This is a repository copy of *SEED: A Scalable Approach for Cyber-Physical System Simulation*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/92917/

Version: Accepted Version

Article:

Garraghan, P, McKee, D, Ouyang, X et al. (2 more authors) (2016) SEED: A Scalable Approach for Cyber-Physical System Simulation. IEEE Transactions on Services Computing, 9 (2). pp. 199-212. ISSN 1939-1374

https://doi.org/10.1109/TSC.2015.2491287

Reuse

Unless indicated otherwise, fulltext items are protected by copyright with all rights reserved. The copyright exception in section 29 of the Copyright, Designs and Patents Act 1988 allows the making of a single copy solely for the purpose of non-commercial research or private study within the limits of fair dealing. The publisher or other rights-holder may allow further reproduction and re-use of this version - refer to the White Rose Research Online record for this item. Where records identify the publisher as the copyright holder, users can verify any specific terms of use on the publisher's website.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

SEED: A Scalable Approach for Cyber-Physical System Simulation

Peter Garraghan, David McKee, Xue Ouyang, David Webster, Jie Xu. *Member, IEEE*

Abstract— Simulation is critical when studying real operational behavior of increasingly complex Cyber-Physical Systems, forecasting future behavior, and experimenting with hypothetical scenarios. A critical aspect of simulation is the ability to evaluate large-scale systems within a reasonable time frame while modeling complex interactions between millions of components. However, modern simulations face limitations in provisioning this functionality for CPSs in terms of balancing simulation complexity with performance, resulting in substantial operational costs required for completing simulation execution. Moreover, users are required to have expertise in modeling and configuring simulations to infrastructure which is time consuming. In this paper we present SEED (Simulation EnvironmEnt Distributor), a novel approach for simulating large-scale CPSs across a loosely-coupled distributed system requiring minimal user configuration. This is achieved through automated simulation partitioning and instantiation while enforcing tight event messaging across the system. SEED operates efficiently within both small and large-scale OTS hardware, agnostic of cluster heterogeneity and OS running, and is capable of simulating the full system and network stack of a CPS. Our approach is validated through experiments conducted in a cluster to simulate CPS operation. Results demonstrate that SEED is capable of simulating CPSs containing 2,000,000 tasks across 2000 nodes with only 6.89x slow down relative to real time, and executes effectively across distributed infrastructure.

Index Terms— Distributed Systems, Distributed Simulation, Cyber-Physical Systems, Service-oriented Simulation.

1 INTRODUCTION

SIMULATION of Cyber-Physical Systems (CPSs) is becoming increasingly important in modern day systems for engineering, research, and commercial pursuits. Given the scale and complexity of modern CPSs such as Cloud datacenters, Smart Cities, and Internet of Things, it has become more feasible from an economical and scientific perspective to build and study simulation models that represent accurate system operation prior to constructing physical prototypes of the production system. Simulation based prototyping has seen recent adoption in manufacturing to speed up the product development process and support early evaluation, and are increasingly used within CPSs to evaluate early mock-ups which are modelled in sufficient detail [1]. Furthermore, manual testing of complex systems is subject to limitations in its coverage and effectiveness in comparison to performing simulation [3]. Compared to physical prototyping, the quality of simulation based prototyping is less sensitive to parameterization choices due to the ability to detect problems early and correct accordingly [4]. Such models allow users to experiment with a wide range of operational scenarios, alternative technologies and reengineering business processes without the requirement for high expenditure costs needed for implementation [1][2]. Specifically, it enables the ability for users to verify and validate the capabilities of a modeled system supported through context specific use cases of interest including operational efficiency, current design optimization, provide means to understand and reduce risks associated with system expansion and alteration, and tolerate different failure scenarios [17].

Performance becomes a key consideration when simulating large quantities of system components with complex interactions due to performance degradation [18]. This is primarily due to limitations in resource availability [7] as well as centralized approaches when managing event synchronization [11]. Given that CPS can be composed of potentially millions of component interactions, centralized approaches for simulation face significant challenges in providing results in a timely manner.

Simulating large-scale complex systems through parallel and distributed simulation has gained traction in recent years. This enables model distribution across a number of parallel and distributed compute nodes to take advantage of additional memory and CPU capacity, increasing simulation scale and reducing simulation execution time [37]. A common approach to managing these simulations is through the use of Parallel Discrete Event Simulation whereby a simulation is partitioned across a set of compute nodes and is managed through discrete timesteps and message passing through events generated from each partition [37]. An effective means to mitigate scalability issues in simulating CPSs is to decompose the simulation into smaller physically distributed logical units, and can be achieved through the use of high power tightly-coupled systems [14][15] or large-scale distributed infrastructure configured to facilitate specific simulation [5][19].

However, there are a number of challenges which reduce the effectiveness of such approaches. In the context of HPC, simulations are imposed with system timing and resource constraints, dictated by scheduling practices [6], with such systems requiring expensive infrastructure acquisition and maintenance. On the other hand, distributed systems require bespoke configuration for a specific type of simulation to run effectively, and are predominately deployed across multiple server racks. Both of these approaches rely on advanced domain user expertise to configure their simulation to specific hardware within an infrastructure and development environment which can be time consuming and expensive [10]. Furthermore, a key

The authors are with the School of Computing, University of Leeds, Leeds, LS2 9JT, United Kingdom.
E-mail:{p.m.garraghan,scdwm,scxo,d.e.webster,j.xu}@leeds.ac.uk

component in distributed simulation is effective event synchronization across all components and simulation partitions. While a number of existing distributed simulators provide different levels of synchronization [16][37], they do so at the cost of significant slowdown relative to real world time performance; a substantial issue when simulating lengthy time periods. Such behavior results in degraded simulation performance leading to increased operational costs in terms of resource usage and energy consumption to complete simulation execution. Finally, both approaches may not be readily available to a large body of researchers and developers due to prohibitive infrastructure requirements and expertise to operate.

