A Novel, Highly Integrated Simulator for Parallel and Distributed Systems

NIKOLAOS TAMPOURATZIS, IOANNIS PAPAEFSTATHIOU, ANTONIOS NIKITAKIS, and ANDREAS BROKALAKIS, Synelixis Solutions Ltd, Greece
STAMATIS ANDRIANAKIS and APOSTOLOS DOLLAS, Telecommunications Systems Institute, Greece
MARCO MARCON, Dipartimento di Elettronica, Italy
EMANUELE PLEBANI, Advanced System Technology, Italy

In an era of complex networked parallel heterogeneous systems, simulating independently only parts, components, or attributes of a system-under-design is a cumbersome, inaccurate, and inefficient approach. Moreover, by considering each part of a system in an isolated manner, and due to the numerous and highly complicated interactions between the different components, the system optimization capabilities are severely limited. The presented fully-distributed simulation framework (called as COSSIM) is the first known open-source, high-performance simulator that can handle holistically system-of-systems including processors, peripherals and

Extension of Conference Paper: N. Tampouratzis et al. An Open-Source Extendable, Highly-Accurate, and Security Aware CPS Simulator. 2017 13th International Conference on DCOSS. When we presented COSSIM at DCOSS, it was a CPS-tailored simulator with limited capabilities and the corresponding evaluation results were, thus, also limited. COSSIM can now support the simulation of Cloud and HPC infrastructures as well, so the work presented in this article extends the work in DCOSS in several aspects: (a) we extended the synchronization system in a novel way so as to allow it to be decentralized/distributed itself while we evaluated it utilizing multiple HLA servers on different physical machines to demonstrate that COSSIM is a fully-distributed framework, with no centralized components. In the conference paper, a simple centralized approach was described, and (b) we extend the COSSIM capabilities in order to support a wide range of different real network protocols by developing the micro-router functionality. In the conference paper, COSSIM could support only wired Ethernet interconnections, so (c) we extended its capabilities so as to allow for different tradeoff between accuracy and simulation times; no such capabilities have been supported in the 1st version of COSSIM. (d) We have extended our complete framework so as to be able to handle a thousand simulated nodes; the original COSSIM simulator could support up to 32 processing nodes. (e) Several real-world use cases of parallel applications are ported to our new system and evaluated across different metrics. For example, we evaluated the new version of COSSIM when simulating up to 1024 X86-based and ARM-based nodes in a cluster that contains 200 physical cores. In addition, in the current article, COSSIM has been evaluated in terms of all the additional features implemented (i.e., accuracy, scalability, and performance) when executing the widely-used Netperf benchmark suite, which allows it to be compared against similar solutions that may appear in the future. Furthermore, a new real-world use case has been added in the evaluation section (Section 6.4), which clearly demonstrates that COSSIM accurately simulates a complete real system. (f) Finally, in order to further increase the impact of our work to the HPC/Cloud/CPS community, the complete source code with its documentation has now become freely distributed through GitHub.

This work is supported by the European Commission in the context of the H2020 COSSIM project (#644042).

Authors' addresses: N. Tampouratzis, I. Papaefstathiou, A. Nikitakis, and A. Brokalakis, Synelixis Solutions Ltd, Farmakidou 10, Chalkida, 34100, Greece; emails: {tampouratzis, nikitakis, brokalakis, ygp}@synelixis.com; S. Andrianakis and A. Dollas, Telecommunications Systems Institute, Technical University of Crete, Chania, 73100, Greece; emails: sandrian@gmail.com, dollas@ece.tuc.gr; M. Marcon, Dipartimento di Elettronica, Politecnico di Milano, Italy; email: marco.marcon@polimi.it; E. Plebani, Advanced System Technology, STMicroelectronics, Italy; email: emanuele.plebani1@st.com.

1544-3566/2020/03-ART2
https://doi.org/10.1145/3378934

networks; such an approach is very appealing to both Cyber Physical Systems (CPS) and Highly Parallel Heterogeneous Systems designers and application developers. Our highly integrated approach is further augmented with accurate power estimation and security sub-tools that can tap on all system components and perform security and robustness analysis of the overall system under design—something that was unfeasible up to now. Additionally, a sophisticated Eclipse-based Graphical User Interface (GUI) has been developed to provide easy simulation setup, execution, and visualization of results. COSSIM has been evaluated when executing the widely used Netperf benchmark suite as well as a number of real-world applications. Final results demonstrate that the presented approach has up to 99% accuracy (when compared with the performance of the real system), while the overall simulation time can be accelerated almost linearly with the number of CPUs utilized by the simulator.

CCS Concepts: • **Computing methodologies** → **Simulation tools**; **Simulation environments**; **Distributed simulation**; • **Networks**→ *Network simulations*; • **Computer systems organization**→ *Embedded and cyber-physical systems*;

Additional Key Words and Phrases: Distributed systems simulator, parallel systems simulator, CPS simulator, integrated simulator

ACM Reference format:
Nikolaos Tampouratzis, Ioannis Papaefstathiou, Antonios Nikitakis, Andreas Brokalakis, Stamatis Andrianakis, Apostolos Dollas, Marco Marcon, and Emanuele Plebani. 2020. A Novel, Highly Integrated Simulator for Parallel and Distributed Systems. *ACM Trans. Archit. Code Optim.* 17, 1, Article 2 (March 2020), 28 pages.
https://doi.org/10.1145/3378934

1 INTRODUCTION

Nowadays, Cyber Physical Systems (CPS) and highly parallel and distributed computing systems (i.e., Cloud and HPC systems) are growing in capability at an extraordinary rate, incorporating processing systems that vary from simple microcontrollers to high performance units connected with each other through numerous networks. One of the main problems the designers of such systems face is the lack of simulation tools that can offer realistic insights beyond simple functional testing, such as the actual performance of the nodes, accurate overall system timing, power/energy estimations, and network deployment issues. On top of that, none of the existing solutions provide security testing so as to be able to assess potential security problems at design time.

In this article, we present the COSSIM Simulation Framework, which is an open-source framework that aims to address all the aforementioned limitations. The proposed solution efficiently integrates a series of sub-tools that model the computing devices of the processing nodes as well as the network(s) utilized in the the interconnections. It provides cycle accurate results throughout the system by simulating together: (a) the actual application and system software executed on each node; (b) the networks that are employed—COSSIM provides very accurate performance/power/energy consumption estimates for both the processing elements and the network(s) based on the actual dynamic usage scenarios. The simulator also provides the necessary hooks to security testing software, making it possible to determine vulnerabilities and examine the robustness of the system under design. Last but not least, COSSIM employs a standardized interconnection protocol between its sub-components (i.e., IEEE 1516.x HLA[1]); thus, it can be seamlessly connected to additional similar tools, such as simulators of physical processes (e.g., HLA-Enabled Ptolemy [Cardoso and Siron 2018]).

The main research challenges that have been addressed so as to create COSSIM, and which are presented for the very first time in this article, are the following: (a) Develop a novel

[1]IEEE 1516.x: Standard for Modeling and Simulation High Level Architecture (HLA).

synchronization scheme so as to support the notion of cycle accuracy throughout the sub-simulators; (b) add to the synchronization scheme the ability to tradeoff the simulation time against the required timing accuracy; (c) adapt the sub-simulators so as to be able to fully handle real network packets; (d) adapt the sub-simulators as well as the synchronization scheme so as to implement a novel fully distributed system where no critical task and/or sub-simulator is centralized so as to be able to efficiently handle the simulation of a thousand parallel processing nodes interconnected with any available network technology.

2 PARALLEL SYSTEMS SIMULATORS AND THE COSSIM APPROACH

The COSSIM simulator can be utilized mainly in the development of two classes of systems—(a) CPS and (b) Cloud and HPC systems. Starting from the CPS domain, the available tools for CPS simulation fall under two main categories. The first one focuses mostly on the functionality of the system under design and thus they try to model different entities in a CPS system—a physical process, an electromechanical physical component, user behavior, events, message exchanges between components, and so on. Such tools are mostly based on the use of well-defined models of computation. Among them, the most profound ones are Ptolemy [2018], Matlab Simulink [2019], and Modelica-based simulation environments. While these tools can model physical processes, they can only be used during the initial design phase of the CPS application (they can even generate application code) since they mainly offer functional simulation. As a result, they cannot handle cycle-accurate processing simulation, power consumption estimations of the actual processing components, nor the simulation of the actual networks that are going to be employed.

Simulators that do handle those aspects can typically be found in the field of Wireless Sensor Networks (WSNs), which is a certain subset of CPS. Most of these tools are designed around a specific WSN-related operating system or a specific device. For example, TOSSIM [2013] can simulate WSN applications designed for the TinyOS operating system; it can simulate the whole OS stack including the applications developed on nesC, while offering models for specific micro-controllers (AVR ATmega128) along with its peripherals. Similarly, COOJA [2016] is built to simulate sensor networks whose nodes run mainly the Contiki operating system [Musznicki and Zwierzykowski 2012]; it can load binaries generated from other OSs as well. While it can simulate the actual software stack that is executed, it can support only simple micro-controllers; additionally, it can handle a small number of network protocols, while it provides the means to track energy consumption through Energest [Dunkels et al. 2007]. Other WSN simulators [Avrora 2010; Fraboulet et al. 2007] focus on specific micro-controllers or motes (they can model the whole sensor node including sensors and the radio communications chip). These simulators can effectively model both the actual software executed on the processing nodes and the network between the devices; however, their applicability cannot be generalized neither in terms of software (e.g., support for different operating systems or software packages), nor in terms of hardware (e.g., support for other more complicated CPUs) or network (e.g., they can only simulate specific wireless protocols). As a result, those tools cannot handle the simulation of general CPS that requires the modeling of very diverse processing devices together with a multitude of networks.

In the area of Cloud simulators, the most widely used simulation framework is CloudSim [Goyal et al. 2012] and its numerous extensions or derivatives (e.g., CloudSim4DWf [Fakhfakh et al. 2017], CloudTax [Pittl et al. 2017], CloudSimDisk [Louis et al. 2016]). Compared to COSSIM, these simulators are working at a much higher abstraction level using generic non-detailed models; they cannot precisely simulate the actual execution of an application, neither can they model the exact processing hardware utilized. Other cloud simulators focus on specific aspects of the data center that are useful for cloud providers such as, and mainly, resources' provisioning. Those simulators model user and application requirements through stochastic processes or mathematical models in order to predict resource usage and provide optimization insights for avoiding excessive overprovisioning and underutilization (GreenCloud [2017], ElasticSim [Cai et al. 2017], CACTOS [Ostberg et al. 2014]). COSSIM is orthogonal to those systems; the initial results from those tools, for a specific application, can be significantly fine-tuned by executing the application on top of COSSIM and extracting precise application performance and/or power consumption indicators. The COSSIM results can certainly contribute to the optimization of the system and data center design, in terms of performance and/or energy.

Cloud computing is mainly about sharing resources and making efficient use of computational machines by multiple users and applications. Complimentary to the cloud data servers, while also sharing certain aims and features, the parallel systems are intended to execute highly complex and demanding applications (High Performance Computing-HPC systems). The problem in this case is not to design an infrastructure that can be effectively used by multiple users but to make a system of systems that is tuned to execute, in the most efficient (performance, energy or both -wise) manner, a single (or a few) parallel application(s). The advent of applications that require extreme computational resources (such as deep learning, AI, analytics, real-world phsyical/chemical/biological simulations, etc.) both for academic/research and commercial purposes makes the design and deployment of such systems and applications a very interesting and expanding area. In the HPC domain, there are certain tools such as those by Mohammad et al. [2017], which model processing units in a cycle accurate way, while others (e.g., see Ahmed et al. [2017]) that focus on the simulation of real networks; thus, the existing approaches either focus on providing high accurate results per node and more simplistic network models or they replace processing cycle-accurate simulation with application traces or functional modeling and focus on the network part. COSSIM goes beyond those approaches by combining successfully both aspects while providing additional functionality (such as testing the robustness of applications).

