differs significantly from the ground truth state of the world.

The human and robotic drivers connect to the clients of the CARLA simulator. The CARLA server receives vehicle control commands, and updates the world model state accordingly. It then renders sensor data streams and sends them to the connected clients. Clients make vehicle control decisions based on information extracted from the sensor data and transmit control commands to the CARLA simulator.

In addition to on-board control, collaborative driving systems need information on the traffic situation. Figure 4 presents a typical setup of communication links, data streams and processing. In vehicular applications, placement of the computation is central. Large data centers are too slow and vehicle systems too limited to handle the load related to collaborative driving.

Fig. 4. In vehicular machine learning systems, there are heavy data streams and plenty of learning stored in deep networks. Autonomous driving AI systems are needed in individual vehicles for perceiving the environment and to control the vehicles. Autonomous driving support systems are placed in proximity of the vehicles to minimize the communication latencies. Fog placed AI support systems can be, *e.g.*, stateful services that process and share data among connected vehicles, or stateless services that offer accelerated computing facilities. AI system refining can be done offline in larger datacenters with more computational capacity.

Fig. 5. Basic high level architecture of CARLA. The simulation is running on a single computer on top of the Unreal Engine. Simulated vehicles can be controlled through CARLA API with clients over TCP.

Fog/Edge computing [1] offers an option for the realization of such services, but calls for scalable computing solutions agreeing with the limited computational capacity available (*e.g.*, micro-datacenters). Our research with distributing the stateful simulation model state and its rendering to multiple nodes faces the same problems as real-time computation of stateful smart mobility services on the Edge and in the Fog.

III. SIMULATOR ARCHITECTURE

As illustrated in Figure 5, the CARLA simulator architecture consists of two separate components: the CARLA clients and the CARLA server. A CARLA client can be used to control the vehicles inside the simulation and to receive live sensor data from the vehicles. A client can also be used to control the simulation environment, for example by changing the weather parameters and adding new vehicles and pedestrians to the simulation. Multiple clients can be connected to a single server simultaneously and a single client can control multiple actors. (In this context, the term *actor* is used to refer to any dynamic object that exists in the simulation, such as vehicles, pedestrians, traffic lights and sensors.)

The CARLA server is a plugin for Unreal Engine that coordinates the simulation inside the simulation instance. The CARLA server reacts to the control inputs from the clients, while the Unreal Engine calculates the physics and renders the graphics in real-time. CARLA runs on top of Unreal Engine 4, which uses the CPU-accelerated implementation of the NVIDIA PhysX [12] for the physics computations, and GPU-acceleration for graphics rendering.

Fig. 6. Improved high level architecture of CARLA for distributed rendering. The physics state is computed on the main server, and is then synchronized to multiple CARLA simulation instances running on separate computers. Clients can request sensor data from any server, allowing the rendering load to be distributed over all servers.

While one CARLA server supports multiple client connections coming from multiple computers, the simulation itself can only be run on a single computer. Therefore, the performance of the simulation is limited by the hardware resources of a single computer. Our initial measurements show that the camera rendering pipeline, which includes the actual rendering of the images from cameras and sending of these images to the client, quickly becomes the bottleneck when the number of cameras is increased. This is the case even with relatively low resolution cameras (*i.e.*, 360x240 pixels).

Our goal is to improve the overall performance of CARLA by implementing a system for distributing the rendering load over multiple computers, as illustrated in Figure 6. From the figure, we can see that the idea is to have a scalable number of servers for rendering and one main server for computing the simulation state. The simulation state is synchronized between the main server and the graphics servers. In other words, every actor in the simulation world should ideally have the same position and velocity on all the servers at every point in time.

IV. DISTRIBUTED RENDERING

The goal of distributed rendering is achieved by modifying the existing CARLA source code to support the Unreal Engine networking features. Unreal Engine is designed to support latency-sensitive online multiplayer games, which is ideal for keeping the simulation instances as closely synchronized as possible.

By default, only the main server instance of Unreal Engine has the authority to affect the simulation in any way. All the commands that are executed on the graphics servers will only affect the state of the simulation in that particular server. This means that all the commands must be redirected from the graphics servers to the main server. This is done with the use of RPCs (Remote Procedure Call).