As a result, in order to overcome these challenges, there is a substantial need to develop approaches that can effectively simulate large-scale CPSs fulfilling the following criteria:

– *Minimal domain user configuration and programming between their native modeling environment and running within a distributed simulator.* This not only allows for rapid deployment and execution of simulation, but also allows the simulator to be provided as a SaaS model due to clear possession and ownership between software and its use [12].
– *Distributed simulations that can execute with little to no assumptions about the underlying simulator hardware.* Specifically, there are significant advantages to using simulator infrastructure composed of heterogeneous readily available Off the Shelf (OTS) components that can run simulation agnostically across different machine architectures and Operating Systems (OS).
– *Supports event synchronization while maintaining acceptable levels of performance relative to real world time*. As simulation scale increases in terms of required infrastructure scale and simulation complexity, performance becomes an increasing concern for both users and system administrators in terms of business requirements and operational costs, respectively.

To facilitate this, Service-oriented Architecture (SOA) - which supports open standards, loose coupling, discoverability and reusability for the integration of distributed systems [13] - appears to be a promising approach in providing an effective means to address the above criteria. This is particularly true for *reducing the coupling between simulation and the underlying hardware*, as well as *transforming distributed simulators into SaaS offerings* which can be accessed through generic protocols globally.

This paper presents SEED (Simulation EnvironmEnt Distributor), a novel service-oriented approach for simulating CPSs across a distributed system. This is achieved through automated simulation partitioning, instantiation and execution across a loosely-coupled infrastructure while enforcing tight event messaging across the system. Users are capable of interacting with SEED as a Windows Communication Foundation (WCF) Web Service using XML-based protocols, and are not required to configure underlying hardware for execution. As a result, SEED operates efficiently within both small and large-scale OTS hardware, agnostic of cluster heterogeneity and OS running on nodes. Furthermore, it is capable of simulating the full system and network stack of a CPS, and captures key characteristics of task execution. Our approach is implemented and validated in a real physical cluster to simulate large-scale CPS operation composed of up to 2,000 simulated nodes and 2,000,000 executing tasks, as well as task and user characteristics from a production CPS.

This paper is structured as follows: Section 2 presents the background; Section 3 discusses the related work; Section 4 presents the SEED system architecture and functionality; Section 5 presents the implementation, clock manager and clock management; Section 6 presents the evaluation of SEED; Section 7 examines the practicality of the approach using a real-world use case; Finally, Section 8 presents the conclusions and future work.

2 Background

2.1 Simulation

Simulation forms a key mechanism for verifying and validating existing and new approaches within many computer science domains. In this context, simulation is defined as the time dependent imitation or emulation of a real world system for the purposes of analyzing and evaluating its design or process [21]. A simulation model of a CPS is composed of numerous components, including server *Nodes*, *Network Links,* and *Tasks* to be executed. Simulations that imitate the behavior of physical systems larger than what is available to hand are particularly effective, however they incur a slowdown with respect to time. Current state-of-the-art approaches claim that slowdown of 100x relative to real system operation is acceptable for reasonable interactivity [5]. Utilizing lower fidelity models to achieve improved scalability and performance results in a degradation of overall simulation accuracy with respect to the real system. Simulation accuracy is also significantly affected by how rigorous timing models are enforced when transitioning between simulation states, represented as either static or dynamic timesteps [5]. It is also essential to understand whether a given simulation simulates deterministic or probabilistic behavior and whether there are mechanisms for emulating components interactions with external physical components or systems [23]. An incorrect specification of the interactions with real external components may result in incorrect operation causing the simulation to produce incorrect results.

2.2 Simulation of Cyber-Physical Systems

It is worth emphasizing that simulation is not solely a replacement for physical prototyping when validating CPS design and operation, and should instead be perceived as a means to enhance prototyping rapidity. This is made possible by the ability to simulate CPS operation under numerous operational conditions to evaluate early mock-ups at a reduced development time and cost compared to physical system implementation [39].

The simulation of CPSs requires modeling the system in terms of software execution and the full network topology of the infrastructure with respect to computation nodes, routers, switches, network links, and network data packets [8][9]. Furthermore, to enable accurate simulation the entire stack should also be captured, including the system stack replicating essential elements of the hardware and firmware behavior. Characteristics of the OSs and applications that they execute also need to be captured in some form in terms of resource utilization or processing times. Finally the network stack must also capture system operation such as session and transport protocols.

Due to the complexity of capturing the entire system model while maintaining reasonable performance, most approaches for simulating distributed systems focus on a single layer of the system stack [7]. Systems such as ns-3 focus on network simulation capturing detailed analytics for data packets, latencies, and throughput, however omit the application layers that utilize that network infrastructure [25]. Alternatively, existing tools that do not emphasize physical infrastructure instead focus on the software layers [26], leaving the responsibility of developing models that accurately reflect real world system operational characteristics and behavior to users.

2.3 Challenges

There is an increasing requirement for simulating complex large-scale CPSs that produce high levels of accuracy and performance while minimizing infrastructure requirements. While there are a number of state-of-the-art tools which can simulate generic [20][27][28] and domain specific CPSs environments [29] through HPC and multi-core environments, there are still limitations when simulating CPSs composed of thousands of nodes and millions of tasks executing for extended periods of operation time. While such simulations will eventually complete using current tools, they do so at significant execution time and infrastructure requirements. This becomes a significant problem for simulations that require several days or even weeks to complete execution for users who desire to acquire accurate results rapidly and system providers who wish to significantly reduce operational costs and infrastructure requirements.

It is also worth noting that with few exceptions, simulation configuration requires significant manual intervention and expertise with respect to both the simulated environment as well as the execution infrastructure used. As a result, simulating complex CPSs is not feasible for many users who lack the domain expertise to configure the underlying infrastructure for their simulation to run effectively.

3 Related Work

Lacage et al. [30] present YANS: a highly detailed tool for simulating networks. This tool captures the full network stack including TCP and UDP protocols and supports standards such as the IEEE 802.11a/e Network Interface model. The approach allows asynchronous parallel execution of simulated components. However, YANS does not support integration with real network components or applications. It is also limited to simulating only the network stack and network topology and does not support distribution of the simulation itself.