There are also the Hornet [Ren et al. 2012] and Sniper [Carlson et al. 2011] simulators, which can simulate multi-core X86 machines with different NoC configurations permitting tradeoffs between accuracy and simulation speed. However, the notion of relaxed synchronization, in those simulators, is different from the one used in COSSIM; in COSSIM, relaxed synchronization is applied in the intercommunication between the processing nodes (through external networks), while Hornet/Sniper employ relaxed synchronization within the processing simulation. Moreover, both Hornet and Sniper can only simulate multi-core systems and not highly parallel distributed systems of systems.

In summary, to the best of the authors' knowledge, no integrated solution exists that can handle the simulation of actual CPS, Cloud, and HPC systems, including their complete software stack and network dynamics. In order to address the lack of such tools, COSSIM efficiently integrates several well-established simulation sub-systems in a single framework that works in the transparent to the user way (i.e., as if it was a single tool rather than a framework). COSSIM is the only known tool that can handle the network, processing, and power simulation in an integrated and fully distributed manner, allowing for much faster and more accurate development.

3 COSSIM DESIGN CHOICES

3.1 Processing Simulator

Highly parallel systems, such as CPS, comprise of a set of nodes and a number of networks that connect those nodes together. The diversity of nodes used in such systems is high; there can be simple micro-controllers that control an actuator device or provide readings from a sensor, network devices, powerful main control units, server systems, and so on. Thus, in order to accurately simulate a processing node, a system simulator that is cycle-accurate, Instruction Set Architecture (ISA) independent, configurable (in terms of supported devices and system features), able to boot real-world operating systems, and execute software compiled for the specified processing node is required.

There is a wide range of open-source processing simulators. Open Virtual Platforms (OVP) [Imperas Software Ltd. 2019] is a high-performance simulator that can simulate advanced multi-core heterogeneous platforms, while it can support the most popular processor families (ARM, MIPS, PowerPC, etc). However, it cannot model any operating system, and it does not provide power models. On the other hand, SimpleScalar [Burger and Austin 1997] is a tool on top of which the designer can perform program performance analysis, detailed micro-architectural modeling, and hardware-software co-verification. It can simulate very complex CPUs (Alpha, x86, ARM, etc.) while it provides means to track energy consumption through the Wattch framework [Brooks et al. 2000]. However, it cannot model or run an operating system.

GEM5 [2019] is a computer system simulation platform that can cover almost all the aforementioned requirements. It can simulate single or multi-core homogeneous or heterogeneous systems including their peripherals. The main engine of the simulator is ISA agnostic, and it can be configured to support a broad range of ISAs. GEM5 does not support natively any power/energy estimations; however, it can be integrated with other tools in order to provide this functionality. The most common power estimator used in tandem with GEM5 is McPAT [Li et al. 2009], while a number of integration tools have been released [Khudia 2014; Endo 2016]. As a result, in order to efficiently simulate all processing aspects, COSSIM employs the GEM5 simulator.

3.2 Network Simulator

Although GEM5 can provide system simulation up to the level of the Network Interface Card (NIC), it does not support network modeling. Therefore, COSSIM employs a dedicated network simulator that handles all network related modeling above the physical layer of a NIC. The simulator selected for this task is OMNET++ [OMNET 2019]. Through OMNET++, different network protocols and topologies can be supported and a realistic network behavior of the parallel system can be modeled; devices such as bridges, switches, and routers integrated in the CPS/HPC/Cloud system developed can also be modeled so as to further increase the simulation accuracy. Through OMNET++'s extensions, it is also possible to estimate the power consumed at the network as well as the radio devices (i.e., radio transceiver power consumption) of the nodes by using the MiXiM add- on [MiXiM 2011].

It should be noted that there exist a number of different alternatives to OMNET++. The most popular ones are the ns2 [2011] and ns3 [2019] network simulators. Similarly to OMNET++, they are also discrete event network simulators, supporting a multitude of network protocols. OMNET++ has been selected mostly because of its wide support of plugins, the reported shorter learning curve than ns2 and ns3, and its higher performance and accuracy [Rehman Khana et al. 2013; Sundani et al. 2011].

3.3 Integration of Components

Binding the aforementioned processing and network simulators together is a very complex task, requiring carefully designed communication interfaces and synchronization schemes. These bidirectional interfaces have to pass information on the type and timing of events as well as to provide a common data representation scheme throughout the framework.

During the last decades, a number of standards for distributed simulation have been developed. Functional Mock-Up Interface (FMI) [FMI 2019] is a tool-independent standard for the exchange of dynamic models and for co-simulation. The primary goal was to support the exchange of simulation models between suppliers and OEMs, while the second goal was the seamless co-simulation using communication technologies like COM/DCOM, CORBA, Windows message-based interfaces, or signals and events. However, FMI does not currently support any communication protocol—something essential for distributed/parallel simulation. Therefore, FMI cannot be considered an appropriate standard for COSSIM's processing and network integration. Moreover, Test and Training Enabling Architecture (TENA) [Noseworthy 2008] is an architecture proposed so as to promote integrated testing and simulation-based acquisition through the use of a real-time synthetic environment, which integrates testing, training, and simulation. However, TENA does not provide integrated Time Management, which is a necessary feature for simulation events because its aim is to integrate real-time systems. Moreover, Distributed Interactive Simulation (DIS) [DIS 1996] and Aggregate Level Simulation Protocol (ALSP) [Wilson and Weatherly 1994] were used to define an interoperability infrastructure for linking simulations of various types at multiple locations so as to create realistic, virtual worlds for the simulation of highly interactive activities. However, DIS and ALSP protocols are currently being replaced by the IEEE High Level Architecture (HLA) [HLA 2010] standard, which unifies their characteristics and mechanisms [Sean P. Griffin and Fischer 1997].

As a result, in order to efficiently integrate the Processing and the Network sub-simulators, COSSIM employs the HLA standard. HLA determines the functional entities, design rules and interfaces for each connected simulation sub-system and specifies the communication between the individual components. It requires a certain Run Time Infrastructure (RTI), which performs event-based tasks that we have used so as to support full synchronization (i.e., having the same notion of global time) across all COSSIM sub-systems. Among the numerous RTI implementations, COSSIM has adopted CERTI [2018], which is an open-source, widely-used RTI implementation.

4 COSSIM ARCHITECTURE

Figure 1 illustrates the COSSIM simulator with all its components and interfaces. Multiple instances of a node simulator module (i.e., a GEM5-based module called cGEM5) are required for the efficient simulation of the numerous processing nodes of a parallel system. The network that binds together the different nodes is simulated by the network simulation module (i.e., OMNET++ based module called cOMNET++). The processing simulation instances are connected with the network one through IEEE HLA-compliant interfaces (interface #1). Additionally, each cGEM5 instance is connected through a custom XML interface with a McPAT instance to estimate the energy/power consumption of each node (interface #2), while cOMNET++ employs internally the MiXIM add-on to estimate the power consumption of the network. Both cGEM5 and cOMNET++ instances have been properly modified to provide hooks that allow security software components to interact with the simulation process so that security tests can be performed during the simulation (interfaces #3, #4, #5).

Figure 1 also illustrates the primary inputs that a user has to provide to the overall system and the primary outputs of the simulation process. Specifically, *System Configuration* (number of CPUs, memory sub-system, peripherals, etc.) have to be defined either using a configuration file or the supplied graphical interface. Furthermore, *image files* of the OSes that will be executed on the nodes of the simulated platform have to be provided by the user. An image file includes the OS kernel, the *application executable,*[2] and all the libraries that are required. Finally, through a *Network & Topology description*, the user can define a network topology (e.g., distance between nodes, topology, mobility between nodes) and describe the interconnection channels and the selected network protocols.

The COSSIM output comprises of the following: (a) *Processing & Network* statistics containing all the measured metrics of the simulation process (e.g., clock ticks, real-time execution, cache

[2]The simulator includes a number of preassembled linux-based OS images in which the user can execute C/C++ and Java applications.

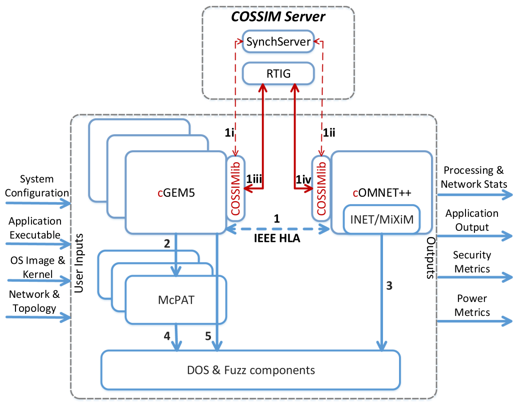


Fig. 1. Top-level view of the COSSIM framework.

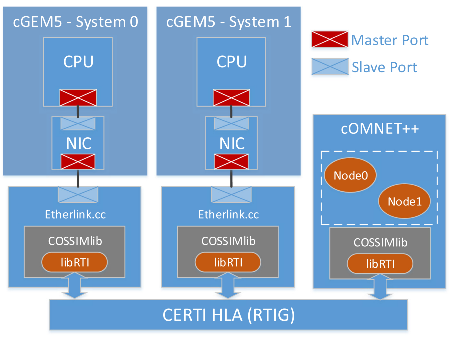


Fig. 2. COSSIM system interconnection.

misses, number of packets sent, number of packets dropped, delay of received packets), (b) the *output of the actual application* that is executed, (c) *security metrics* (e.g., increase in response time, rejection rate, vulnerability density, system robustness), and (d) power and energy estimations for each node (including details such as peak power, runtime dynamic power, leakage, etc.) as well as for the network(s) utilized.

The implemented Interface #1 is responsible for the integration of the processing and the network simulation instances through HLA. The HLA-compliant interfaces that have been added to the GEM5 and OMNET++ simulators implement both the required interconnection and synchronization functions. Interface #1 consists of four sub-parts per node—(i) two responsible for the exchange of the Data Packets and the synchronization of the Processing with the Network subtools and (ii) two responsible for the initialization and synchronization of the overall COSSIM framework through custom-made SynchServer. Finally, *COSSIMlib* is implemented so as to enable the interoperability between cGEM5/cOMNET++ and CERTI/HLA as described in Section 5.3.

4.1 cGEM5

In order for GEM5[3] to serve as the processing simulation sub-system in the COSSIM framework, it has properly been extended to support certain additional features. Based on the COSSIM requirements, the underlying processing simulation sub-system should support the simulation of a CPU including several levels of memory hierarchy and complex peripherals (such as network cards, accelerators, or other components), as well as the execution of a full operating system on top of the simulated device. Currently, cGEM5 can support single and multi-core *ARM* and *X86* architectures, *real network cards*, and a complete *TCP/IP protocol stack* included in a Linux-based OS *Kernel* module with the appropriate drivers. Additionally, cGEM5 can also execute safety-critical systems (such as RTOS) [Namitha Gopalakrishna 2014].

In the following sections, the limitations of the current, publicly available version of GEM5 are described in tandem with the modifications and extensions that have been implemented to alleviate those restrictions.

4.1.1 Extending the Network Model of GEM5. In GEM5's publicly available repositories, the only network interface card implemented, tested, and verified is the Intel 8254x based gigabit Ethernet adapter. It is provided as a PCI GEM5 network device using the e1000 Linux driver.

[3]The adapted version GEM5 for COSSIM will be referred to as *cGEM5*.

However, the latest version of GEM5 supports this real-network device only on ARM-based architectures [Stevenson 2014]. GEM5 has been modified so as to support the Intel 8254x network card for the x86 ISA [Tampouratzis 2016]. On top of that, proper drivers to support it have been built; it should be noted though that the available drivers by Intel required Linux kernel 3.x support; therefore, the COSSIM underlying kernel supporting the network cards had to be custom built since the GEM5 repositories only offered Linux kernel 2.x for x86 simulated systems.

In addition to the network interface cards, GEM5 supports networking through a simple Etherlink device. Etherlink is a virtual dummy link that emulates a cable over which Ethernet packets are sent and received without any delay (no switching or routing functionality is implemented).

