Submission Template for ACM Papers

Performance evaluation of distributed hardware-in-the-loop simulators for autonomous driving vehicle validation

An industrial case study applying the performance evaluation techniques measurements, discrete event simulation, queuing theory and network calculus and comparing them to each other

First Author's Name, Initials, and Last name[[1]](#footnote-1)\*

First author's affiliation, an Institution with a very long name, xxxx@gmail.com

Second Author's Name, Initials, and Last Name

Second author's affiliation, possibly the same institution, xxxx@gmail.com

Third Author's Name, Initials, and Last Name

Third author's affiliation, possibly the same institution, xxxx@gmail.com

Hardware-in-the-loop test benches are distributed computer systems including software, hardware and networking devices. They are used for Validation purposes of complex technical systems like autonomous driving systems here in this study For this purpose they require strict real-time guarantees and high data integrity. To guarantee strict real-time of the simulator the performance of the software processes in the streaming chain of the simulator needs to be evaluated. To guarantee data integrity the queues between the software processes need to be designed properly, to prevent any data loss by buffer overflows. To evaluate the timing performance and data integrity measurement on a prototype HIL are done and a discrete event simulation model is built up. The simulation model is validated via trace driven simulation with measurements from a prototype of the real system. The input modeling is done with distribution fitting based on measurement data from the real system in a prototype phase. The workload is increased by decreasing the cycle-time to see if the simulation model performs well near to the bottleneck of the streaming software chain. The results of key performance indicators, namely latency and queue length are compared between measurements, the distributional simulation model as well as analytical solutions derived by queueing theory and linear network calculus properties applied on the measurement data. The results derived by the different methods are discussed critically and the methods are compared to each other.

CCS CONCEPTS • Insert your first CCS term here • Insert your second CCS term here • Insert your third CCS term here

**Additional Keywords and Phrases:** distributed system, networks, hardware-in-the-loop simulation, HIL, real-time performance evaluation methods and techniques, discrete event simulation, DES, OMNeT++, network calculus, queuing theory

ACM Reference Format:

First Author’s Name, Initials, and Last Name, Second Author’s Name, Initials, and Last Name, and Third Author’s Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock ’18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1. Introduction

Autonomous driving functions are safety-critical and have to be tested thoroughly. The hardware-in-the-loop (HIL)-based testing approach is an enabler for this and has been established in the automotive industry for years as an effective and efficient method of validating control units, interfaces, and functions. Since the system under test works under hard real-time (RT) conditions, the HIL test bench needs to be RT capable to enable the testing and be able to check the RT properties of the system under test. The focus here is on the use-case of the so-called open-loop reinjection [1], as streaming of measurement data to the device under test (DUT) under strict RT constraints. The HIL test system needs to be assessed regarding the RT capability of the whole streaming chain. Streaming tones down the RT requirements within the chain to soft RT and with the help of over-provisioning and pre-buffering, the design process can be easened. But there are a lot of dimensioning issues for buffers sizes, throughput and processing delays of Software elements running on Hardware left, for that purpose we want to analyze the system more deeply with the help of Discrete Event Simulation.

1. Background
   1. Hardware-in-the-loop Simulation

Hardware-in-the-loop Simulation is a widely used approach within the industrial development process of mechatronic systems including electronic devices. The HIL is part of the validation side of the development process and used for validation and verification of functionality and robustness of mechatronic systems, including software and hardware.

There is no official definition for HIL by the IEEE as depicted in [1], nor in the AUTOMOTIVE SPICE but the paper itself proposes the following definition for HIL systems: “Hardware-in-the-loop system is a non-intrusive test approach, containing physical controller connected in open- or closed-loop with virtual or semi-virtual subsystems, providing faithful physical replicas of the real world and evaluating the SUT in either black / grey / white box manner.” That definition gives a full picture of a HIL system and includes all important technical terms. A HIL system normally looks like that:



Figure : Conceptual model of a HIL system

It consist of a HOST PC, a HIL device and a device or system under Test.

The HOST PC is the master of the HIL system. It controls and configures the HIL system and is the interface to the user, which is commonly a Test engineer or a HIL system developer.

The HIL device is a real-time computer which is connected via Real Time communication interfaces to the DUT.