Nunez et al. [31] provides a unique approach to managing the scalability of simulating distributed systems by automatically parallelizing the simulation for execution. This is achieved by partitioning the network topology according to the number of communications between components. This approach is however targeted specifically at the simulation of HPC systems and assumes that the inter-component communication is known prior to execution which may not be the case for specific simulation configurations.

Garg et al. [24] propose NetworkCloudSim which extends the commonly used CloudSim [32] toolkit with detailed models of network components including links and switches. Although the generic approach allows for modeling of the entire distributed system it does not support the distribution of the actual simulation. Furthermore, it uses a serial simulation execution model leaving it highly susceptible to scalability issues.

Miller et al. [29] propose an advanced simulation tool for modeling distributed multicore systems: Graphite. Their

TABLE 1. COMPARISON OF SIMULATION APPROACHES

Feature	Environment	Target domain	Configuration & Deployment	Distributed	Slowdown	Synchronization	Task Types
SST+ gem5	HPC	HPC	Manually configured via API	Yes	N/A	Not managed	Real
Emulab	HPC	Generic	Configured using Tcl	Yes	Dependent on config & user time slice	Configuration dependent	Configuration dependent
PlanetLab	HPC	Generic	Manually configured via API	Yes	Dependent on config & user time slice	Configuration dependent	Configuration dependent
SIMCAN	Cluster	HPC & File I/O	Configured using scripts	Yes	N/A	N/K	Configuration dependent
Graphite	Cluster	Multi-core systems	Manual; specification of memory allocation & thread models	Yes	41x	Lax	Real
YANS	Desktop	Network only	Some default models; needs user knowledge of models & threading	No	Dependent on user thread model config	Event-based	Models
Network CloudSim	Desktop	Generic	Requires user knowledge of simulation system	No	Dependent on user config; scalability issues serial execution	Configuration dependent	Models
SEED	**Desktop or Cluster**	**Generic**	**Guided setup**	**Yes**	**Between 6x and 15x depending on simulation configuration**	**Event-based**	**Models or Real**

approach focuses on emulating the full distributed system, including applications and OSs. They find that they are able to simulate a system nearly 13x the size of their execution environment (1024 cores on 80 machines) with a slowdown of approximately 41x. However, the lack of enforcement for event ordering within the system results in a reduced level of accuracy and confidence in the simulation results. Additionally, the user is required to manually configure the memory distribution and thread allocation.

There are also alternative approaches for distributed system simulation that require the use of HPC environments for execution [20][27][28]. Such approaches all support simulation distribution and are capable of modeling different aspects of the full system and network stack. However, such approaches typically require tight coupling between the simulation and the underlying hardware, as well as manual configuration by both the system provider and the user domain expert. Furthermore, users are restricted by imposed scheduling restrictions of other system users, and face high expenditure if they desire to control their own infrastructure.

Our approach addresses many of the issues previously discussed when compared to related work as shown in Table 1. By providing an automated service-oriented process for configuration and deployment of a simulation, our system supports simulation deployment and execution across a heterogeneous distributed environment with no assumptions concerning underlying hardware specifications, as well as minimal user configuration. Importantly, our approach is also capable of managing event synchronization. The presented approach achieves this primarily through an automated partitioning mechanism that optimizes the simulation for distributed parallel execution. Finally our event based approach to manage clock jitter within the simulation allows us to support tasks which are either modeling realistic behavior or are actually executing tasks themselves.

4 SEED: Scalable Distributed Simulation

In this section we describe in detail core components and functionality of SEED; a service-oriented approach to facilitate large-scale CPS simulation.

For the purposes of this work we define a simulation of a distributed CPS as a *Topology* consisting of *Nodes* and *Links*. Simulated components are distinguished from physical components that form the execution infrastructure. The simulated components form the *Virtual Network* consisting of *Virtual Links*, *Virtual Nodes*, and Virtual Network Switches (*Switch Nodes*). The physical components are the machines on which simulation executes, known as *Physical Machines*.

The high level architecture of SEED is composed of several services that form components which ensemble together to perform CPS simulation as depicted in Figure 1. The advantage of such an approach is that the architecture is designed to be loosely-coupled, allowing (i) less dependency between components when altering specific component functionality, (ii) simulation components to reside across different machines within a network, reducing performance bottlenecks and increased fault-tolerance, and (iii) integration of additional components into the system.