In the scope of COSSIM, these limitations are unacceptably restrictive. Therefore, Etherlink could not be used in its current form and thus it has been modified, while NICs supporting more protocols have been developed. Since NIC device models cannot easily be developed without specific information from their manufacturer—the Intel NIC model used in GEM5 has been contributed by Intel itself—and in order to support different physical networks, the COSSIM novel approach is to tap the Ethernet packets from Etherlink and send them to the Networking Simulator, modifying at the same time the packets to match the specific network protocol required by the simulated application (i.e., Ethernet, WiFi, 3G). In more details, the network-related simulation data flow is as follows: (a) in the processing simulator, a NIC is fully modeled and the application is connected to it through its developed device driver; (b) in the network simulator, the network packets produced by the NIC are changed, by our developed microrouters (see below), to any network protocol (without any effect in the simulated time/characteristics) and are accurately and fully handled by the network simulator. The user can alter the characteristics of the NIC if he/she wants to model an NIC with different latency/throughput/and so on, while if he/she is not that interested to simulate the exact NIC characteristics (which is probably the case for most Cloud/HPC developers), the default NIC can be used, and any network topology, technology, and protocol can be accurately simulated within cOMNET++. In such a way, COSSIM can also simulate applications that receive data in user space (without Linux interaction) as long as the application has been linked to the software implementing the NIC interface.

In order to achieve the aforementioned objectives, the CERTI HLA interface [Certi 2018] has been employed. Specifically, the *COSSIMlib* has been integrated with the main core of the GEM5 system through Etherlink. *COSSIMlib* is a wrapper to an RTI library, which is responsible for the exchange of the messages over the network with the HLA Server via TCP and UDP sockets. *COSSIMlib* exchanges Ethernet Packets captured from the Etherlink Device and sends (and accordingly receives) them to (from) the HLA Server. Figure 2 provides an overview of the implemented scheme.

4.1.2 Supporting Parallel/Distributed Simulation. The simplistic network model of GEM5 has another serious limitation. It only supports the simulation of *two identical networked systems* (for example, two identically configured ARM processors with exactly the same peripherals and memory configuration). Furthermore, the simulation of both systems is executed within the same thread; thus, a serious performance penalty is triggered while no synchronization primitives between the two systems are provided. (Recently, authors in Mohammad et al. [2017] provide a distributed GEM5 version that supports only ARM-based processors and a simplistic network model.)

By using HLA-compliant cGEM5 interfaces combined with a network simulator, as described in the previous section, the different cGEM5 instances can be efficiently connected. Each cGEM5 instance models a single node and different GEM5 instances are connected through a simulated network (more precisely, through HLA links and a network simulator). By following this approach, COSSIM overcomes all network related limitations of GEM5 as there are no limitations for identically configured systems, nor is there a restriction on the number of GEM5 systems that participate in the network. A very important additional benefit is that parallelism can be extracted at the process level, since multiple GEM5 instances can be run in parallel. On top of that, as CERTI HLA functions over IP, the different GEM5 instances do not even need to be on the same physical machine, and the overall COSSIM simulator can thus be considered a fully distributed simulator.

4.2 cOMNET++

The key idea behind using a dedicated network simulator in COSSIM is to be able to support multiple network protocols, topologies, and devices through which nodes (represented by cGEM5 instances) can be interconnected. OMNET++ has been chosen as the most capable and feature-rich network simulator in that context; however, it also has certain limitations. OMNET++ does not support the real protocol stacks (e.g., Linux ones) as GEM5 does; therefore, in order to efficiently interconnect the two simulators, a module for encapsulating/decapsulating cGEM5 binary packets into OMNET++-compatible packets is designed. In addition, in order to support different network protocols (e.g., WiFi protocols) within OMNET++, micro-router modules are implemented as described in the next paragraphs.

4.2.1 HLA Enabled Node Functionality. cGEM5 instances are connected to cOMNET++ through HLA. For each cGEM5 instance, a node is created within cOMNET++, called HLA-Enabled node. These nodes can transparently communicate with cGEM5 through a designed HLA RTI wrapper, while encapsulating/decapsulating network packets in a comprehensible form for both simulators. Within OMNET++, the HLA-Enabled nodes can communicate with any normal OMNET++ node, e.g., a router or a switch; thus, any kind of topology can be supported.

In this respect, the COSSIM approach is based on having each simulated node consisting of two parts: the processing part (simulated in cGEM5) and the network part (handled in cOMNET++), which communicate through the HLA interfaces. Each of the HLA Enabled Nodes has a minimum functionality that allows them to communicate with their counterpart on the cGEM5 side. The network sub-system of the COSSIM simulator is designed to model the complete network related behavior of each simulated node along with all the network characteristics. The upper protocol stack (from Layer 2 and above) is accurately simulated from the processing sub-system.

More specifically, a CERTI-HLA-compliant wrapper has been developed and integrated within cOMNET++, offering an interface to each node simulated in cGEM5, supporting consistent communication and synchronization. Moreover, since the goal of COSSIM is to provide cycle-accurate simulation, the whole Linux protocol stack is executed within cGEM5, thus producing a fully RFC-compliant binary IP packet. In order for these packets to be forwarded, they are encapsulated properly by an HLA-node that is designed and instantiated by us inside the cOMNET++ L2/L3 packet structure.

The first step in designing this HLA-node, was to modify the standard cOMNET++ cPacket structure so as to include payload data. The second step was to develop a custom-fit functionality that is automatically inherited in each of the HLA enabled nodes of the simulation to seamlessly convert the cGEM5 packets to cOMNET++ packets and vice versa. As a result, in COSSIM, each packet sent from the cGEM5 subsystem into the cOMNET++ one passes through a sequential de-capsulation procedure (i.e., parsing) from the Ethernet/L2 level to the L3 level (the IP level) followed by an encapsulation procedure in the cOMNET++ protocol stack. In the opposite direction (from cOMNET++ to cGEM5), all the L2/L3 fields are properly de-serialized into a single RFC-compliant binary packet (i.e., a valid Ethernet packet) that forms the payload for the HLA channel as illustrated in Figure 3.

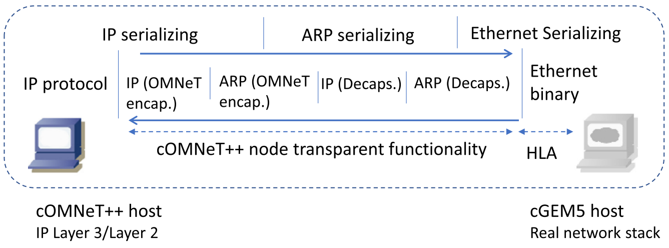


Fig. 3. The En/De-capsulation vs. serialization process inside OMNET++ HLA nodes.

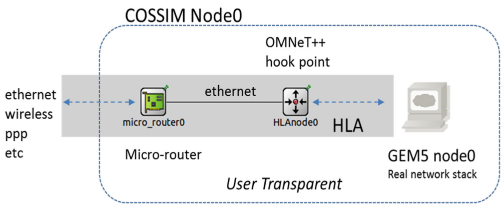


Fig. 4. COSSIM node with micro-router support.

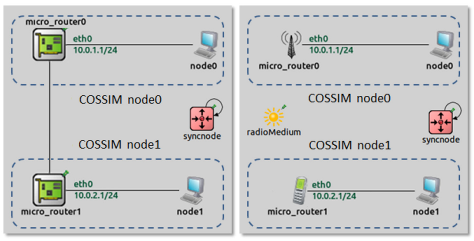


Fig. 5. Ethernet and Wireless Network simulation with the micro-router functionality.

4.2.2 Transparent Micro-Routers Functionality. Currently, there are no other Network Interface Cards supported by GEM5 besides Ethernet ones. To be able to support different network protocols (e.g., LoRa, 5G), within OMNET++, a micro-router module has been implemented allowing the user to seamlessly change the physical layer from Ethernet to any other option supported by OMNET++. In this way, the cGEM5 nodes always see an "Ethernet attached" network on the cOMNET++ side, but the actual network employed within cOMNET++ is fully user-defined. The implemented COSSIM transparent micro-routers have an Ethernet interface (100% compatible with the Linux protocol stack) and are connected to the cGEM5 node counterpart, while their second interface is interchangeable through the OMNET++/COSSIM Graphical User Interface (GUI). The micro-routers are implemented in such a way that they do not alter the timing characteristics of the packets in terms of the simulated time in any way, or, in other words, they work transparently, and they do not affect the timing accuracy of the simulator.

To demonstrate the functionality of the mico-routers, as shown in Figure 4, HLAnode0 is the attachment point of the cGEM5 node and the OMNET++ simulated infrastructure. On the left of the figure, micro_router0 is responsible for changing the network interface from Ethernet to wireless, ppp, and the like. Specifically, the micro-router is implemented as an INET StandardHost enhanced with routing capabilities and multiple interfaces. Figure 5 (left) depicts a scenario where two HLA enabled nodes are connected via an Ethernet link, while Figure 5 (right) depicts a scenario where the COSSIM micro-router links an HLA enabled node to a wireless interface without affecting the cGEM5 configuration. In both configurations, the IP assignment between the OMNET++ nodes can be set either automatically or manually.

4.3 Integration of Security Tools

The COSSIM simulation framework integrates a number of tools that allow the CPS designer to evaluate the security of the system simulated. COSSIM's security module is an extension of COSSIM, which consists of two main sub-modules—the *DoS Testing System* (DTS) and the *Fuzz Testing System* (FTS).

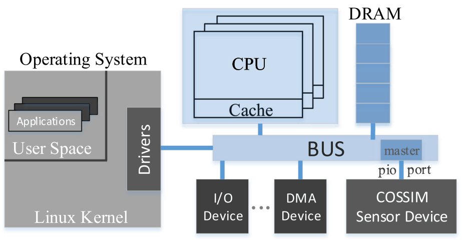


Fig. 6. Integration of COSSIM Sensor Device with cGEM5 (full-system mode).

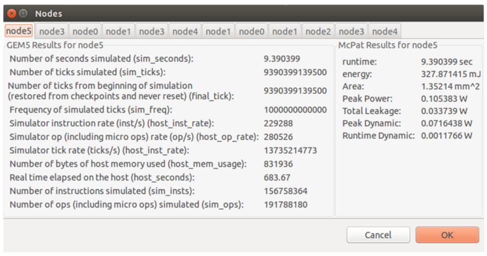
The DTS is responsible for testing the resilience of a parallel system under simulation against various types of denial-of-service (DoS) attacks. As DoS is a network attack to CPUs, DTS is tightly connected with both the network simulation components of the COSSIM framework and the Node simulator, via HLA. In the scope of DTS, a Resource Monitor Detector is implemented utilizing cOMNET++'s facilities for collecting statistics data; those data are then inspected, analyzed, and transformed by DTS's embedded visualizers and statistics tools. In more details, the detector gets the network data through cOMNET++'s gates through which messages and datagrams travel back and forth. As a result, when the monitor detects some kind of anomaly regarding the monitored node's resources, the detector can instantly filter or modify the packets passing through it and generate different black and white lists based on modern approaches for packet filtering, as well as queuing theory.

The FTS provides automated testing of components' interfaces for discovering vulnerabilities. It produces input test vectors that are fed into the parallel system/application under test. This process (fuzzing) is capable of exposing errors that arise as a result of the processing of these input vectors. Specifically, FTS is made up of two sub-systems: the *FTS Client* and the *FTS Server*. The *FTS Server* is responsible for catching the data on the hooked interfaces, providing them to the *FTS Client*, fetching back the modified/fuzzed values, and continuing the execution with these modified data fields. On the other hand, the *FTS Client* manages the fuzzing effort; it has interfaces for providing the test vectors for the data fields that the *FTS Server* provides id's for. The *FTS Server*, requires to start and restore GEM5 simulations and run the test vectors from the same checkpoint, (e.g., the same processor state) so as to be able to handle clean test cases. For this reason, it utilizes the pause-resume functionality of COSSIM, which allows all the simulation components to be paused and resumed at/from the same simulated time.