The Device under test (DUT) or in case of many Devices connected together to a system it is called System Under Test (SUT) are described in the upper definition as physical controllers. These are electornical devices or mechatronical systems that are plugged to the HIL testbench and they are a part of the HIL system itself.

Furthermore there are two basic possibilities of operational use and setup of the HIL, also described in the upper definition. One possibility is the Open-Loop approach which means that measured or simulated data is streamed in RT to the DUT, and the output of the DUT is measured and compared to a ground-truth. So there is no feedback loop back to the HIL and the creation of the input data to the DUT is independent of it’s output data, which means its feedback loop is open. The other possibility is the Closed-Loop approach where commonly simulated data is generated by a RT model running on the HIL device, and the output is feed-back to the HIL device and used as an input for the RT model, so a feed-back loop is created, which means the loop is closed.

In our Use-Case the SUT are High Performance Computers for Autonomous Driving Functions, testing the Robustness of the Perception Software in an open-loop approach at the HIL.

* 1. Computer Performance Evaluation

Why is Computer Performance Evaluation so important? The goal is to get the highest performance for the lowest cost. [4]

Why is Computer Performance Evaluation often seen as an art than a science? [4] [5]

As depicted in [4] there are numerous types of applications, so no standard performance measure can be identified and every performance analyst has its own style. So two performance engineers can analyse a computer system with completely different techniques or even outcome when choosing the right system or interpreting the results.

But although it is understood as an art, there are common steps to do a computer performance evaluation according to [4]:

* choose the right metrics,
* chose the right techniques
* define the right workload.

Another aspect is that an evaluation needs a deep knowledge of the system under study [4], so the steps for performing the computer evaluation are highly individual to the Use-Case of the Computer system.

To evaluate the performance of a computer system, according to [4] at least two of the following three methods are needed to validate each other: measurements, simulation and analytical modeling.

If the system is physically available, measurements should be performed to understand the actual system behavior, and for practical purposes to recognize and correct malfunctions.

In addition, a simulation or an analytical model should be generated to validate the measured results, as measurement can never give you a true value due to measurement inaccuracies or look into the future. But with an model you can extrapolate the results to other workloads, other setups or system parameters or scaling up your system. So you are able to have a look on the performance on a physically unavailable system. The generated model should be compared with measurements to see if it is valid or if it is lacking of details to represent the real world accurately enough.

* 1. Discrete Event Simulation

Discrete Event Simulation is an often used technique for performance evaluation of real world systems. As described in [6] it is composed of an event list, a system stat and the elapsed simulation time. The progress of the simulation is executed by processing events from the event list. Event processing can modify the system state, increase the simulation time or produce new events. In contrast to continuous simulation it changes it system state in discrete sequences of time events.

The shortcome of DES is that it remains a statistical experiment, in contrast to numerical or analytical techniques, as described by Haverkort at QUEST 2021 in [5].

But according to Haverkort the benefit is that it is often “easier than trying to deal with modeling limitations of “more advanced” numerical or analytical techniques”, when carefully dealing with the shortcome.

Another advantage is, as experienced within that work, that DES helps to understand the system better. Within the modelling process a lot of adaption and refinements of the model have been done in particular after each validation process, when comparing the simulation results to the measurements, especially with the Trace Driven Simulation Approach, which is “an important tool in many simulation applications in which the model’s inputs are derived from a sequence of observations made on a real system.” [5]

* 1. Queuing Theory
  2. Network Calculus

1. Related Work

To the best of our knowledge there exist no papers on performance evaluation of HIL systems.

But there are numerous papers about performance evaluation in related fields, like distributed streaming systems, Internet of Things and Network/System Co-Simulation.

In the field of distributed streaming systems there are the following two works [12] [13] which have similarities to our system.

Fu et al. focus on a data stream management system with real-time constraints. [12] They use a M/M/ki Queuing model to describe the scheduling performance on CPU cores, they were neglegting networking network latency. They found a good estimation of delays with that performance model, even not all assumptions were fullfilled. In contrast to [12] we don’t neglect networking delay, as it is a crucial part of our system-

Lohrmann et al. used a G/G/1 queuing model to optimize elastic stream processes and give latency guarantees [13]. They used the model for optimization of the networking devices as 120 workers were connected via 1Gbps Ehternet connection in a star topology. This system under study is much more complex than our HIL system and focusses in optimizing parallel processes. In contrast the processes of one stream in our system are connected in series. But many Streams can run in parallel, but the processes between the streams are not directly connected to each other. Any Correlation between them will come from the HW resources which are shared among them.