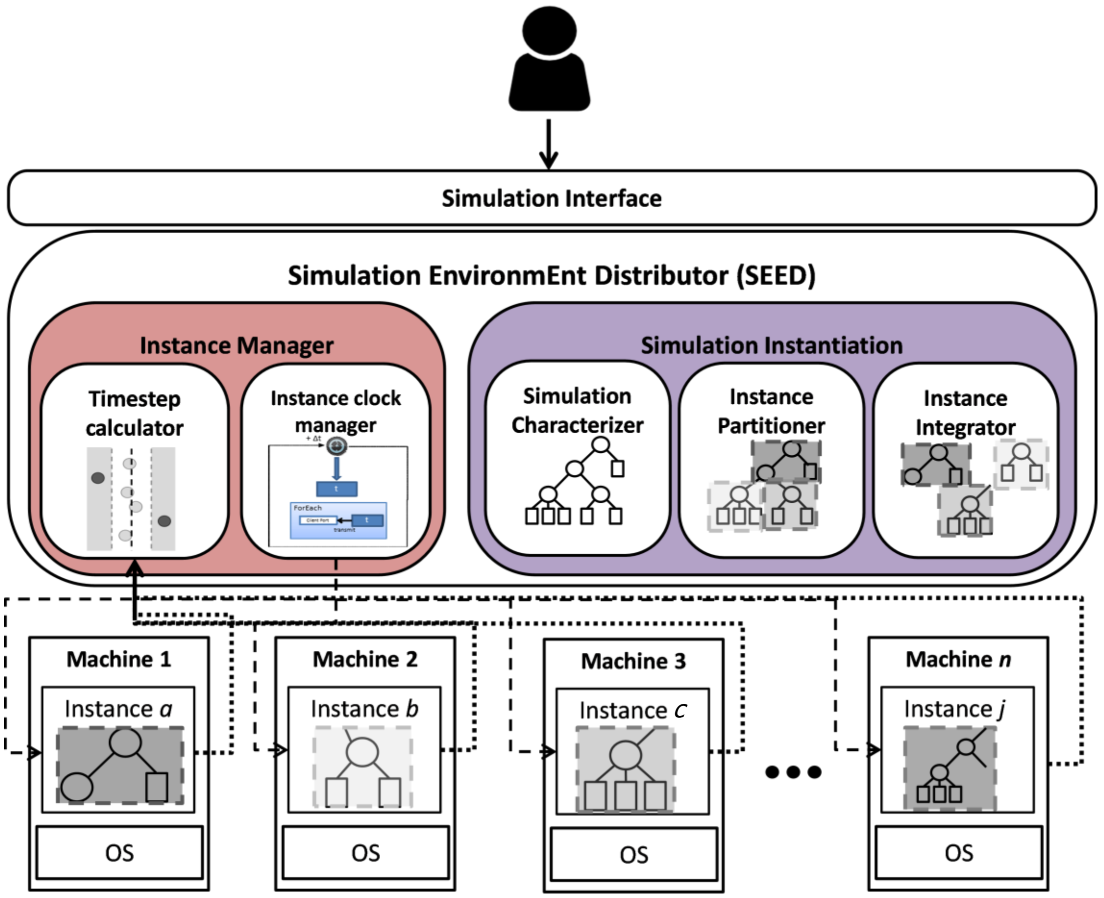


Figure 1. High level architecture of SEED.

These components are characterized as follows:

- **Simulation Instantiation**: automates the generation and characterization of the simulated network topology as well as the configuration and partitioning of a simulation across a loosely-coupled distributed physical infrastructure.
- **Clock Manager**: provides a scalable approach to maintain a highly synchronized simulation which is deployed across the distributed infrastructure. It provides an open framework for synchronization between local clocks for *Instances* to manage message ordering between *Virtual Nodes* that exist within different Partitions.
- **Instances**: logical unit of simulation computation comprised of various interacting components: *Nodes*, *Links*, and *Tasks*. An individual Instance is formed by a subset of the total *Virtual Network* topology and a specified set of tasks which executes on a partition created automatically by SEED, hosted on a specific physical machine. Each Instance has its own local clock which is managed externally by the *Clock Manager*.

SEED adheres to the following operational workflow performed by an ensemble of components, which consists of four key phases of performing distributed simulation: characterization, partitioning, deployment, and synchronization:

1. **Characterization** – Requirements for defining the operational characteristics of the CPS simulation is submitted by a user, consisting of configuration files, specifying the *Virtual Network* Topology and size, as well as the *Tasks* to be executed on it. Using the *Virtual Network* configuration file a *Virtual Network* model is generated.
2. **Partitioning** – The optimum number and size of partitions is computed using both the virtual and physical network topology and configuration files generated. The *Virtual Network* is then partitioned accordingly and configuration files are generated for each partition. A configuration file is then generated to inform the network communication between each simulation *Instance*.
3. **Deployment** – The generated partition configuration files are deployed to simulation instances across the distributed infrastructure according to the partitioning specification. Each instance begins execution using the locally available partition configuration, which includes *Virtual Network* specification and *Task* specifications.
4. **Synchronization** – All instances are centrally controlled and synchronized using the *Clock Manager* to maintain simulation accuracy.

Example Partition Specification

```
#Partition Specification#
$ Physical_IP_Address, Physical_Port, Clock_Port
192.168.1.17, 8888, 10001
192.168.1.23, 8888, 10002
192.168.1.32, 9999, 10001
```

Figure 2. Example of partitioning specification file generated.

Requires:

```
Nodes: {Compute_Nodes, Switches}
Links: {Link(EndPoint_1, EndPoint_2)}
Parts: {Part(Nodes, Links, Physical_IP,
Physical_Port)}
Partition (Nodes, Links, Parts)
1.  AssignSubnetWeightings(Nodes);
2.  Nodes.SortByIPAndWeight();
3.  partSize = Nodes.Count / Parts.Count;
4.  start = 0, length;
    //Partition Nodes
5.  For(∀k | k < Parts.Count)
6.      length = start + partSize;
7.      Parts[k] = Nodes[start → length];
8.      start = start + length;
9.  End For
    //Partition links generating proxies to
    traverse between partitions
10. Foreach(link ∈ Links)
11.     part1 = Parts[j] | link.EndPoint_1 ∈
12.     Parts[j].Nodes;
        If(link.EndPoint_2 ∉ part1.Nodes)
13.         part2 = Parts[m] | link.EndPoint_2 ∈
14.         Parts[m].Nodes;
            mlink1 = new Mediator
15.         Link(EndPoint_1, EndPoint_2,
16.            part2.Physical_IP,
            part2.Physical_Port);
17.         part1.AddLink(mlink1);
            mlink2 = new Mediator
            Link(EndPoint_1, EndPoint_2,
               part1.Physical_IP,
            part1.Physical_Port);
            part2.AddLink(mlink2);
18.     Else
19.         part1.AddLink(link);
20.     End if
21. End Foreach
    //Write XML
22. Foreach(part ∈ Parts)
23.     WriteXML(part)
24. End Foreach
```

Algorithm 1 - Partitioning of Virtual Network for Simulation.

The remainder of this section focuses on four core components within SEED as shown in Figure 1; **Simulation Characterizer, Instance Partitioner**, **Instance Integrator**, and **Clock Manager**.

4.1 Simulation Characterizer

The Simulation Characterizer is responsible for configuring and defining the *Virtual Network* topology of the simulation. This is achieved by user specific configuration files into a set of weighted subnets [35][36] which are used for partitioning within the Instance Partitioner.

The characterizer initially provides the user with the facility to auto-generate a network topology with a specified number of nodes which are computing servers or network switches as shown in Figure 2. The characterizer can optionally consume a model specifying the characteristics of nodes and network links such as CPU capacity, and latencies within the system serialized as an XML description file.