4.4 Simulation of Physical Parts of a CPS

In the case of CPS simulation, COSSIM simulates the cyber part rather than the physical part of the system; in order to support the efficient simulation of the physical part as well, COSSIM provides two main options. The first one, extends GEM5 by adding support for sensor devices in the processing system through programmed I/O using one memory bus master port to read the sensor values. To incorporate efficiently different sensors within cGEM5, a set of device drivers has been developed, while a specific ioctl [2017] function has been implemented so as to achieve efficient user-kernel space communication as illustrated in Figure 6.

In addition, the COSSIM framework can be seamlessly interconnected with any other HLA-compliant approaches simulating the physical aspects of the system (e.g., Ptolemy II [2018]). Specifically, our tool can be connected with any other HLA-compliant tools in order (for each processing node) to have access to the physical processes that "generate" the sensing data. Regarding the physical aspects that relate to the network performance, those are already included in COSSIM


Nodes
node5 node3 node0 node1 node3 node4 node1 node0 node1 node2 node3 node4
Cancel
OK

GEM5 Results for node5

Number of seconds simulated (sim_seconds):	9.390399
Number of ticks simulated (sim_ticks):	9390399139500
Number of ticks from beginning of simulation (restored from checkpoints and never reset) (final_tick):	9390399139500
Frequency of simulated ticks (sim_freq):	1000000000000
Simulator instruction rate (inst/s) (host_inst_rate):	229288
Simulator op (including micro ops) rate (op/s) (host_op_rate):	280526
Simulator tick rate (ticks/s) (host_inst_rate):	13735214773
Number of bytes of host memory used (host_mem_usage):	831936
Real time elapsed on the host (host_seconds):	683.67
Number of instructions simulated (sim_insts):	156758364
Number of ops (including micro ops) simulated (sim_ops):	191788180

McPat Results for node5

runtime:	9.390399 sec
energy:	327.871415 mJ
Area:	1.35214 mm^2
Peak Power:	0.105383 W
Total Leakage:	0.033739 W
Peak Dynamic:	0.0716438 W
Runtime Dynamic:	0.0011766 W

Fig. 7. Node results for a specific combination of nodes.

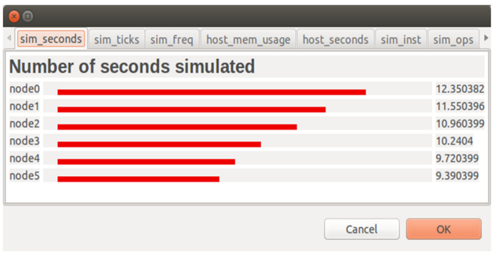

sim_seconds sim_ticks sim_freq host_mem_usage host_seconds sim_inst sim_ops
Number of seconds simulated
node0 12.350382
node1 11.550396
node2 10.960399
node3 10.2404
node4 9.720399
node5 9.390399
Cancel
OK

Fig. 8. Comparison of "Number of seconds simulated" value.

through the OMNET++ simulator, which offers an extension that models the physical world regarding the reaction between the physical and network processes. For example, a designer can seamlessly model, within COSSIM, the movement of the nodes in the space, obstacles in the wave propagation, attenuation, noise, re-transmissions, and the like.

4.5 Eclipse-based GUI

The Eclipse-based GUI of OMNET++ has been extended so as to fully integrate the capabilities of the COSSIM tool. Specifically, the COSSIM GUI consists of two Eclipse plugins: (i) the simulation configuration tool and (ii) the execution monitoring tool.

The simulation configuration tool has the form of a wizard that is installed as a plugin in Eclipse/OMNET++ and guides the user through the cGEM5 configuration process for each of the simulated nodes. This process was a very time-consuming one, as it was usually performed through the command line and there was a need for a large number of parameters that have to be set for each of the nodes. Based on our initial measurements within the COSSIM design team, the GUI, by itself, reduces the configuration time of the simulation by 90% in the case of a 10-node system. In addition, our GUI prevents the user from setting wrong parameters and thus minimizing the risk of starting a time-consuming simulation that will latterly be proven wrong or inadequate.

The second plugin that has been developed is the COSSIM execution monitoring tool, which is a graphical interface that integrates and visualizes the most important results from the COSSIM simulations; the output results can be presented either per node or per complete simulated parallel system. Figures 7 and 8 illustrate the COSSIM results for a certain simple application scenario with six nodes.

5 NOVEL INTERCOMMUNICATION AND SYNCHRONIZATION MECHANISM

One of the main novel aspects of COSSIM is the developed intercommunication and synchronization scheme, which is fully compliant with the IEEE HLA standard.

5.1 Overview of HLA

HLA is a standard for distributed discrete-event simulations, generally used to support analysis, engineering and training; it has been developed so as to promote reusability and interoperability. In HLA terminology, the logical representation of an interconnection of different simulators is called a Federation and includes multiple modules, which are called Federates, and which communicate via an RTI. The RTI provides a number of services like Federation Management, Time Management, and Object Management that are utilized in simulation control, synchronization, and data exchange [Roth et al. 2011]. The connection between the simulators, based on the HLA standard, is established by a federate library that encapsulates the HLA interfaces and which can be

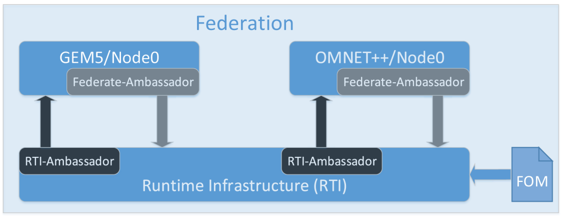


Fig. 9. The High Level Architecture (HLA).

specialized to certain simulation scenarios; the communication from the RTI to a federate and vice versa is established via the RTI-Ambassador and the Federate-Ambassador modules, which cause a strict segregation of simulation while supporting several communication primitives as illustrated in Figure 9. Specifically, Figure 9 presents one Federation for each cGEM5 instance (Federate 1) together with the communication with the cOMNET++ counterpart node (Federate 2), while the Federation Object Model (FOM) contains the description of the data exchange process within the federation.

HLA services are grouped into six groups based on where they are utilized in the federate life cycle. Table 1 describes the subset of HLA services used in COSSIM together with a brief description of each service. The reader is referred to [HLA 2010a, 2010b] for a complete description of all HLA services. The services in Table 1 cover the following aspects:

(1) ***Federation management***—Defines how federates can connect to the RTI; create, join, and manage federations; save and restore federation states; and defines a system implementing an accurate synchronization scheme.
(2) ***Declaration management***—Defines how federates declare their intentions with regard to publication and subscription of classes and interactions.
(3) ***Object management***—Defines how federates can utilize objects and interactions.
(4) ***Time management***—Defines how time is used in a federation and how it affects object and interaction updates.

5.2 COSSIM HLA Services

The novel COSSIM HLA time management services enable, for the first time in the area of highly parallel system simulation, deterministic, cycle accurate and reproducible distributed simulations providing dual-stage synchronization scheme (Section 5.4). Each federate manages its own logical time and communicates this time to the RTI. The RTI ensures correct coordination of federates by advancing time coherently. Logical time is roughly equivalent to “simulation time” in the classical discrete event simulation terminology, and is used to ensure that federates observe events in the same order [Lee 2011]. Logical time is not necessarily mapped to real time.

(1) ***Federate’s time policies:*** HLA time policies describe the involvement of each federate in the progress of time. It may be necessary to map the progress of one federate to the progress of another. There are two sets of federates: A *regulating federate* participates actively in the decisions for the progress of time while a *constrained federate* follows the time progress imposed by other federates. A combination of both federates is also allowed by HLA.
(2) ***Time progress:*** Time advancement requests by federates are made through the HLA *timeAdvanceRequest service* (TAR), which is used to implement time-stepped federates, while the granted time is provided by *timeAdvanceGrant* (TAG) callback. The time

Table 1. HLA Services for COSSIM Implementation

	Services	Description
Federation	createFedExecution()	Create a federation
	destroyFedExecution()	Unregister a federation
	joinFedExecution()	Participate in a federation
	resignFedExecution()	Resign a federation
	registerFedSynchPoint()	Register a Synch Point
	synchPointRegSucceeded() *	Reg Synch point succeeded
	announceSynchPoint() *	Wait a Synch Point
	synchPointAchieved()	Release form a Synch Point
	federationSynchronized() *	Announce Synchronization
	tick()	Allow to deliver callback
Declaration	publishInteractionClass()	Declare publication of a class
	unpublishInteractionClass()	Unpublish a class
	subscribeInteractionClass()	Subscribe to receive Inter
	unsubscribeInteractionClass()	Unsubscribe to receive Inter
Object	sendInteraction()	Generate Interaction event
	receiveInteraction() *	Deliver the Interaction event
Time	enableTimeRegulation()	Federate is regulator
	timeRegulationEnabled() *	Regulator succeeded
	enableTimeConstrained()	Federate is constrained
	timeConstrainedEnabled() *	Constrained succeeded
	queryFederateTime()	Query curr. time from RTI
	timeAdvanceRequest(), TAR	Requests an advance of time
	timeAdvanceGrant(), TAG *	Time Advancement granted

(Services with a * are ent from RTI to Federates (callbacks))

Table 2. Overview of COSSIMlib Architecture

	GEM5/OMNET++ Attributes	CERTI bindings
Pre-Initialize	join()	createFederationExecution joinFederation
	publish()	publishInterctionClass
	subscribe()	subscribeInteractionClass
Initialize	setTimeRegulation()	enableTimeRegulation timeRegulationEnabled * enableTimeConstrained timeConstrainedEnabled *
	pause()	registerFederationSynchPoint announceSynchPoint *
	synchronize()	synchronizationPointAchieved federationSynchronized *
Interaction	step()	queryFederateTime timeAdvanceRequest timeAdvanceGrant *
	sendInteraction()	sendInteraction
	getPacket()	receiveInteraction *
Final	resign()	resignFederationExecution destroyFederationExecution

advancement phase of a Federate F in HLA is a three-step process: (i) F sends a request using the *TAR* service; (ii) F can receive interactions (Data Packets) using *receiveInteraction* callback; (iii) F waits for the granted time *tG* (TAG). At the *TAG*(tG) reception, the federate's local time will be advanced to *tG* according to the data in the *TAR* request.

The time management scheme and its semantics are shown in Figure 10. As demonstrated in this example, a federate produces an event at time *t1* sending one DataPacket using *sendInteraction()*. The current (logical) time is *t1*; let us consider that the next event is *e2* with timestamp *t2*, $t2 = t1 + \Delta$, $\Delta > 0$. The federate asks the RTI whether the time should be progressed with the invocation of *TAR(t2)*. Until the reception of the *TAG(t2)* callback, the logical time of the federate is stalled at *t1*, and it can receive a *ReceiveInteraction(v,t1')* callback. Timestamp *t1'* is in the interval [*t1, t2*]. Subsequently, Federate can execute a computation step with the values received within the *ReceiveInteraction* callback. The federate then receives *TAG(t2)* and can increase its local time to *t2*. In a nutshell, if a *TAR(t2)* has been sent, the time granted by the *TAG* service is $tG = t2$.

5.3 COSSIMlib Arhitecture

The implemented *COSSIMlib* enables the interoperability between cGEM5/cOMNET++ and the CERTI/HLA Federations. It is built on top of the CERTI API, a C++ binding of CERTI based on HLA version 1.3. Since the HLA communication mechanisms are based on TCP/IP packets, both HLA Server (RTIG) and SynchServer can be executed either in the same physical machine or in physically-entangled servers. The same applies for all the components of the simulator (i.e., each cGEM5 instance and cOMNET++), thus enabling fully distributed simulation of large parallel

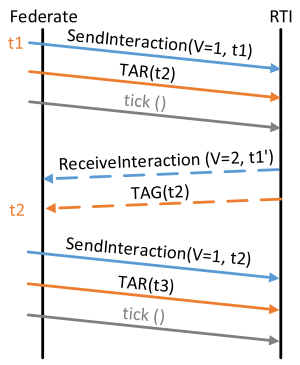


Fig. 10. Time management services.

systems. Table 2 summarizes the implemented methods in COSSIMlib and the corresponding HLA services of Table 1.