In the field of Performance Evaluation of IOT Applications are the following two papers from the recent years worthwhile to be named [14][15].

Both works use an OMNET++ Simulator and SW instrumentation for latency measurements of applications including the INET network simulation. They enrich the OMNET++ model with a scheduler and CPU model with different technologies. They use the simulation model to make early performance predictions. Inspired by their approaches we developed our own CPU and scheduler model based on FIFO and Round-Robin Scheduling Strategies and integrated it in our model. The code of the model can be found in github. [16]

La Boudec

1. Contribution

To the best of our knowledge there is no further work about Performance Evaluation of HIL systems in the Autonomous Driving domain with performance techniques like Discrete Event Simulation, network

As the HIL system is build up on our own, with own testsystem software a special use-case and workload, there is no official benchmark by HIL component or SW deliverer, which is matching to our use case. So the computer performance evaluation has been done on our own and is unique.

1. System description and detailed conceptual model

A HIL is a distributed computer system, for which common methods from computer system performance evaluation can be applied to evaluate the RT performance. At first, a very simplified conceptual model has been developed to understand the system, set up requirements and be able to develop a model from it.

The HIL system under analysis consists of two distributed computer systems as depicted in Figure 1 connected via network interface cards (NIC) and an Ethernet connection using the TCP-IP protocol. Each computer has an application layer, a Linux operating system (OS) and a hardware layer (HW). Additionally, the left PC has the Robot Operating System (ROS) [2] as a middle layer between the application layer and the Linux OS with RT preempt patch [3]. The streaming application is running as a multi-threaded multi-processing ROS application and it is instrumented at the user-code level in C++ to gain timing information when a process starts and when it is finished. The measurements from the system will be used for input modeling in the simulation model.

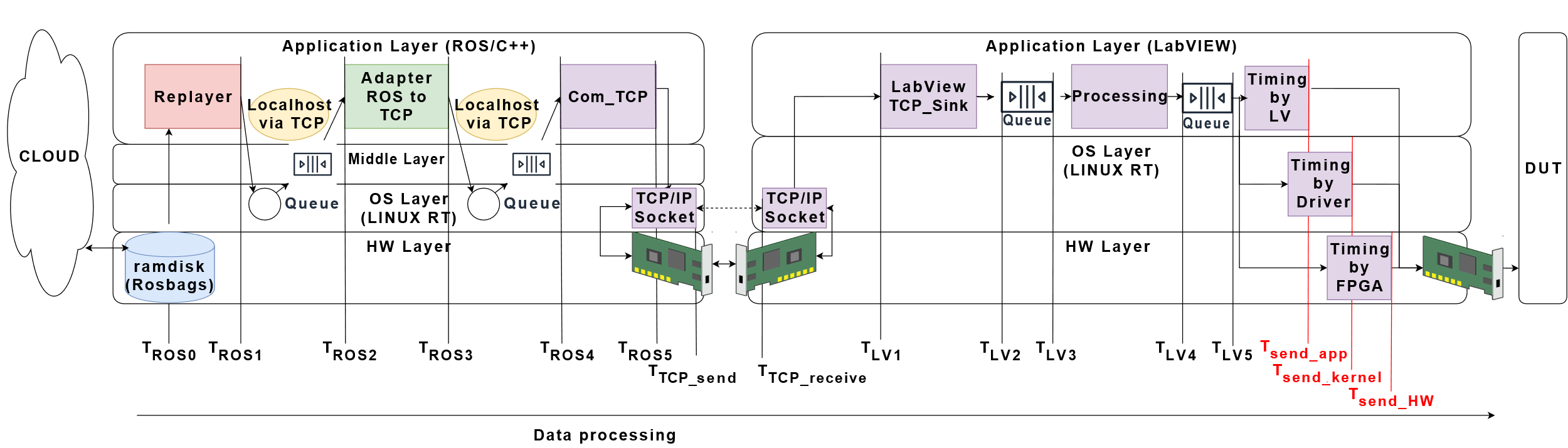


Figure :Detailed Conceptual Model and Software (SW) Instrumentation.