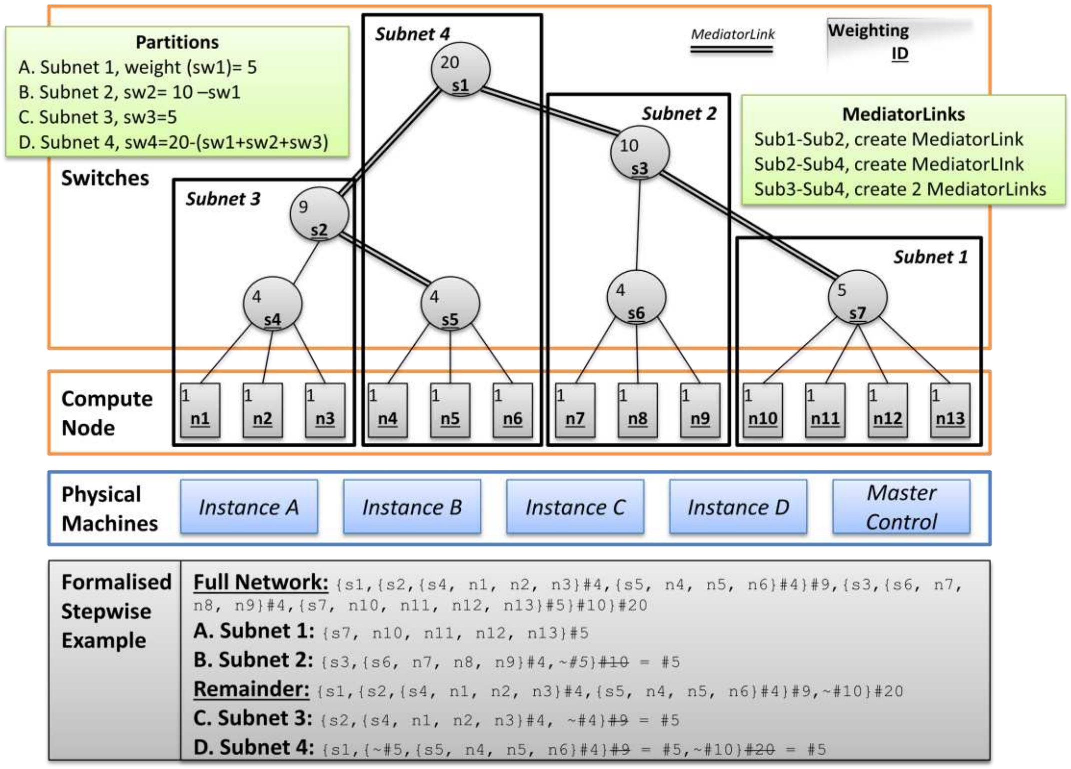

Partitions
A. Subnet 1, weight (sw1)= 5
B. Subnet 2, sw2= 10 –sw1
C. Subnet 3, sw3=5
D. Subnet 4, sw4=20-(sw1+sw2+sw3)
Switches
Subnet 4
Subnet 3
Subnet 2
Subnet 1
MediatorLink
Weighting
ID
MediatorLinks
Sub1-Sub2, create MediatorLink
Sub2-Sub4, create MediatorLInk
Sub3-Sub4, create 2 MediatorLinks
20 s1
10 s3
9 s2
4 s4
4 s5
4 s6
5 s7
Compute Node
1 n1
1 n2
1 n3
1 n4
1 n5
1 n6
1 n7
1 n8
1 n9
1 n10
1 n11
1 n12
1 n13
Physical Machines
Instance A
Instance B
Instance C
Instance D
Master Control
Formalised Stepwise Example
Full Network: {s1,{s2,{s4, n1, n2, n3}#4,{s5, n4, n5, n6}#4}#9,{s3,{s6, n7, n8, n9}#4,{s7, n10, n11, n12, n13}#5}#10}#20
A. Subnet 1: {s7, n10, n11, n12, n13}#5
B. Subnet 2: {s3,{s6, n7, n8, n9}#4,~#5}~~#10~~ = #5
Remainder: {s1,{s2,{s4, n1, n2, n3}#4,{s5, n4, n5, n6}#4}#9,~#10}#20
C. Subnet 3: {s2,{s4, n1, n2, n3}#4, ~#4}~~#9~~ = #5
D. Subnet 4: {s1,{~#5,{s5, n4, n5, n6}#4}~~#9~~ = #5,~#10}~~#20~~ = #5

Figure 3. Sample partitioning of a *Virtual Network* into 4 partitions according to weightings applied to each subnet.

Given the generated topology, the characterizer traverses the network tree using a depth-first approach assigning a weighting to each node corresponding to the cumulative of its child nodes and its own weight. A node's weight is computed from its normalized CPU speed. Subsequently, the nodes within the network are sorted by their IP address and then by the subnet weightings such that subnets of components are grouped within a vector. This is in order to allow the partitioner to locate subnets within the same partition. Upon completion, the Instance Partitioner automatically executes on the sorted and characterized *Virtual Network*.

4.2 Instance Partitioner

The instance partitioner is responsible for dividing a simulation's *Virtual Network* generated by the Simulation Characterizer into instances that contain a subset of the total simulation, comprised of unique simulation components such as compute nodes, switches and network links. This is achieved through the use of a partitioning specification and algorithm to generate the optimal number of instances depending on the simulation infrastructure.

It is assumed that interacting tasks are more likely to be deployed on compute nodes within the same subnet; as a number of tasks have architecture constraints [39] and Quality of Service (QoS) deadlines [40], such that interacting tasks are more likely to be deployed in computer nodes within closer proximity to one another. Therefore, in order to minimize the level of communication between partitions, larger subnets are prioritized to be placed within the same partition if possible. Therefore, the set of *Virtual Nodes* is firstly ordered by IP address, starting with the most significant (leftmost) byte, and then each switch is allocated a weighting for its subnet. *Nodes* are then reordered such that subnets with a weight closest to the partition size occur first. Then iterating over the parts using a sliding window (defined by start, and length variables), *Nodes* are allocated to a partition.

Given that *Virtual Nodes* may now exist in different partitions, it is necessary to adjust the specification of the *Virtual Network Links* to accommodate this feature. For this process the endpoints for each link are checked to determine whether they exist within the same partition. If they exist in different partitions, two *Mediator Links* are created with the physical IP addresses and ports of their opposite partitions. Once *Mediator Links* have been created they are added to the appropriate partition. The above process can be expressed as an algorithm as shown in Algorithm 1.