However, as Unreal Engine treats RPCs asynchronously, the communication between the clients and the servers is no longer synchronous. For example, after requesting a new vehicle, we can no longer know when the vehicle is created and if the creation was successful. We had to make changes to the CARLA client-server API to take this in to account.

While the commands are redirected to the main server, we have to ensure everything is correctly synchronized from the main server back to the graphics servers. For all simulation actors created on the main server, we need to create identical actors on the graphics servers as well. This is mostly done automatically by Unreal Engine, except for the CARLA server specific components such as sensors and actor identifiers.

As anything that is created on the graphics server (and not redirected to the main server) will only exist on that particular server, the rendering is easily distributed across servers by creating the sensors only where they are required. As the sensors do not exist on other servers, they cannot affect their computational load. This is ideal for the proof-of-concept demonstrated on this paper. However, for some use cases it could be useful to gain access to the parameters, locations and images of all cameras in all simulation instances.

While the main server holds the absolute ground-truth of the simulation state, the graphics servers are also extrapolating physics between receiving updates from the main server. Extrapolating physics can help to reduce the perceived latencies between the physics states of the separate simulation instances. However, it might also cause minor, temporary deviations from the ground truth of the main server.

V. TIMING ASPECTS

In mixed collaborative driving scenarios timing of events plays a critical role. AI system interactions (also with humans) requires that all entities receive a consistent view of the model world. Computation and communication need to be performed in a predictable low-latency way.

In real vehicular systems, GPS systems can be used to synchronize clocks to a global ~ 100 ns accuracy [9]. But, using GPS devices with hybrid simulation setups and partly virtual simulation entities is impractical.

In our system, the main server clock is the ground truth time. The evaluation of AI system behavior is based on comparing AI assumed states to the ground truth state and the global time of the model. This comparison is possible by introducing timestamps to the model state updates and the related rendered sensor data. When sensor data is being processed on the system pipeline on multiple different steps, the relative time spent in each step can be calculated using local clocks. The total time that it takes from updating the model on the main server to receiving new control commands back can be measured exactly. The total time spent on the communication links can be calculated, but there is uncertainty on the exact time spent on the individual links. This can be estimated by responding with acknowledgement messages whenever a message has been transmitted from a node to another. The individual link latencies can be derived from the round-trip time over the link.

Figure 7 presents computational pipeline that forms around the simulation control loop. The distributed rendering nodes transform the model state data into sensor data from which affects the control commands taken by the clients.

Fig. 7. Measuring the time spent in the simulation-control loop is essential for the evaluation of the AI systems using the corner case search. By having a ground truth clock on the main server, and by timestamping each model update and the related rendered sensor data streams, our system can track the time spent in each main computational phase of the system pipeline.

In Figure 7, the exact timestamping locations and the measurement points for relative time ranges are presented. The times t_0 and t_1 represent the model update time and the time when control commands are received back from the client. The total time:

$$t_{\text{total}} = t_1 - t_0$$

is also the time it takes for a client to react to changes in the model world.

In Figure 7, the computational times in separate nodes are presented with time ranges $\Delta t_{\text{physics}}$, $\Delta t_{\text{rendering}}$ and Δt_{client} . These durations are measured using node local clocks.

The time ranges Δt_{c1} , Δt_{c2} and Δt_{c3} represent the communication durations between computational nodes. Their sum, the total communication time, is upper-bounded by:

$$t_{\text{communication}} = t_{\text{total}} - \Delta t_{\text{physics}} - \Delta t_{\text{rendering}} - \Delta t_{\text{client}}$$

Estimates for the individual node-to-node communication latencies can be approximated by sending acknowledgement messages from the receiving node back to sender upon message arrival. The node-to-node link latency is approximately half of the round-trip time over the link.

VI. EXPERIMENTATION

The experiments were done by connecting multiple simulation instances together and measuring the performance while periodically increasing the computational load. The computational load was increased by adding new vehicles to the simulation. Each vehicle is equipped with one forward-facing camera with image resolution set to 360x240 pixels. The resolution was chosen as it is relatively small, but enough for CNN models suitable for controlling autonomous vehicles [8]. However, higher resolutions, *e.g.*, 640x480 could help with detecting distant objects [3]. All vehicles exist on all simulation instances, but the cameras are distributed evenly over all graphics servers and the main server. Each camera produces one image per simulation step. The computationally

Fig. 8. This graph shows how the total image throughput scales when the number of servers used for rendering is increased. Y-axis is the total number of images received from all vehicles per second. X-axis is the number of cars and cameras in the simulation.

most powerful server has the same workload as the weakest server.