The streaming process starts at the far-left SW process, the ROS replayer, that grabs the data from the RAM disk and sends it, according to its timestamps, to the next node. The interprocess-communication is done via TCP/IP and the data is sent to a queue in the ROS middle layer. This is done by every process until the data reaches the last node. Then the data is sent over the TCP socket to the network stack of Linux depicted in the kernel layer to the network interface card (NIC). On the right PC, the process is similar, but starting from the NIC to receive the TCP segments via interrupts over the kernel to the first application process in LabVIEW. Within this first process, the segments are collected and reassembled or segmented to the original frame size, depending on the original frame size in relation to the maximum segment size (MSS). When the original frame is restored, it will be sent to the next process until the last queue, before sending it out to the device under test (DUT). This last queue has a very important function, which is the compensation of the delay and jitter introduced by the whole processing and queuing chain.

1. Approach/Methodology
   1. Measurement System and Instrumentation

As Measurement System an pure SW logger was used. The Logs were intrumented in user space. When data arrives a system timestamp, checksum and byte-lengh information is generated at each processing step. The timestamp is generated directly and put on a separate worker queue. The meta-information is processed in a separate thread in parallel, to not influence the normal system behaviour too much.

As latency input the minimum value of the latency between arrival to a queue input of a process and departure from a process and time difference (ΔT) of two consecutive departure processes are taken to model the maximal possible Service. This is done to exclude any waiting time in queues into the process latency. So we assume that the process is beginning directly after the process before finished, as we are not able to log at all queue outputs directly, this holds especially for the inter-process communication level.

(1)

* 1. Performance Metrics definition

To define the most important performance metrics the requirements to the system are important. The first important Requirement is to meet RT guarantees, this leads to the following technical requirements to the system. The buffer directly before sending to DUT should never run empty during the streaming phase. To full fill that requirement the end-to-end delay from starting entity to the arrival into the last buffer must be bounded. So the end-to-end delay is the first important Performance metric.

The second important requirement is to meet data integrity, so the streaming data should not be changed or get lost within the system. The corruption of data can be checked by a checksum logged with the timestamps of the data. But a data change is assumed as neglectable, as the transportation is done via TCP/IP or directly vie memory.

One possibility of data corruption what is more likely is data loss due to buffer overflows. The buffers need to be designed properly and as long as the RAM is not a bottleneck they are oversized. The second important Performance metric to look on is the queue length.To wrap it up, for un-interrupted streaming there are the 3 following Performance indicators in the system:

* Max. Delay < end-to-end delay bound
* Min. Queue length >0 at the last buffer during streaming
* Max. Queue length < max buffer size

Further performance metrics for the system design focused on the Hardware are CPU Network and RAM utilization.

But this is out of scope in this study as the CPU power, Network bandwidth and RAM size are oversized for this study and assumed as not being the bottleneck.

* 1. Workload Definition

Sensor data is the workload of the HIL system and the most important input parameters of the workload are the measured Sensor cycle-times and the byte-sizes of the messages send by the sensors. So the workload is defined by the Sensor-set which is measured and used for the dedicated HIL in the special project.

The easiest possible workload is a fixed and deterministic cycle-time and fixed byte-size for each frame, as in the case of Camera data. This kind of workload is used in this study. The cycle-time for input modeling is defined big enough to be far away from a processing latency bottleneck to prevent any queuing in the system. The resulting data is used for distribution fitting as input modelling for the processing latencies of the application layer. The workload is than changed into the direction of the bottleneck of processing latencies, so the cycle-time is decreased, to see if the model is sensitive to that change.

* 1. Discrete Event Simulation Model

In the next step, a discrete-event simulation (DES) model [9] was built up to gain a better understanding of the system. Discrete event simulation is particularly suitable for evaluating the stochastic behavior of processes in computer and communication systems [10] and gaining statistical insides. [5]

The model is build up according to the conceptual model of Figure 1 in OMNeT++ based on standard servers and queue elements and with the INET framework for the networking unit using the TCP/IP protocol implementation. Some servers are customized with a special latency for the first element to model a buffer time as well as a possibility to include timestamps to the data and a possibility to use these timestamps to replicate the sending behavior to the device under test. Another feature is the recreation of TCP packetized data with merging or splitting it to the original data frames. The servers are also extended to be able to use measurement data as xml input, to be able to do tracedriven simulation.