The pre-initialize methods of the *COSSIMlib* implement certain services of the Federation management (e.g., create and join tasks for a federation) and some services of the Declaration management that have to do with the publication and subscription of object's instances in a federation. Specifically, three basic methods are implemented—*Join*, which is responsible for creating a Federation and joining the Federate to this Federation; *Publish*, which is responsible to publish the interaction class so as to allow for the sending of Data Packets; *Subscribe*, which is responsible for the subscription to the interaction classes, in order to receive Data Packets.

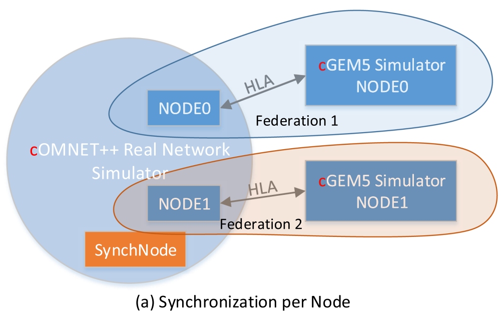
The Initialize methods of the *COSSIMlib* implement certain services of the Federation and Time management, which are relevant to the HLA synchronization and time policy (e.g., constrained and regulating federates). Specifically, three basic methods are implemented—*SetTimeRegulation* to declare the federate policy; *Pause* to initiate the establishment of a named checkpoint that serves to synchronize all federates according to federation-defined semantics; *Synchronize* to declare that all other federates (in the same federation) have joined.

The Send & Receive Interaction methods of the *COSSIMlib* implement certain Time management and Object management Services. Specifically, three basic methods are implemented that support the send and receive Interactions within the HLA Federations: *SendInteraction* sends Data Packets from cGEM5/cOMNET++ Federate to the CERTI server; *ReceiveInteraction*, which inherits the *receiveInteraction* callback decodes and reads the *DataPacket* whenever is triggered; *Step*, which requests an advance of the logical time of the federate to a specified federation time using the *timeAdvanceRequest* HLA service. Finally, the *Resign* (finalize) method is implemented, which is related to the *Declaration* and *Federation* management services, and which is responsible for the correct termination (i.e., resignation) of the federates as well as the destruction of a federation.

5.4 Synchronization between COSSIM Modules

Within COSSIM, we have developed a novel dual-stage synchronization scheme consisting of the following sub-schemes:

(1) ***Synchronization per node.*** Each cGEM5 node needs to communicate, in a consistent way, with its corresponding node in the network simulator (i.e., cOMNET++) and exchange data packets. This type of synchronized communication is necessary because certain network data between the two simulators must be exchanged in a manner that will preserve the exact time ordering. For this reason, one federation is created per node pair so as to support full synchronization as illustrated in Figure 11(a).



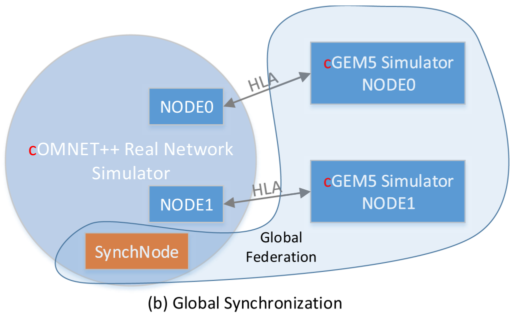


Fig. 11. COSSIM HLA federations.

Table 3. Synch per Node Intervals for Typical Network Bandwidths

Network Bandwidth	Synch per node Interval
100Mbit	100us
1Gbit	10us
10Gbit	1us

Table 4. Simulation Speedup vs. Accuracy Using Different Global Synch Interval (**1Gbit**)

Global Synch Interval	Network Accuracy (%)	Simulation Speedup
1ms	5.36%	3.84x
100us	25.28%	2.67x
10us	**99.63%**	1x

Table 5. Simulation Speedup vs. Accuracy Using Different Global Synch Interval (**10Gbit**)

Global Synch Interval	Network Accuracy (%)	Simulation Speedup
100us	10.56%	5.63x
10us	31.89%	2.82x
1us	**98.68%**	1x

(2) ***Global Synchronization.*** The COSSIM simulator needs to periodically synchronize all nodes. This is because it supports different types of CPUs with potentially different clock cycles and/or different network protocols, all resulting in varying workload for the simulators' engines. Therefore, the simulated time (i.e., the time aspect of the modeled system) in each node can be completely different for the same wall clock time. For this reason, a Global Synchronization federation is developed to achieve a unified notion of time that contains all cGEM5 nodes and one OMNET++ helper Node (SynchNode) as illustrated in Figure 11(b). The SynchNode is a normal user-space instantiated node (as the rest of the HLA Enabled Nodes) inside OMNET++ that follows the standard Node structure; as a result, it is 100% compatible with OMNET++.

The *Synchronization time per node* and the *Global Synchronization time* are two different entities that can be separately defined by the user. The first one is mostly defined by the latency of the network interface, and it doesn't constrain the simulation speed while the second is a tradeoff between simulation speed and simulation accuracy as demonstrated in the validation and performance section. More specifically, Table 3 presents the *Synchronization per node interval* for typical Network bandwidths, while Tables 4 and 5 present the tradeoff between network accuracy and simulation speed when using different Global Synchronization intervals for 1Gbit and 10Gbit network links, respectively. It should be noticed that both synchronization intervals affect the network accuracy while the processing simulation accuracy is not altered as GEM5 is cycle-accurate by default. The relaxed synchronization feature of COSSIM is supported in case the user is less interested in the accuracy (i.e., if some network packets are dropped/not handled) and more interested in the speed of the simulation process (e.g., in the initial design space exploration phase).

The proposed global synchronization scheme is responsible for the preservation of the, unique, cycle-accurate notion of the COSSIM simulation process. OMNET++ is natively an event driven simulator, however, by employing the global synchronization scheme, it becomes hooked to the

Table 6. Systems Used to Evaluate COSSIM Simulator

Machine #	Type	CPU Model	Total Cores	RAM	OS
1	Workstation	Intel i5-4590	4 Physical	16GB	Ubuntu 16.04
2	Server1	2x Intel Xeon E5-2440v2	16 Physical	64GB	CentOS 7.2
3	Server2	2x Intel Xeon E5-2440v2	16 Physical	64GB	CentOS 7.2
4	Cluster	44x AMD Opteron 2300	176 Physical	176GB	Ubuntu 14.04

"cycle-events" of each of the cGEM5 simulated nodes. This not only prevents the clocks of all the nodes from any drift but also implicitly "forces" the cOMNET++ to act like a "cycle-driven" event simulator. In this respect every component of the simulated system has the exact same notion of time (i.e., in terms of clock cycles).

6 VALIDATION AND PERFORMANCE ANALYSIS

The evaluation of the COSSIM simulation framework has been initially performed using light versions of two Linux distributions—Gentoo Base System for x86 processors and BusyBox for ARM ones. In both systems, the Ubuntu-minimal package and the JRE7 are installed so as to enable the execution of C, C++, and Java applications (thus, resembling a realistic deployment scenario).

6.1 Evaluation of Processing Simulator Part

COSSIM utilizes a modified version of the mainstream GEM5 simulator. However, all modifications are mainly related to the Ethernet interface and the developed HLA components. As such, the main computation engine of the GEM5 simulator is left intact and, therefore, the performance of the cGEM5 component of the COSSIM simulator is in line with the reported performance of the publicly available version of GEM5. GEM5's performance varies greatly with the complexity of the processing system that is being simulated. A typical simple CPU (InOrder CPU) can be simulated at a rate of 1 to 3 Million Instructions/Sec (MIPS). More complex CPU structures (e.g., Out of Order CPUs) and memory subsystems can reduce the rate of simulated instructions to as low as 0.1–0.3 MIPS [Saidi 2012; Carlson 2015].

The key concept of the COSSIM approach is to execute the cOMNET++ simulator in a typical workstation (this is not a bottleneck as demonstrated in the next paragraphs) so as to easily utilize the GUI that facilitates the orchestration and visualization of the simulation, while the cGEM5 instances are executed in one or more servers (distributed simulation). For this reason, in the following experiments, the cOMNET++ simulation was executed on a workstation based on an Intel i5-4590 processor (Machine 1). On the other hand, the simulation of the CPUs as well as the HLA Server were executed on two servers based on four Intel Xeon E5-2440v2 (totally 32 physical cores with 128GB of RAM—Machines 2&3) as described in Table 6.

Figure 12 illustrates the mean instructions per second simulated when Mobile Visual Search[4] is executed in the COSSIM system including the time needed for the negotiations and setup of the corresponding networks interconnecting the different nodes for both X86 and ARM processors; it should be noted that those numbers do not differ between the numerous CPU-bound applications we have executed on the COSSIM platform. As this figure demonstrates, as the number of nodes increases, the rate of simulated instructions (thus, performance) decreases. This is mainly due to the Global Synchronization schemes for the 2–32 node experiments, while there is a steep performance drop when more than 32 cGEM5 instances are executed on the 32 physical cores due

[4]Mobile Visual Search is a computer vision application built on the idea of retrieving interesting information about physical objects.

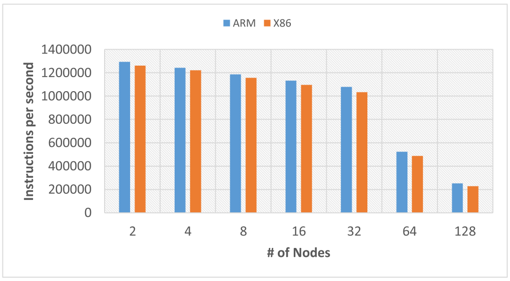


Fig. 12. Performance results using typical GEM5 configuration.

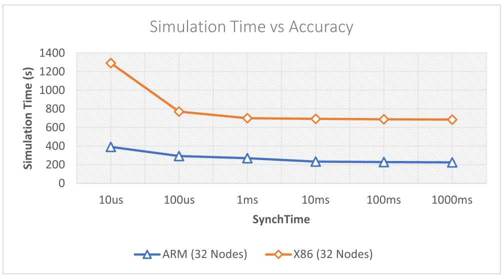


Fig. 13. Simulation time using different synchronization intervals.

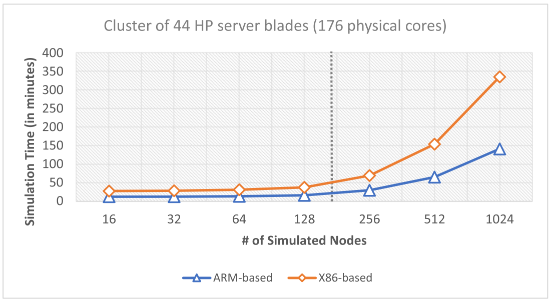


Fig. 14. Simulation time using a cluster of 44 server blades.

to lack of processing resources. For the specific experiment, the synchronization interval is set at 10ms, which gives us a good tradeoff of accuracy and simulation speed, since the focus of this experiment is to investigate the scalability of our approach, which is not affect by the interval value. Consequently, after every 10ms of simulated time, all nodes are halted so that they can all be synchronized and then resume operation. Apparently, this synchronization interval determines the accuracy of captured events. A short synchronization interval will yield the most accurate results at the cost of lower overall performance, while a more relaxed (i.e., longer) synchronization interval will result in no practical slowdown due to synchronization, at a potential loss of accuracy.

Figure 13 depicts the impact of the synchronization interval in the performance of the overall simulation, as measured by the wall-clock time required to complete a simulation with 16 nodes. There is a dramatic drop in performance for very short interval times (smaller than 100us), mainly due to the increased number of messages that cOMNET++ has to manipulate.