Figure 8 shows the total image throughput achieved with different experiment setups. In more detail, the image throughput shown on y-axis is the sum of images received from all servers on average per second. If server A was to produce 10 images per second and server B 40 images per second, the throughput shown on the graph would be 50 images per second.

By looking at Figure 8, we can see that the throughput clearly increases as the number of servers is increased. The lowest throughput of ~ 80 fps is achieved by running the simulation only on the main server and the highest throughput of ~ 150 fps is achieved by running the simulation on all servers. These results show the benefits of distributed rendering to overall rendering throughput. However, the throughput is not multiplied by four, even if the number of servers is increased from one to four. This is mostly due to the significantly weaker hardware on the other servers, but there might also be some additional overhead caused by the Unreal Engine’s replication system.

We can also see that each setup becomes saturated at a different number of vehicles. With one server, the maximum throughput is achieved around 6–7 vehicles. Similarly, two, three, and four server setups achieve their maximums around 10, 15, and 20 vehicles, respectively.

With low number of vehicles, the throughput is low because there is not enough rendering load to fully utilize the GPUs. This is because each camera renders only once per simulation step. Each simulation step also includes additional computations, *e.g.*, the physics step. With only one vehicle in the scene, the additional computation steps are acting as a bottleneck as the GPU spends more time waiting for new rendering commands.

Figure 9 shows the relative differences in the performance of the servers. Y-axis shows the average time each server uses

Fig. 9. This graph shows the performance difference between the separate servers used in the experiment. The y-axis is the mean time each server spends computing a single simulation step and the x-axis shows the number of cameras assigned to each server. A single simulation step consists of the rendering of one image from all the cameras assigned to the server and advancing the physics and simulation logic by one step.

to render each camera once and to advance the simulation logic and physics by one step. X-axis shows the number of cameras assigned to each server.

In Figure 9, the mean rendering time is the time range $\Delta t_{\text{rendering}}$ presented in Figure 7. For evaluation of multiple client interactions, the mean rendering time should be consistent across the servers in order to avoid timing anomalies. If rendering time is limited to, *e.g.*, 0.10 s, then the servers 0, 1, 2, and 3 can handle a maximum of 7, 4, 1, and 2 cameras respectively. System monitoring and load balancing is needed to enable system timing consistency.

The server 0 that is used as the main server, is equipped with GeForce GTX 1080 TI GPU and Intel i7-5820K CPU. The other servers have significantly slower CPUs and slower GPUs (GeForce GTX 1050 TI and GeForce GTX 680). These hardware differences are highlighted by the fact, that the servers 1, 2, and 3 performed roughly 40%, 150%, and 200% slower (respectively) than the main server.

VII. DISCUSSION

From the results in the previous chapter, it can be seen that our distributed rendering system offers significant performance benefits over a single computer setup. Using the approach, we can scale the simulation to support a realistic number of connected clients. This enables real-time simulation of realistic-scale autonomous traffic scenarios, allowing the evaluation of mixed collaborative driving together with machine learning based AI systems and human drivers.

The rendering related computation scales up with increased number of servers. In our experimentation, the server setup contained of heterogeneous hardware components. The performance differences between the servers, as shown on Figure 9, offer a reasonable explanation for the throughput seen in Figure 8.

There are a number of directions for further research. Different server capabilities and different workloads, such as AI systems requiring acceleration of machine learning inference, open up the need for load balancing. It is possible to develop a system that can dynamically scale the simulation to multiple machines in a computing cluster, *e.g.*, by using datacenter management tools such as Kubernetes [11]. Ideally, each CARLA client would connect to a single end-point, which then redirects the communications to a suitable server based on current server load and the demands of the client.

A challenge is to evaluate how well the physics states of each simulation instance is synchronized. States synchronization in our experimentation was done over UDP, where potentially dropped packets between the servers can cause synchronization issues. It is possible that two images taken exactly at the same time on two different servers are not identical. However, one should note that both human drivers and CNN-based robotic drivers are not perfect in their decision making. Considering the whole system operation, their typical limitations also limit the requirements that are useful for a machine learning environment.