* + 1. Tracedriven Simulation for Valdiation of Discrete Event Simulation Model

For validation purposes, the model is fed with measurements, called trace-driven simulation [4]. Then the simulation output is compared to the measurements. Latency measurements are taken as service times of the servers in the model, and we compare these service times of the simulation output of the model and the measurements as depicted in Figure 2.

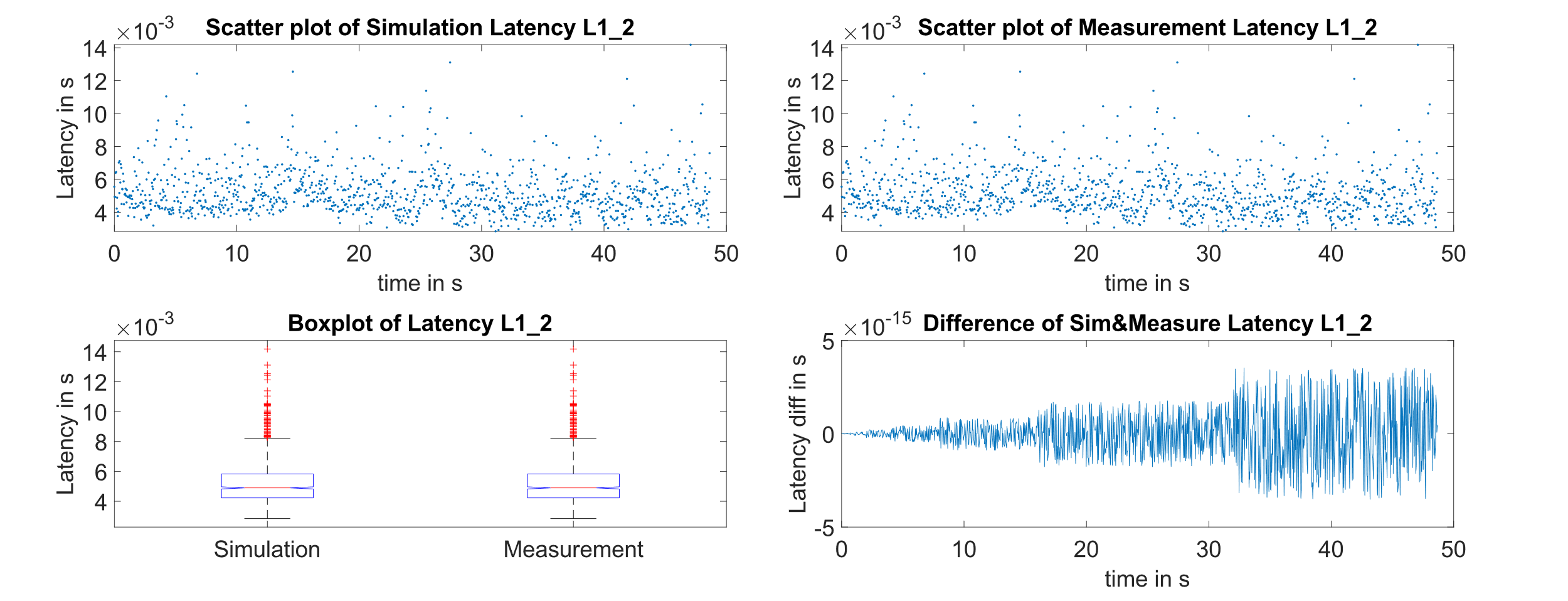


Figure :Example of process service times; from left to right:  
upper figures: scatter plot of simulation and measurement data,   
lower figures: Boxplot of Simulation and Measurements, Difference between Simulation and Measurements at each point over time

As can be seen from Figure 2, the simulated and the measured data are highly comparable. From the boxplots, we can see that they are distributed equally. The last right plot shows the calculated differences between the simulation and measurement latency. This is in the range of 10-15 s for processing latency and 10-7 s in case of end-to-end latency. The conclusion is that the conceptual model of one data stream is representing the actual system in a very exact way.

* + 1. Fitting Distributions of processing latencies for input modelling

As the simulation results of the model are satisfying, a mapping of the stochastic process behavior with theoretical distribution functions can be started. This will bring independence of a specific trace and simulate the behavior over a longer period and with random state combinations.