Furthermore, Figure 14 illustrates the simulation time required to boot the OSs with their network card configuration for 16–1024 ARM-based nodes and X86-based nodes using a 10us synchronization interval. For this experiment, a cluster that contains 44 *HP Proliant BL465c* server blades with totally 176 physical cores is used (Table 6). Based on those results, it becomes evident, as in the previous figure, that performance is heavily impacted when the number of cGEM5 instances spawned becomes higher than the number of physical processor cores of the distributed system on which the simulation is executed. Specifically, in the 16–128 nodes experiments, the simulation time is almost constant, while, after 176 cores, doubling the number of cGEM5s instances triggers an increase in the simulation time by a factor of two. Summarizing, the effect of the network has a negligible impact on the overall performance; as a result, COSSIM has good scalability in case the number of cGEM5 instances is lower or equal to the number of the physical cores of the underlying system.

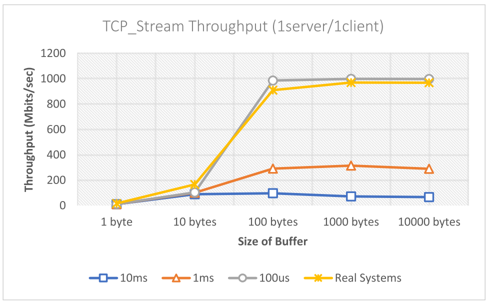


Fig. 15. TCP_Stream throughput for 1server/1client using different Global Synchronization intervals.

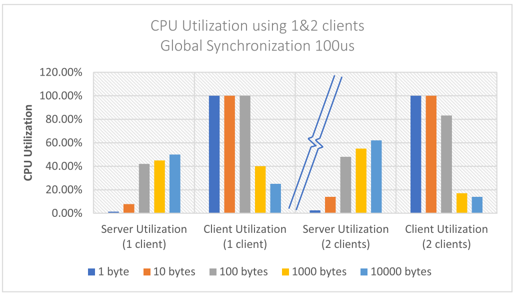


Fig. 16. CPU Utilization using TCP_Stream for 1server/1client & 1server/2clients experiments.

6.2 Evaluation of Network Simulator Part

The network performance of COSSIM's novel approach has been evaluated using the widely used Netperf 2.7.0 benchmark suite and, specifically, the very demanding TCP_STREAM and TCP_RR benchmarks. Netperf [HP Networking Performance Team 2018] was developed by Hewlett-Packard; it is widely used in measuring the performance of many different types of networks and it provides tests for throughput and end-to-end latency.

6.2.1 TCP_STREAM Benchmark. The TCP_STREAM test is the most widely-used test in netperf. It transfers certain data from the system where netperf is executed to that running netserver. Figure 15 illustrates the throughput (Mbits/sec) achieved on this benchamark in the case of two COSSIM simulated x86 systems and two real x86 systems; in both cases, the simulated and real systems are connected through Gigabit Ethernet. Specifically, 5MB[5] of total data are exchanged in each experiment, while different buffer sizes are utilized in the "send" calls of the test. Command 1 shows the netperf TCP_STREAM configuration, which is used for 100 bytes buffer size.

```
netperf -H IP -t TCP_STREAM-c -C -l -5000k -- -m 100        (1)
```

As described in Section 5.4, COSSIM Synchronization is based on two different entities that can be separately defined by the user; for the TCP_STREAM experiments, the synchronization per node interval is set at 10us (i.e., each node is guaranteed to receive at most one Data Packet every 10us). This interval is selected due to the Gigabit Ethernet interconnection between the nodes. Specifically, since the maximum Ethernet packet is 1536 bytes [eth 2016], by selecting a synchronization period of 10us, each node can receive up to 1536 bytes/10us, or 153.600.000 bytes/sec, or 1.228.800.000 bits/sec, so it can support full Gigabit Ethernet speed. On the other hand, three different intervals have been tested for the Global Synchronization, from 100us to 10ms. Figure 15 illustrates that as the Global Synchronization decreases, more accurate results are obtained (i.e., much closer to the actual performance of the real system), while using 100us as the Global Synchronization interval, triggers very accurate results for all buffer sizes. Furthermore, as the number of buffer messages increases the throughput of the application, both the simulated and the real system moves toward the maximum possible one.

CPU utilization is an important and very often overlooked component of the overall networking performance. Unfortunately, it can be one of the most difficult metrics to measure accurately and portably. Netperf is one of the most accurate benchmarks for measuring CPU utilization [HP Networking Performance Team 2018]. CPU utilization in netperf is reported as a value between 0 and

[5]As adequate quantity of data, 5MB was selected to reach the maximum speed of experiment.

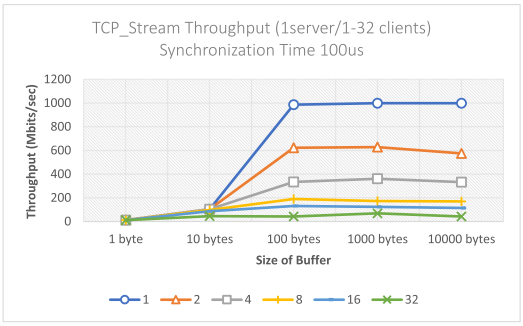


Fig. 17. TCP_STREAM Throughput using 1server and 1–32 clients.

100% regardless of the number of CPUs involved; for a TCP_STREAM test, it is the microseconds of CPU time consumed to transfer 1024 bytes of data.

Figure 16 shows the CPU utilization (in the simulated node) for 1-server/1-client and 1-server/2-clients experiments using a Global Synchronization interval of 100us, which is adequate for full speed simulation, and Synchronization per node of 10us. Figure 16 demonstrates that as the buffer size is less than 10 bytes, the CPU utilization is steadily at 100% for both 1 client and 2 clients experiments (i.e., they are CPU-bound). This is reasonable because the clients need to truncate the 5MBs in a lot of small packets, while the throughput achieved is much lower than the maximum one (Figure 15). In addition, as the buffer size gets larger and up to 1000 bytes, the CPU utilization is much lower and the experiment becomes network-bound. The 2-client experiment needs less CPU processing in each client CPU because the packets are sent at lower rates while the server CPU utilization is similar to that of the 1-client with small differences due to the higher number of total packets the server gets.

Figure 17 illustrates the throughput for the TCP_Stream experiments using one server and from 1 up to 32[6] clients. In that configuration, a router has been configured with 2–33 Gigabit Ethernet ports to establish a star network topology. Figure 17 shows that as the number of clients increases, the throughput decreases because the server has to serve all clients. The maximum throughput achieved by our simulator using 1000-bytes Ethernet packets is ranging from 998Mbits/sec to 69Mbits/sec for 1 and 32 clients, respectively.

6.2.2 TCP_RR Benchmark. Request/Response performance is often overlooked, but it is just as important as bulk-transfer performance for several applications. While things like larger socket buffers and TCP windows, as well as stateless offloads like TSO (TCP segmentation offload) and LRO (Large Receive Offload) [Corbet 2007] can cover numerous latency and even path-length sins, those sins do surface in the request/response tests. While in a bulk-transfer test the reporting metric is the units of bits transferred per second, the TCP_RR test reports the transactions per second where a transaction is defined as the completed exchange of a request and a response.

Figure 18 illustrates the transactions per second in the case of two COSSIM-simulated x86 systems and two Real x86 systems; the configuration consists of one server and one client running the netperf TCP_RR benchmark and connected through Gigabit Ethernet. In more detail, different request/response packet sizes have been evaluated ranging from 1 byte to 10K bytes within 1[7]

[6]In this case, COSSIM is evaluated for up to 32 clients, due to physical cores bound; the aim of this experiment was to evaluate the system when assigning one cGEM5 instance per physical core so as not to suffer from the performance degradation due to lack of processing power.
[7]One second was selected since it allows the simulated system to reach each maximum performance/speed.

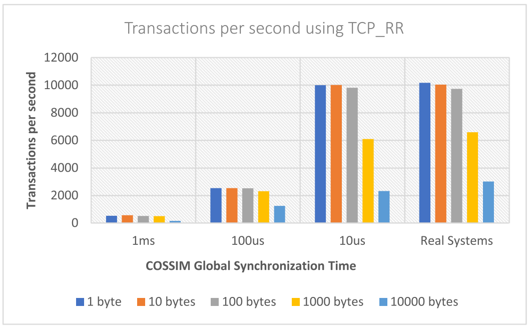


Fig. 18. Transactions per second using 1server/1client with different Global Synchronization intervals.

Table 7. Transmission Rate of TCP_RR Using Different Global Synchronization

	1ms	100us	10us	Real Systems
1 byte	1,883.2 us/pkt	395.8 us/pkt	99.9 us/pkt	98.3 us/pkt
10 bytes	1,757.4 us/pkt	395.9 us/pkt	99.8 us/pkt	99.6 us/pkt
10^2bytes	1,972.3 us/pkt	396.9 us/pkt	101.8 us/pkt	102.6 us/pkt
10^3bytes	2,012.7 us/pkt	434.7 us/pkt	163.9 us/pkt	151.7 us/pkt
10^4bytes	680.7 us/pkt	801.2 us/pkt	431.1 us/pkt	331.8 us/pkt

second of total time. Command 2, below, shows the netperf TCP_RR configuration, which is used for a request/response packet size of 100 bytes.

```
netperf   -H IP   -t TCP_RR   -l 1   --   -r 100,100        (2)
```

In all the TCP_RR experiments, the synchronization per node interval is set at 10us so as to achieve full Gigabit Ethernet rates (as described in the previous paragraph) between the nodes. On the other hand, three different intervals are selected for Global Synchronization from 10us to 1ms. Figure 18 illustrates that as the Global Synchronization decreases, more accurate results are obtained (i.e., much closer to the actual performance of the real system); when using a Global Synchronization of 10us, the transactions/second is very similar to those achieved by the Real Systems for all request/response sizes. TCP_RR requires a smaller Global Synchronization interval (10us) than the TCP_STREAM experiment (100us) in order to obtain very accurate results due to the smaller length of each packet in TPC_RR (i.e., higher number of packets are sent per second). Finally, Table 7 summarizes the transmission rate of the above systems; the highest transmission rate for the simulated system is achieved using 1-byte request/response packets; when we use a 10us Global Synchronization interval we can simulate this high bandwidth system with very high accuracy—99.9 microseconds/packet are Reported by COSSIM against the 98.3 microseconds/packet measured in the Real-nodes.

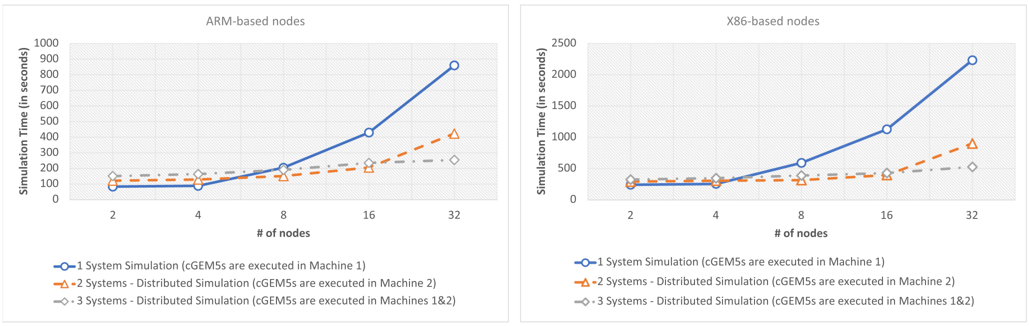
Both netperf experiments demonstrate that with a relatively low Global Synchronization Interval, COSSIM can extremely accurately simulate both network-bound and CPU-bound applications.

6.3 Performance Evaluation of Distributed COSSIM Simulator

In this section, the performance of the COSSIM simulator is presented when simulating multiple processing systems in multiple distributed machines. In addition, the performance of COSSIM is also demonstrated when the only centralized module (i.e., HLA Server) is also replicated and executed in a distributed manner, thus making the complete COSSIM framework fully distributed.