To give a practical example, Figure 3 depicts the partitioning of a simulated network consisting of 13 *Virtual Nodes* and 7 *Switch Nodes* with the corresponding 19 *Network Links*. With 4 physical machines available to host *Instances*, the algorithm to generate four partitions is as follows:

1. The *Virtual Network* is characterized using weights as shown in the top-left of each network *Node* as shown in Figure 3. Note that all *Nodes* have been assigned normalized weightings of '1' for example simplicity.
2. With an ideal partition size S (in this example, $S = 5$) the first 'X' components whose weights sum to S are allocated to subnet 1 whose *Nodes* are removed from the overall vector. This process repeats until all *Nodes* have been allocated to a subnet.

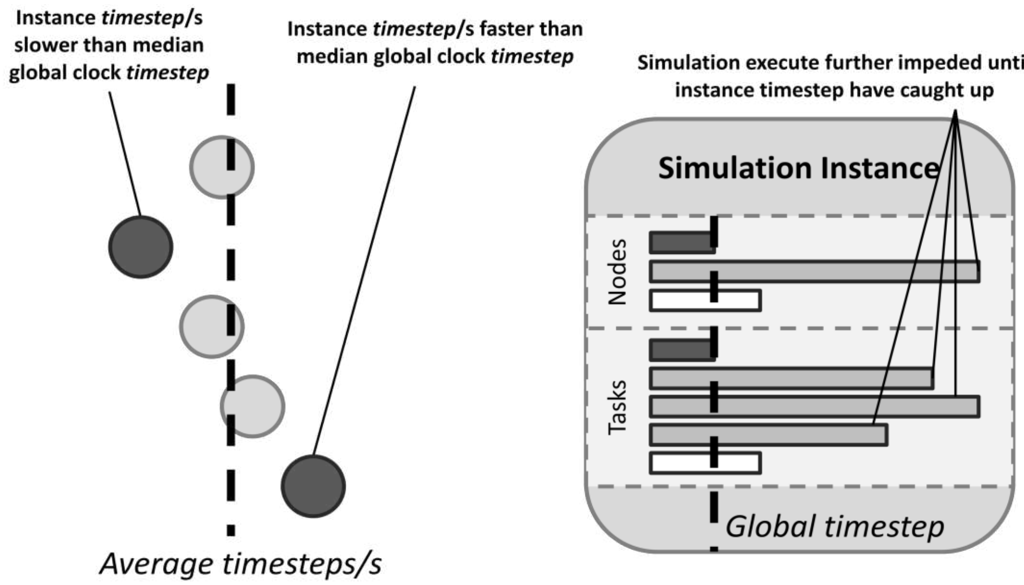

Instance *timestep*/s slower than median global clock *timestep*
Instance *timestep*/s faster than median global clock *timestep*
Simulation execute further impeded until instance timestep have caught up
Simulation Instance
Nodes
Tasks
Global timestep
Average timesteps/s

Figure 4. Depiction of simulation Instance jitter.

3. Any *Network Links* that connect *Nodes* which exist in different partitions are adapted to form *Mediator Links*. For the depicted configuration, four *Mediator Links* are required to facilitate inter-node communication across partitions.

4.3 Instance Integrator

Once the partition specifications have been generated, the Instance Integrator automates the process of deploying and instantiating the simulation partition into instances. Each generated network partition topology model is deployed to the respective physical machine, specified by the partition specification document, along with a task configuration file. The specification files define the architecture of the CPS to be simulated while the configuration files dictate the behavior and characteristics of *Tasks* with regards to their resource consumption (including CPU and memory utilization), as well as execution duration and number of occurrences within the simulation. Within each physical machine, a simulation *Instance* is started with the specified configuration files as well as a reference to the target output log file. The log of each *Instance* is generated in each physical machine and then automatically collated by the Instance Integrator after the simulations have completed allowing for analysis of simulation time against real world time, as well as detailed component analysis regarding aspects such as CPU and memory utilization or task status.

4.4 Clock Manager

Once instances have been instantiated it is necessary to maintain the accuracy of the simulation. This is achieved by proposing an approach that manages the *Instance* simulation local clocks with respect to the entire simulation's global clock. This aspect of the system is important when considering interactions between components existing in differing partitions, and their respective component local clocks times being as close as possible in order to guarantee simulation accuracy.

Additionally, due to users' different business requirements for simulation there is a need to support configuration of the simulation fidelity level. A major component of this fidelity is the clock frequency and synchronization approach. By increasing the frequency to 1000Hz, the event log of the simulation will remain accurate to 1ms. Alternatively, decreasing the frequency to 1Hz would significantly reduce the accuracy of the simulation with respect to communication between partitions, however, would produce an improvement in simulation performance. It is also worth noting that performance degradation can be reduced when only simulating the state change of *Nodes* and *Links* through task completion and interactions with other components within the system. As a result, the user is able to specify - depending on the analysis they wish to perform - the number of milliseconds within which the simulation *Instances* are synchronized. For this paper we used a frequency of 10Hz, and future work will discuss the performance tradeoff against the step size. An example of such behavior is depicted in Figure 4 which demonstrates jitter between various instances across the distributed simulation. Furthermore, in order to facilitate the simulation of simulated, emulated, or real tasks, the clock model must accommodate for varying degrees of control over the task execution. As a result, the clock manager allows additional jitter within any given instance (as also seen in Figure 4). The jitter is caused by tasks executing uncontrolled until

```
Global Clock:
∀ Instance ∈ Simulation.Parts |
    $Instance.Step \Leftrightarrow Instance.Time \leq Simulation.GlobalTime$
Instance.Step:
∀ node ∈ Instance.Nodes |
    $node.Execute \Leftrightarrow node.stateChanged.time \leq Instance.Time$,
∀ task ∈ node.ExecutingTasks |
    $task.Continue \Leftrightarrow task.Status \neq FINISHED \wedge$
    $task.time \leq Instance.Time$
∀ link ∈ Instance.Links |
    $link.Execute \Leftrightarrow link.stateChanged.time \leq Instance.Time$
```

Figure 5. Clock algorithm synchronizing *Instances*.

they either complete or interact with their host *Node* to request additional computational resources or else to communicate with other tasks within the simulation. This approach is summarized in the formalism presented in Figure 5 which summarizes the constraints used for state based optimization of the execution process.