For distribution fitting from the measurement data, MATLAB is used to fit the data into different statistical distributions, which are supported by OMNeT++. In a further step, the distribution is truncated to the minimum and maximum measured values. Then a goodness of fit analysis as described in [7] is performed. The distribution with the highest mean p-value is taken and feed into the model. As can be seen exemplarily in Figure 3, the process latencies are lognormal distributed with a mean p-value of 37%, which met the requirement to be at least at 5%.

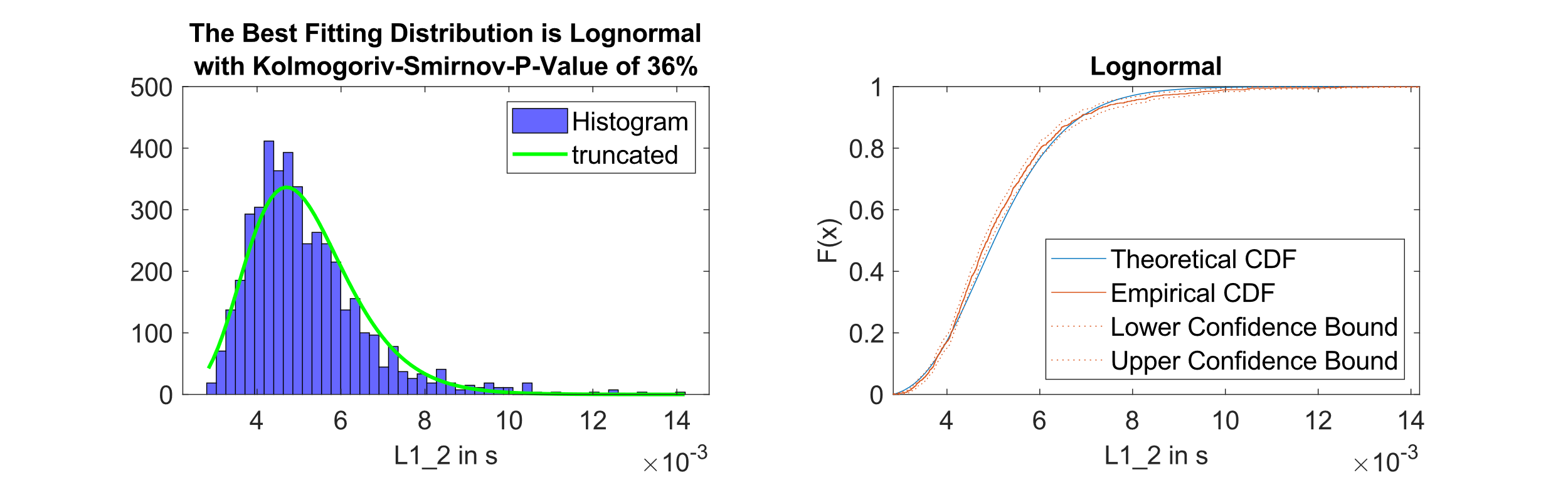


Figure : from left to right: Histogram and Distribution Fit of ROS Process Latencies, Comparison between Theoretical & Empirical Cumulative Distribution Function (CDF).

* + 1. Refinement of the model with special behavior

After comparing the simulation results to the measurements the model is refined when it shows a significant difference, especially when the simulation model underestimates the upper bounds of the performance parameters compared to the measurements.

When analysing the interarrival times of the ROS Replayer it showed a behaviour like a Token-Bucket server in the beginning of the stream and after a pause when restarting. With number of bursts are equal to 100ms/cycle-time coming deterministic when starting the replay mode. Modeling this behaviour is easy and straight forward with including a first waiting time of 100ms, afterwards the processing time of the first server representing the ROS replayer can be 0 or a stochastic behaviour with a normal distribution influencing the interarrival time as needed, depending on the influence on the overall system behavior.

* 1. Queueing Theory
  2. Network Calculus

1. Case Studies

The case study is based on a single CAN data stream. The stream duration is about 90s and the measurements are done 10 times for statistical confidence. We do performance evaluation on that kind of data with all the prior described methods. Starting with deterministic cycle-time of 10ms.