The performance of our novel COSSIM simulation framework has been further analyzed on the distributed scenario described in Section 6.1 (three Systems—one workstation and two servers).

Specifically, Figure 19 illustrates the performance of the simulator when the main COSSIM components (cOMNET++, HLA servers, and all cGEM5 instances) are executed on the same physical machine (machine 1—straight line) and when they are executed in the distributed scenario (machine 2 and machines 2 and 3—dashed lines). Figure 19(a) illustrates the simulation time required to boot the OSs with their network card configurations for 2–32 ARM-based nodes, while Figure 19(b) for 2–32 X86-based nodes.



(a) ARM-based nodes (b) X86-based nodes

Fig. 19. COSSIM simulation time using Distributed Machines.

As clearly demonstrated in Figure 19, performance is heavily impacted when the number of cGEM5 instances spawned becomes higher than the number of physical processor cores present in the underlying machine (i.e., there is a steep performance drop when more than four cGEM5 instances are executed on the 4-core machine 1, and more than 16 cGEM5 instances are executed on one of two 16-core servers).

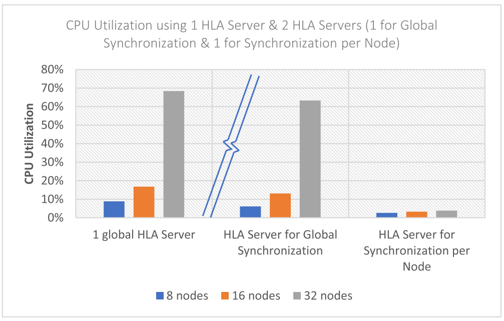
On the other hand, the effect of the network simulation has a negligible impact on the overall simulation performance. Specifically, in 2 and 4 nodes experiments, Machine 1 is faster than the distributed system due to the faster single thread execution on the Intel i5-4590 of Machine 1 when compared with that of the Intel E5-2440v2 CPUs of the two servers. When the number of cGEM5 instances increases, the single server setup is actually faster for up to 16 cGEM5 instances (i.e., equal to the number of the physical cores); the slight performance degradation in the 2-server setup is due to the fact that there is an additional overhead for the packets exchanged between the two servers. However, in the 32-nodes experiment, the 2-server setup, with its higher available physical cores, surpasses the single-server one by approximately 70%. Similar results are obtained when simulating X86-based nodes, which require more processing time to boot the OS (Figure 19(b)). Summarizing, when there are 32 cGEM5 instances and they are executed in the 32 physical cores of the two servers, the overall simulation time is almost constant for all simulated nodes (each cGEM5 node is executed per physical core) achieving up to 337% and 423% speedup (for ARM and X86 based architecture respectively) when compared with the case that all cGEM5 instances are executed on the same 4-physical core/8-virtual-core CPU.

6.3.1 *Evaluation of COSSIM Utilizing Multiple HLA Servers.*

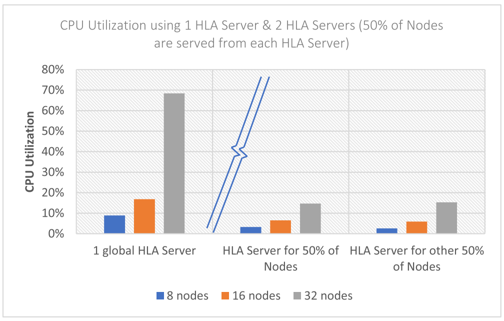
The only centralized sub-system of COSSIM is, so far, the HLA Server; so we use the TCP_RR benchmark to evaluate its performance when sending a very large number of messages, and the Global Synchronization Interval selected is short (i.e., for high accuracy). Specifically, we trigger the worst-case scenario by sending 1-byte request/response packets and having a 10us Global Synchronization Interval.

Figure 20 (left) illustrates the CPU Utilization of the HLA Server (using one Global HLA Server for both Global Synchronization and all the necessary Synchronizations per node) for the experiments described in the last paragraphs. As clearly demonstrated, the CPU load grows significantly with the number of nodes simulated. As a result, and in order to be able to simulate thousands of nodes in a fully distributed manner, we develop a version of our HLA Server, which can also be executed in a decentralized manner. In particular, we have performed two experiments, utilizing two identical HLA Servers in two different scenarios.

In the first scenario, one HLA Server is used for the Global Synchronization, and the other for all the node synchronizations. In this case, the CPU Utilization on the first HLA Server, due to the



(a) 1 HLA Server for Global Synch & 1 for Synch per node



(b) 1 HLA per 50% of nodes

Fig. 20. HLA CPU utilization using 1 HLA and 2 HLA servers.

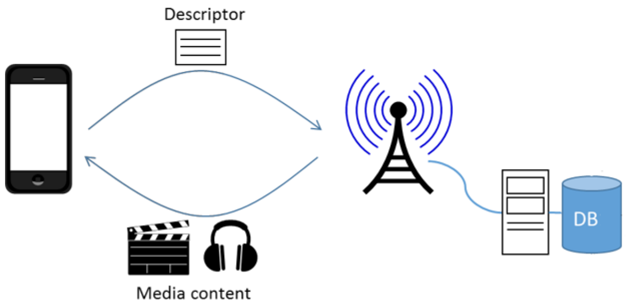


Fig. 21. Mobile visual search topology.

small Global Synchronization interval, is kept high, while the second HLA Server is under-utilized (Figure 20(a), right).

In the second scenario, we use the unique feature of our approach, which allows to distribute even the Global Synchronization part without any synchronization deficiency. So one HLA Server serves 50% of the simulated nodes (both the Global and per node Synchronization schemes), and the other, the remaining 50%. In this case, the CPU Utilization of both nodes is very low, as shown in Figure 20(b), right. Those results clearly demonstrate that COSSIM is a fully-distributed framework, with no centralized components; thus, it can efficiently handle thousands of simulated nodes if/when executed on a multi-thousand core machine.[8]

6.4 Evalaution on a Real-world Application/System

Our simulator has also been evaluated when simulating Mobile Visual Search (MVS) [Paracchini et al. 2016], which is a real-world distributed application. MVS is built on the idea of retrieving interesting information about physical objects using only the image content; multimedia content can be sent back in response to a search or to a user's action. The aim of MVS is to analyze a query image taken by the user and search for similar images inside a large database as illustrated in Figure 21. MVS is already used in many different contexts such as interactive museum guides, e-commerce mobile applications, localization systems, and so on.

The MVS algorithm is composed of two interconnected modules: the Image Analyzer (IA) and the Retrieval Stage (RS). Firstly, in the IA, a series of advanced image processing techniques are

[8]Network simulation in cOMNET++ is several orders of magnitude faster than processing simulation in cGE5, so we believe that there is no need to parallelize cOMNET++.

Table 8. Global and Local Matching Scores on the Building Database

Query	1st result	2nd result	3rd result
Native			
Local Score	93.72	60.21	13.06
Global Score	315.759	63.202	50.172
Simulated			
Local Score	93.72	60.21	13.06
Global Score	315.759	63.202	50.172

applied on the image acquired by the user in order to localize the interesting regions of it (i.e., key-points). Each selected key-point is then analyzed and, for each one of them, a local descriptor is extracted. The last operation of the IA is the aggregation of the computed local descriptors into a compact global descriptor, which describes the whole image. These descriptors are concatenated and further compressed in order to reach a size of around 15–20 KB for each VGA image. The whole IA stage can be executed in the user's mobile device.

On the other hand, the RS is responsible to compare the image descriptor created in the IA with a plethora of other descriptors extracted offline from each image in the database. In the first step of RS, the images in the database are quickly ranked in respect to the similarity of their global descriptor to the one computed from the query image. In this way, only a short list of the images in the database are selected as possible matches. A more precise comparison using local descriptors, including geometry checks and outliers rejection, is performed and the most similar image to the query one is finally selected. The RS requires access to the whole database, and for this reason, it is executed on a server.

The real system that is modeled/simulated by COSSIM consists of:

— One central node with an Intel Core i5-5200 2.2 GHz quad Core CPU and 4 Gbytes of RAM
— Two sets of ARM-based devices, one consisting of Cortex-A15 2.0 GHz quad core CPUs (Odroid XU3) and another one consisting of Cortex-A7 1.4 GHz quad core CPUs; all systems have 2Gbytes of RAM.

Having two different ARM-based devices let us compare the capabilities of COSSIM when simulating both an in-order CPU (A7) and an out-of-order one (A15). In addition, the Odroid XU3 comes integrated with the power analysis tool that was modified and used by us in order to collect the data reported in the next sections. The MVS cloud application simulated by COSSIM consists of:

— One central node with X86 2.2 GHz CPU and 4GB of RAM
— Mobile nodes with two possible different processors—In-order ARM working at 1.4 GHz CPU, and Out-of-Order ARM working at 2 GHz CPU; both of them have 2 GB of RAM.
— Wireless Network (WiFi)

6.4.1 Verification of COSSIM Functionality. In order to verify the correctness of the simulation process, the final output of the simulation was compared with the output of the native system. In particular, the local and global descriptor scores extracted from the ordered list of results were compared. Table 8 reports the top three matches for the query image on the left. As it can be

Table 9. MVS Simulated Time Comparison Using Two Different Synchronization Intervals (1ms & 10us)

	Simulated ARM In-Order		**Native ARM A7**	**Simulated Out-of-Order**		**Native ARM A15**	**Simulated X86**		**Intel i5-5200**
Synch Interval	**1ms**	**10us**	**-**	**1ms**	**10us**	**-**	**1ms**	**10us**	**-**
Image 1	50,687ms	57,945ms	58,244ms	14,499ms	15,478ms	15,613ms	645ms	522ms	508ms
Image 2	66,782ms	88,236ms	89,568ms	21,459ms	22,951ms	23,229ms	641ms	484ms	498ms
Image 3	50,782ms	60,936ms	61,481ms	14,437ms	15,627ms	15,855ms	644ms	516ms	502ms
Average	56,084ms	**69,039ms**	**69,764ms**	16,798ms	**18,018ms**	**18,232ms**	643ms	**507ms**	**501ms**

Table 10. MVS Simulated Time Comparison Using Two Different Synchronization Intervals (1ms & 10us)

	ARM A7 Energy (mJ)		**ARM A7 Peak Power (W)**		**ARM A15 Energy (mJ)**		**ARM A15 Peak Power (W)**	
	Simulated	**Native**	**Simulated**	**Native**	**Simulated**	**Native**	**Simulated**	**Native**
Image 1	1,060.85	1,381.52	0.132	0.12	692.66	753.71	1.503	1.54
Image 2	1,733.85	1,982.53	0.172	0.14	1,041.13	1,236.84	1.518	1.55
Image 3	887.62	899.63	0.142	0.12	693.60	771.97	1.503	1.54
Average	**1,227.44**	**1,421.22**	**0.148**	**0.123**	**809.13**	**954.17**	**1.508**	**1.54**

observed, all the result images depict the same building as the query one. For each result, we report the local (result of local matching) and global (result of global matching) scores in the simulated and real scenarios. The simulation replicates exactly the computations performed by the native system, as stated by the equivalent values of the native and the simulated search results.

6.4.2 Timing Results. The simulation involves the complete MVS application composed of several mobile imaging nodes and a central one. As stated, the query image is first processed on the mobile device in order to extract some descriptors, thus creating a compact representation of it, and then it is sent to the central node to be compared with all the other descriptors extracted from each image in the database, in order to find visually similar images. In this configuration, only compressed information about the image (descriptor) are sent through the network from the mobile device to the central node.

Table 9 lists the simulated time of both ARM A7 In-Order, ARM A15 Out-of-Order and X86 distributed nodes. In this experiment, two different synchronization intervals are used (1ms and 10us) in order to prove the accuracy of our simulator. As shown in Table 9, more accurate results are obtained when using smaller synchronization intervals; when the synchronization interval is equal to 10us, the simulated time reported by COSSIM is very close to the measured time that we get from the native system for all CPU models.