5 Implementation

Due to the variation in infrastructure that is available to simulation users, it is essential to support heterogeneous systems ranging from individual desktop machines running Windows, MacOS or Linux based OSs to dedicated clusters running various Linux variants. The simulator was implemented using the C#.Net 4.0 language (and compiled using the Mono Framework for platform portability) which is deployed along with the simulator and all necessary dependencies. Our approach has been found to perform and operate on Windows 7, Debian, CentOS, and Mac OSX. This section describes the various components of the system along with their interactions.

5.1 Instance

Following from the topology defined in Section 4, the simulation is composed of several SEED instances each providing the facility to simulate a CPS partition consisting of a *Virtual Network* com prised of: *Nodes*, *Links*, and *Tasks* (simulated, emulated, or real) as shown in Figure 7.

Any interaction between nodes and links is managed entirely through an API hiding the detail of the implementation of any given component from the network topology. The most basic instance can be compiled from two XML and text-based configuration files, shown in Figure 6. This function is implemented using the WCF Web Service, a popular approach for deploying Web Services. For the purposes of strict synchronization, each component is allocated a local clock manager which all execute in parallel with reference to an Instance clock manager.

Example Network Specification

```
<?xml version="1.0" encoding="utf-16"?>
<Network Clock_Port="0">
  <Nodes>
    <Node ID="VN0_0.0.0.1" IP_Address="0.0.0.1" />
    <Node ID="VN1_0.0.0.2" IP_Address="0.0.0.2" />
    <SwitchNode ID="SW0_0.0.0.0" IP_Address="0.0.0.0" />
  </Nodes>
  <Links>
    <Link ID="LINK 1" NodeA="SW0.0.0.0" NodeB="VN0_0.0.0.1" />
    <Link ID="LINK 2" NodeA="SW0.0.0.0" NodeB="VN1_0.0.0.2" />
  </Links>
</Network>
```

Example Task Specification

```
#Task Specification#
$ Task duration (ms), Number of Tasks
500,  10000
2000, 200000
```

Figure 6. Example of *Network* and *Task* specification file.

Specifically, a *Network Node* can take the form of either a *Switch* or a *Compute Node* such as a server. The default implementations of the components can be used and configured using XML scripts specifying characteristics such as CPU and memory capacity. Alternative implementations can be introduced and integrated seamlessly as long as they adhere to the generic API which is used by all interacting components. A *Compute Node* is itself responsible for managing the execution of all *Tasks* deployed to it as well as managing the communication between *Tasks* existing either locally or across the simulated network. All nodes are synchronized using time stamped event logs. Their clock manager only requests execution in the event that their state has been changed and otherwise advances the clock appropriately.

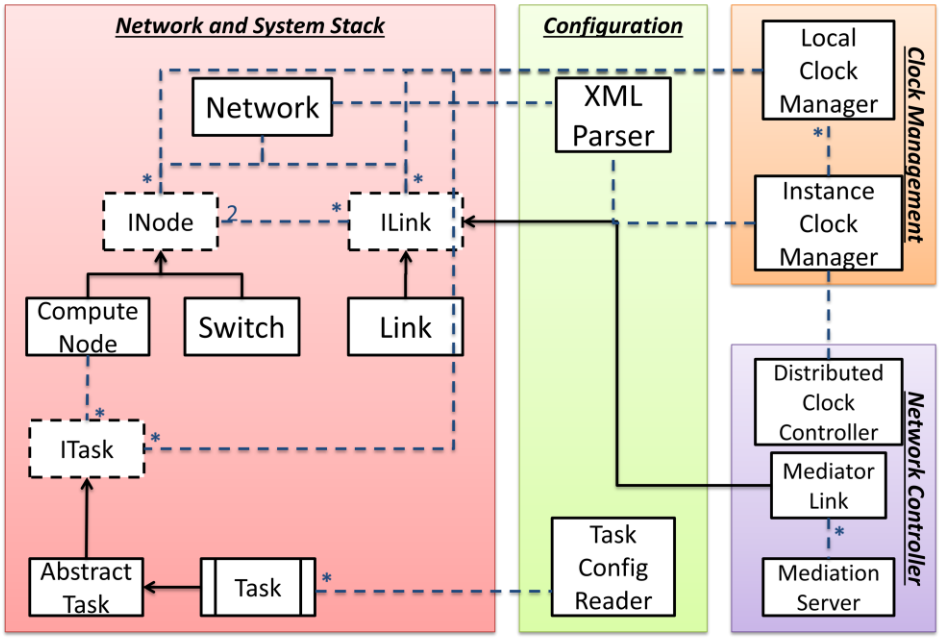


Figure 7. High level architecture of a simulator *Instance.*

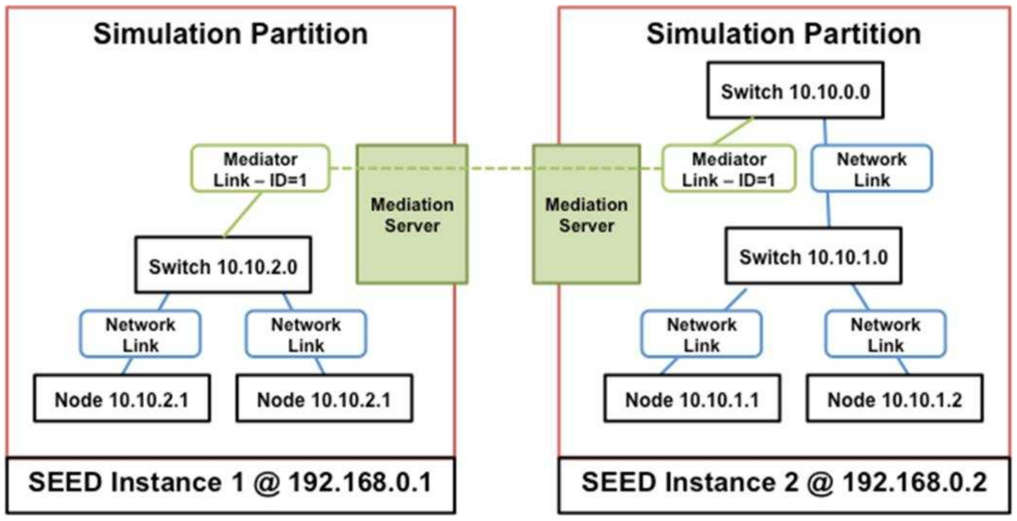


Figure 8. Architecture of *Network Mediator* operating between two simulation *Instances*.