* 1. CAN stream with 10ms cycle-time

One important Key Performance Indicator (KPI) is the max. Queue length in all buffers in the system. This KPI is used to design the buffer sizes. The queue length should never overshoot the buffer size, as this would mean loss in case of drop-tail queues or blocking of processes in case of blocking queues. The following table gives a comparison of the max. Queue length found in measurement compared to tracedriven and simple distributional simulation

Table : Comparison of maximum Queue Length of Measurement, Tracedriven Simulation and regular Distributional Simulation with input-workload with deterministic cycle-time and input-workload with stochastic cycle-time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Max. no of elements in the Queues | | | | | | |
| Queue | Measurements  with const input (10ms) | Tracedriven simulation  with const input (10ms) | Distributional simulation with const input (10ms) | Distributional simulation– with const input (10ms) but 11 bursts in the beginning | Distributional simulation– with const input (10ms) but 11 bursts in the beginning and higher processing latency in the first processing step | Queuing Theory | Network calculus |
| ROS 12 | 11 | 11 | 2 | 8 | 11 |  |  |
| ROS 34 | 11 | 11 | 1 | 2 | 11 |  |  |
| LV 12 | 10 | 11 | 2 | 2 | 11 |  |  |
| LV 34 | 1715 | 1715 | 1715 | 1715 | 1714 |  |  |

Conclusion from the upper table is, that the maximum measured and with tracedriven model confirmed max. queue length for all buffer in the streaming chain is 11 elements. The queue length of the distributional model is much smaller, as the distribution covers the regular streaming mode but does not include any bursty behaviour from the start-up phase of the ROS Replayer. When simulating the ROS Replayer behavior of a Token-Bucket Server with 11 Buckets in the beginning the behavior comes near to the measurement and trace-driven simulation but it underestimates the queue-length nevertheless. A further refinement of the model was done by having a deeper look into the measurement data. The measurements have shown that especially the interprocess communication has a significant higher delay in the first step of the processing. So with including this fist higher latency for processing of the first element into the servers, the max. queue size can be resimulated.

Table Latencies Statisics of Services, \*without latency on the ethernet link between the systems, as systems could not be timesynced due to technical restrictions

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Latency between Services in [µs] | Measurements  with const input (10ms) | Tracedriven simulation  with const input (10ms) | Distributional Simulation model with const input (10ms) | Distributional simulation– with const input (10ms) but 11 bursts in the beginning | Distributional simulation– with const input (10ms) but 11 bursts in the beginning and higher processing latency in the first processing step | Queuing Theory | Network calculus |
| Min Latency T1-T5 | 243 | 243 | 244 | 244 | 8 |  |  |
| Min Latency T6-T8 | 168 | 74 | 66 | 51 | 64 |  |  |
| Min Latency T1-T8 | 448\* | 449 | 400 | 353 | 79 |  |  |
| Median Latency T1-T5 | 440 | 440 | 439 | 442 | 57 |  |  |
| Median Latency T6-T8 | 203 | 203 | 234 | 230 | 110 |  |  |
| Median Latency T1-T8 | 654\* | 655 | 676 | 676 | 192 |  |  |
| Max Latency T1-T5 | 895 | 895 | 625 | 1868 | 8002 |  |  |
| Max Latency T6-T8 | 9720 | 9720 | 1207 | 1551 | 55898 |  |  |
| Max Latency T1-T8 | 10199\* | 10200 | 1557 | 2246 | 55947 |  |  |

The statistics of the latencies in both computer systems between the measurements and the tracedriven model are nearly identical, so the model concept and the measurement points are valid. There is one outlier, which has nearly a factor-two difference, the minimum Latency between T6-T8. But as it is in the range of 100µs and the Inter-Arrival Times of the streams are in >1ms it is neglectable.

The median and minimum latencies of each system (T1-T5 & T6-T8) and of the overall system (T1-T8) between measurements and the simulation model with distributional input are also highly comparable, so the simulation model with the distributional inputs represents the “normal” behaviour of the system very good.

The biggest difference can be found in the maximum latency between the distributional models and the measurements, so the distributional model does not reflect rare worst-case events of the system, what was also not expected. An idea would be to measure how often such rare extreme events similar to system freezing occure and how long their duration is, to bring them into the model and to make a propper propagation of maximum latencies.