Further research is required to find out the scale and consequences of possible de-synchronization issues, and how the effects can be reduced in distributed simulation. One potential workaround could be to run the simulation in a discretely synchronized manner, always waiting for each server to be completely synchronized before computing the next simulation step. But this would yield significant slowdown compromising the real-time operation.

Real-time simulation is required for testing how real hardware and software perform in real, non-simulated situations. Testing real systems in non-real-time scenarios may hide issues related to computation time requirements and the latencies caused by, for example, communicating over a network.

Real-time simulation is also a necessity for developing and testing human reactions to AI communication mechanisms. Photorealistic simulation allows the development and testing of different mixed collaborative driving solutions. Human reaction time can be measured in the same fashion as AI system latencies. The corner case search subsystem can also be used to spot abnormal behavior in human drivers control commands. The machine learning environment allows the testing of different AI to human communication mechanisms, from, *e.g.*, head-up-display messages to AI vehicle installed physical messaging actuators.

VIII. CONCLUSION

Photorealistic distributed simulation is an approach for autonomous driving research. The heavy computational requirements of producing realistic sensor data can be tackled by distributed computation. This leads to synchronization and timing challenges, which can be overcome using detailed timestamping and time intervals.

In this paper, we presented our experimental results on the scalability of a real-time machine learning environment based on simulation. Our results show that distributed computing

systems are able cope with the simulation load of realistic driving scenarios.

In our research, we improved the volume of photorealistic sensor data that the CARLA simulator can yield for a machine learning environment. Our results show that heterogeneous computer platforms can be used to serve multiple clients at the same time. The results further indicate, that system monitoring is needed to balance the individual server loads, in order to minimize the timing and synchronization challenges.

Real-time simulation is especially important for evaluation of scenarios with both humans and AI systems involved. Evaluation of real hardware and software needs to be done in real-time to determine their performance capabilities. Human driver reactions can only be measured realistically when the sensory input is realistic enough, both visually and timing wise.

Autonomous driving research is sought widely by industry and academia. New tools and research methodologies are needed as the research challenges are complex. As pointed out by our paper, the specific properties of driving situations offer ways overcome some of the challenges.

REFERENCES

- [1] Yuan Ai, Mugen Peng, and Kecheng Zhang. 2018. Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks* 4, 2 (2018). <https://doi.org/10.1016/j.dcan.2017.07.001>.
- [2] Chen, Chenyi, et al. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*. 2722–2730.
- [3] Cho, Hyunggi, et al. 2014. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1836–1843.
- [4] Codevilla, Felipe, et al. 2018. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1–9.
- [5] Dosovitskiy, Alexey, et al. 2017. "CARLA: An open urban driving simulator". In *Proceedings of the 1st Annual Conference on Robot Learning*.
- [6] Unreal Engine. 2019. (2019). <https://www.unrealengine.com>
- [7] Martín Abadi et al. 2015. "TensorFlow: Large-scale machine learning on heterogeneous systems". (2015). <https://www.tensorflow.org>
- [8] Mariusz Bojarski et al. 2016. End to End Learning for Self-Driving Cars. (2016). <https://arxiv.org/abs/1604.07316>
- [9] Alessandra Flammini and P Ferrari. 2010. *Clock Synchronization of Distributed, Real-Time, Industrial Data Acquisition Systems*. DOI : <http://dx.doi.org/10.5772/10458>
- [10] Huval, Brody, et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716* (2015).
- [11] Kubernetes. 2019. (2019). <https://www.kubernetes.io>
- [12] NVIDIA PhysX. 2019. (2019). <https://developer.nvidia.com/gameworks-physx-overview>
- [13] Sabella, Dario, et al. 2017. Toward fully connected vehicles: Edge computing for advanced automotive communications. *5G Automot. Assoc.(5GAA), White Paper* (2017).
- [14] Shital Shah, Debadepta Dey, Chris Lovett, and Ashish Kapoor. 2018. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*. Springer, 621–635.
- [15] Siam, Mennatullah, et al. . 2017. Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–8.
- [16] Wymann, Bernhard, et al. 2000. Torcs, the open racing car simulator. *Software available at http://torcs.sourceforge.net* 4, 6 (2000).