6.4.3 Energy/Power Results. Table 10 compares the Energy and Peak Power of the simulated and the native ARM A7 and ARM A15 based nodes. We can observe that in the majority of the cases, and also on average, the ARM energy consumption estimated in the simulation is quite close to the real one. Finally, it should also be noticed that the peak power consumption is very accurate (average error 0.025W (min: 0.012W | max: 0.032W) on ARM A7 and 0.03W (min: 0.032W | max: 0.037W) on ARM A15).

7 CONCLUSIONS AND FUTURE WORK

In this work, we present the COSSIM Simulation Framework, an open-source novel solution that addresses, in a single integrated toolset, the simulation of both the processing and the networking parts of highly parallel systems, while also taking into account the security and the power consumption aspects of them. By using a novel dual-stage synchronization scheme, COSSIM is the first known system providing global cycle accurate simulation when/if required, highly parallel/distributed execution, and high extensibility; COSSIM can also operate as a cycle-approximate simulator so as to allow for faster simulation times. Based on a number of experiments, the COSSIM simulator proved to produce very accurate results for both network and processing heavy applications, while it is probably the only simulator that can efficiently simulate a thousand processing nodes, interconnected together, in a full distributed manner; the claimed efficiency is denoted by the fact that in order to simulate a thousand nodes, COSSIM needs only 5x more time than when only 16-nodes are simulated.

As future work, and in order to further extend the functionality of the Simulation Framework in the CPS domain, we aim to integrate COSSIM with a widely used physical process simulator (i.e., Ptolemy) by using the already developed HLA interface. Additionally, we have already started working on accelerating COSSIM by implementing certain parts of it into reconfigurable systems.

In order to further increase the impact of our work, the complete source code with its documentation are freely distributed to the HPC/Cloud/CPS community [COS 2019].

REFERENCES

2010a. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—Federate interface specification. *IEEE Std 1516.1–2010 (Revision of IEEE Std 1516.1–2000)* (Aug 2010), 1–378. DOI: https://doi.org/10.1109/IEEESTD.2010.5557728

2010b. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—Framework and rules. *IEEE Std 1516–2010 (Revision of IEEE Std 1516–2000)* (Aug 2010), 1–38. DOI: https://doi.org/10.1109/IEEESTD.2010.5553440

2016. IEEE standard for Ethernet. *IEEE Std 802.3–2015 (Revision of IEEE Std 802.3–2012)* (March 2016), 1–4017. DOI: https://doi.org/10.1109/IEEESTD.2016.7428776

2019. COSSIM—A novel, highly integrated Simulator for Parallel and Distributed Systems. Retrieved from https://github.com/H2020-COSSIM.

Kishwar Ahmed, Jason Liu, Abdel-Hameed Badawy, and Stephan Eidenbenz. 2017. A brief history of HPC simulation and future challenges. DOI: https://doi.org/10.1109/WSC.2017.8247804

Avrora 2010. AVRORA - AVR Simulation and Analysis Framework. Retrieved from http://tinyos.stanford.edu/tinyos-wiki/index.php/Avrora.

David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News* 28, 2 (May 2000), 83–94. DOI: https://doi.org/10.1145/342001.339657

Doug Burger and Todd M. Austin. 1997. The SimpleScalar tool set, Version 2.0. *SIGARCH Comput. Archit. News* 25, 3 (June 1997), 13–25. DOI: https://doi.org/10.1145/268806.268810

Zhicheng Cai, Qianmu Li, and Xiaoping Li. 2017. ElasticSim: A toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. *Journal of Grid Computing* 15, 2 (1 Jun 2017), 257–272. DOI: https://doi.org/10.1007/s10723-016-9390-y

Janette Cardoso and Pierre Siron. 2018. *Ptolemy-HLA: A Cyber-Physical System Distributed Simulation Framework: Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday.* 122–142. DOI: https://doi.org/10.1007/978-3-319-95246-8_8

Trevor Carlson. 2015. Full-System Simulation at Near Native Speed. Retrieved from http://www.gem5.org/wiki/images/4/4f/2015_ws_11_20150614_-_Trevor_E._Carlson_-_gem5_workshop.pptx.

T. E. Carlson, W. Heirman, and L. Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–12. DOI: https://doi.org/10.1145/2063384.2063454

CERTI 2018. CERTI Project. Retrieved from http://savannah.nongnu.org/projects/certi.

COOJA 2016. COOJA Simulator. Retrieved from https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja.

Jonathan Corbet. 2007. (2007). Retrieved from https://lwn.net/Articles/243949/.

1996. IEEE standard for distributed interactive simulation—Application protocols. *IEEE Std 1278.1–1995* (1996), i–. DOI: https://doi.org/10.1109/IEEESTD.1996.80831

Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. 2007. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets'07).* ACM, New York, NY, 28–32. DOI: https://doi.org/10.1145/1278972.1278979

Endo 2016. GEM5ToMcPAT parser tool. Retrieved from https://github.com/markoshorro/gem5McPATparse.

F. Fakhfakh, H. H. Kacem, and A. H. Kacem. 2017. CloudSim4DWf: A CloudSim-extension for simulating dynamic workflows in a cloud environment. In *Proceedings of the 2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*. 195–202. DOI: https://doi.org/10.1109/SERA.2017.7965728

FMI 2019. Functional Mock-up Interface. Retrieved from http://fmi-standard.org/.

A. Fraboulet, G. Chelius, and E. Fleury. 2007. Worldsens: Development and prototyping tools for application specific wireless sensors networks. In *Proceedings of the 2007 6th International Symposium on Information Processing in Sensor Networks*. 176–185. DOI: https://doi.org/10.1109/IPSN.2007.4379677

GEM5 2019. The GEM5 Simulator. Retrieved from http://gem5.org/.

Tarun Goyal, Ajit Singh, and Aakanksha Agrawal. 2012. Cloudsim: Simulator for cloud computing infrastructure and modeling. *Procedia Engineering* 38 (2012), 3566–3572. DOI: https://doi.org/10.1016/j.proeng.2012.06.412 INTERNATIONAL CONFERENCE ON MODELLING OPTIMIZATION AND COMPUTING.

Greencloud 2017. The Greencloud Simulator. Retrieved from https://greencloud.gforge.uni.lu.

HLA 2010. IEEE 1516-010—Standard for Modeling and Simulation High Level Architecture—Framework and Rules. Retrieved from https://standards.ieee.org/findstds/standard/1516-2010.html.

HP Networking Performance Team 2018. *Network Performance Benchmark*. HP Networking Performance Team. Retrieved from https://hewlettpackard.github.io/netperf/.

Imperas Software Ltd. 2019. *Open Virtual Platforms*. Imperas Software Ltd. Retrieved from http://www.ovpworld.org/.

IOCTL 2017. *ioctl—Control Device*. ioctl. Retrieved from http://man7.org/linux/man-pages/man2/ioctl.2.html.

Khudia 2014. GEM5ToMcPAT conversion tool. Retrieved from https://bitbucket.org/dskhudia/gem5tomcpat/src/master/.

Edward A. Lee. 2011. Heterogeneous actor modeling. In *Proceedings of the 9th ACM International Conference on Embedded Software (EMSOFT'11)*. ACM, New York, NY, 3–12. DOI: https://doi.org/10.1145/2038642.2038646

S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 469–480.

B. Louis, K. Mitra, S. Saguna, and C. Åhlund. 2016. CloudSimDisk: Energy-aware storage simulation in CloudSim. In *Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. 11–15.

Mixim 2011. MiXIM Simulator. Retrieved from http://mixim.sourceforge.net/.

A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim. 2017. dist-gem5: Distributed simulation of computer clusters. In *Proceedings of the 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 153–162. DOI: https://doi.org/10.1109/ISPASS.2017.7975287

Bartosz Musznicki and Piotr Zwierzykowski. 2012. Survey of simulators for wireless sensor networks. In *International Journal of Grid and Distributed Computing*. 23–50.

Namitha Gopalakrishna. 2014. RTOS 2014. *Execution Time Analysis of Audio Algorithms*. MSc Thesis. Computer Engineering Department of Electrical Engineering Faculty of EEMCS Delft University of Technology.

J. R. Noseworthy. 2008. The test and training enabling architecture (TENA) supporting the decentralized development of distributed applications and LVC simulations. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*. 259–268. DOI: https://doi.org/10.1109/DS-RT.2008.35

Ns2 2011. The Network Simulator—ns-2. Retrieved from http://nsnam.sourceforge.net/wiki/index.php/User_Information.

Ns3 2019. ns3 Network Simulator. Retrieved from https://www.nsnam.org/.

OMNET 2019. OMNET++ Discrete Event Simulator. Retrieved from https://omnetpp.org/.

M. Paracchini, M. Marcon, E. Plebani, and D. P. Pau. 2016. Visual search of multiple objects from a single query. In *Proceedings of the 2016 IEEE 6th International Conference on Consumer Electronics—Berlin (ICCE-Berlin)*. 41–45. DOI: https://doi.org/10.1109/ICCE-Berlin.2016.7684712

B. Pittl, W. Mach, and E. Schikuta. 2017. CloudTax: A CloudSim-extension for simulating tax systems on cloud markets. In *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 35–42.

Ptolemy 2018. The Ptolemy Project. Retrieved from http://ptolemy.eecs.berkeley.edu/.

Atta Rehman Khana, Sardar M. Bilalb, and Mazliza Othman. 2013. A performance comparison of network simulators for wireless networks. *ArXiv abs/1307.4129* (2013).

P. Ren, M. Lis, M. H. Cho, K. S. Shim, C. W. Fletcher, O. Khan, N. Zheng, and S. Devadas. 2012. HORNET: A cycle-level multicore simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 6 (June 2012), 890–903. DOI: https://doi.org/10.1109/TCAD.2012.2184760

Christoph Roth, Oliver Sander, Matthias Kühnle, and Jürgen Becker. 2011. HLA-based simulation environment for distributed SystemC simulation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools'11)*. 108–114.

Ali Saidi. 2012. Accelerating Simulation with Virtual Machines. Retrieved from http://gem5.org/File:2012_12_gem5_workshop_kvm.pdf.

C. Zachary Furness Sean P. Griffin, Ernest H. Page and Mary C. Fischer. 1997. Providing uninterrupted training to the joint training confederation (JTC) audience during transition to the high level architecture (HLA). In *Proceedings of the Simulation Technology and Training Conference*. http://www.thesimguy.com/articles/simtect97.pdf.

Simulink 2019. Model-based design of cyber-physical systems in MATLAB and Simulink. Retrieved from https://www.mathworks.com/discovery/cyber-physical-systems.html.

Pete Stevenson. 2014. GEM5 mailing List, 2014. X86 full system dual. Retrieved from http://www.mail-archive.com/gem5-users@gem5.org/msg09897.html.

Haoyue Sundani, Vijay K. Li, Mansoor Devabhaktuni, Prabir Alam, Bhattacharya, Harsh Sundani, Haoyue Li, Vijay Devabhaktuni, Mansoor Alam, and Prabir Bhattacharya. 2011. Wireless sensor network simulators: A survey and comparisons. *International Journal of Computer Networks* 2 (January 2011).

Nikolaos Tampouratzis. 2016. GEM5 mailing List, 2014. X86 full system dual Solution. Retrieved from https://www.mail-archive.com/gem5-users@gem5.org/msg12680.html.

TOSSIM 2013. TinyOS SIMulator. Retrieved from http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM.

A. L. Wilson and R. M. Weatherly. 1994. The aggregate level simulation protocol: An evolving system. In *Proceedings of the Winter Simulation Conference*. 781–787. DOI: https://doi.org/10.1109/WSC.1994.717433

P. O. Östberg, H. Groenda, S. Wesner, J. Byrne, and D. S. Nikolopoulos et al. 2014. The CACTOS vision of context-aware cloud topology optimization and simulation. In *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. 26–31. DOI: https://doi.org/10.1109/CloudCom.2014.62

Received May 2019; revised October 2019; accepted January 2020