1. Discussion

All methods are based on measurements and hence get no full guarantee about a bounded end-to-end delay and queue length, as the computer system is too complex to be described in a pure analytical way. But with measurements at least empirical models can be created which give at least a statistical prove. DES gains deep insides into the system and its potential bottlenecks, but remodeling all the special behavior is work intensive. The Network Calculus Approach is a really effective and efficient way to calculate max. delay and queue size bounds, especially when working with linear properties. The short come is that it can lead to overestimation.

ACKNOWLEDGMENTS

This research was supported by ZF AG and ZF Mobility Solutions GmbH (a company of ZF group).

1. HISTORY DATES

In case of submissions being prepared for Journals or PACMs, please add history dates after References as (*please note revised date is optional*):

Received November 2019; revised August 2020; accepted December 2020

REFERENCES

1. N. Brayanov and A. Stoynova, “Review of hardware-in-the-loop -a hundred years progress in the pseudoreal testing”, 2019
2. M. Quigley et al., „ROS: an open-source Robot Operating System“, 2009
3. F. Reghenzani, G. Massari and W. Fornaciari, „The real-time linux kernel: A survey on Preempt\_RT“,2019
4. R. Jain, “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling”, 1. Edition. New York: Wiley, 1991.
5. A. Abate and A. Marin, Eds., Quantitative Evaluation of Systems: 18th International Conference, QEST 2021, Paris, France, August 23–27, 2021, Proceedings, vol. 12846. Cham: Springer International Publishing, 2021. doi: [10.1007/978-3-030-85172-9](https://doi.org/10.1007/978-3-030-85172-9).
6. S. Ross, The Discrete Event Simulation Approach. Elsevier, 2013, p.111–134.
7. D. Nurmi, J. Brevik, and R. Wolski: “Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments”, 2005
8. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System,”, 2009
9. J. Banks, J. S. Carson, B. L. Nelson, D.M. Nicol. “Discrete-Event System Simulation”, 2000
10. A. M. Law, W. D. Kelton. “Simulation Modeling & Analysis”, 2000
11. A. Varga; R. Hornig. “An overview of the OMNeT++ simulation environment”, 2008
12. T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, Y. Yang, and Z. Zhang, “DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams,” in *2015 IEEE 35th International Conference on Distributed Computing Systems*, Columbus, OH, USA, Jun. 2015, pp. 411–420. doi: [10.1109/ICDCS.2015.49](https://doi.org/10.1109/ICDCS.2015.49).
13. B. Lohrmann, P. Janacik, and O. Kao, “Elastic Stream Processing with Latency Guarantees,” in *2015 IEEE 35th International Conference on Distributed Computing Systems*, Columbus, OH, USA, Jun. 2015, pp. 399–410. doi: [10.1109/ICDCS.2015.48](https://doi.org/10.1109/ICDCS.2015.48).
14. G. Amarasinghe, M. D. de Assunção, A. Harwood, and S. Karunasekera, “ECSNeT++ : A simulator for distributed stream processing on edge and cloud environments,” *Future Generation Computer Systems*, vol. 111, pp. 401–418, Oct. 2020, doi: [10.1016/j.future.2019.11.014](https://doi.org/10.1016/j.future.2019.11.014).
15. Z. Zhao, V. Tsoutsouras, D. Soudris, and A. Gerstlauer, “Network/system co-simulation for design space exploration of IoT applications,” in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, Pythagorion, Jul. 2017, pp. 46–53. doi: [10.1109/SAMOS.2017.8344610](https://doi.org/10.1109/SAMOS.2017.8344610).
16. L. Böhm, “Evaluation of an OS scheduler, task dispatcher & CPU model and Integration into a existing OMNeT++ model for Performance Evaluation of Hardware-in-the-loop rigs for Autonomous Driving System Validation,” https://github.com/lukasbm/HiL-Simulator 2020

A  APPENDICES

A.1  Description of HIL cluster used in experiments

The HIL cluster consists of a HOST PC and a HIL RT PC.

The HOST PC is equipped with a 2012 Intel Xeon

E3-1230 V2 3*.*3 GHz processor (four physical CPU cores)

and 16 GB RAM. All nodes worker nodes are connected via

40 GBit Ethernet in a single-switch star topology. Each node

ran Gentoo Linux (kernel version 3.6.11) and Java 1.7.0.13.

1. \* Place the footnote text for the author (if applicable) here. [↑](#footnote-ref-1)