Network *Links* operate using the same model as nodes but additionally provide models for latencies and throughput. Of particular interest is the composition of links whose endpoints may exist in different simulation instances. In this case a client-server approach is used which is described below. As with the Network *Nodes*, alternative implementations for *Network Links* can be used to model different types of connections such as IR or Bluetooth connections.

A *Task* itself can be configured to simulate the interaction behavior of a real task including the execution time, resource utilization, and interactions with other simulated components. Alternatively a *Task* may be a real process performing real computation in which case it is treated as a black box component which can only be controlled at the points of interaction with the simulator. Specifically, when a *Task* interacts with the host Compute Node its allocated clock manager will suspend the task's execution until the *Compute Node*'s clock has been synchronized.

5.2 Distributed Clock Manager

In order to maintain synchronization between clocks in physically distributed simulator network instances, a networked clock is used, from experience and experiments we have found it to be sufficient in maintaining synchronization across a loosely-coupled infrastructure.

Once a simulator instance is initialized, it registers itself with the networked *Master clock*. Following successful registration of all simulator instances, the *Master clock* increments its time T by Δt. A thread is initialized for each registered simulator instance and the current value of T is communicated to the simulator. Once all the simulator instances have received the current time, the current time becomes $T = T + \Delta t$. The management of Δt is handled entirely by the *Master clock* and not bound to real world time; therefore, if there is a delay in communicating to a simulator instance then the simulated passage of time can be temporarily slowed down if necessary.

5.3 Network Message Mediator

In order to permit physically distributed simulated network partitions to inter-communicate, a network message mediator as shown in Figure 8 has been developed to support and manage this task. A challenge to overcome was in the case of a distributed virtual simulator network, a *Link* object needs to contain references to two *INode* objects, both in different physical simulators. To support this capability, the *Link* class is extended to become a *Mediator Link*. This new type's constructor method accepts not just the source *INode*, however also the ID of the paired *Mediator Link* in the corresponding simulator network partition along with the physical IP address of the compute node hosting that simulated partition.

Once a data packet is sent to a *Mediator Link* this is passed on to a *Mediator Server* component, one of which is assigned to each simulator network partition. The *Mediator Server* receives a data packet and then marshals it for transport over the physical network to the *Mediator Server* associated with the target simulated partition. The target *Mediator Server* locates the paired *Mediator Link* in the target simulator Instance, unmarshals the data packets, and then places it in its send queue. It is important to note that the simulated link delay only occurs in one of the paired *Mediator Links* so that the delay is not artificially doubled.

6 EVALUATION

6.1 Experiment Setup

By using the implementation detailed in Section 5, experiments were conducted on a University cluster frequently used by students and researchers consisting of 40 x quad-core Intel machines @ 3.40GHz CPU running CentOS. Experiments were automated through the use of bash scripting for simulation partitioning, scheduling and instantiation.

The effectiveness of the approach is validated through varying a number of key parameters which are known to substantially affect simulation performance. This consists of (i) the size of the physical infrastructure the simulation is deployed on, and (ii) the amount of *Tasks* submitted and executing with the simulation. The simulation used in these experiments was a script which created a generic CPS

TABLE 2. STATISTICAL PROPERTIES OF SEED PERFORMANCE EVALUATION.

SEED nodes	Simulated Tasks	Mean 100ms simulation execution (s)	Med. 100ms simulation execution (s)	St. dev. 100ms simulation execution (s)	Mean instance instantiation (s)	Max. instance instantiation (s)	Timesteps/s	Slowdown
40	200,000	0.686	0.442	0.603	2.330	5.458	145.810	6.89 x
40	1,000,000	0.733	0.670	0.277	4.170	16.930	136.335	7.36 x
40	2,000,000	0.852	0.825	1.735	20.033	48.815	117.368	8.52 x
27	200,000	0.941	0.647	0.701	2.962	7.680	106.226	9.41 x
27	1,000,000	1.008	0.881	0.916	8.446	24.203	99.229	10.08 x
27	2,000,000	1.214	1.046	1.006	37.380	74.889	82.369	12.14 x
15	200,000	1.135	1.012	0.352	2.160	10.604	88.141	11.35 x
15	1,000,000	1.260	1.146	0.294	22.407	88.052	79.343	12.60 x
15	2,000,000	1.520	1.320	1.941	73.456	141.512	65.780	15.20 x
1	200,000	9.072	8.605	2.259	13.475	13.475	11.023	90.720 x
1	1,000,000	16.470	16.220	2.064	38.079	38.079	6.072	164.70 x
1	2,000,000	30.918	23.400	0.109	123.044	123.044	3.234	309.18 x

whose topology was shaped given by the number of nodes and links specified as script inputs. The simulation was executed on SEED using four different infrastructure sizes (1, 15, 27 and 40 physical machines), as well as varying amount of simulated *Tasks* (200,000, 1,000,000 and 2,000,000) forming a total of twelve experiment cases, with each respective case being executed 20 times. The size of the simulated CPS was configured to 2000 *Virtual Nodes* within each simulation in order to compare and contrast simulation performance for each experiment case. The *Simulation Instantiation* component and *Master Clock* were instantiated on separate nodes within the infrastructure.

6.2 Evaluation

Table 2 summarizes the performance evaluation of SEED for different experiment cases with varying simulation and infrastructure scale. It is observable that SEED allows for

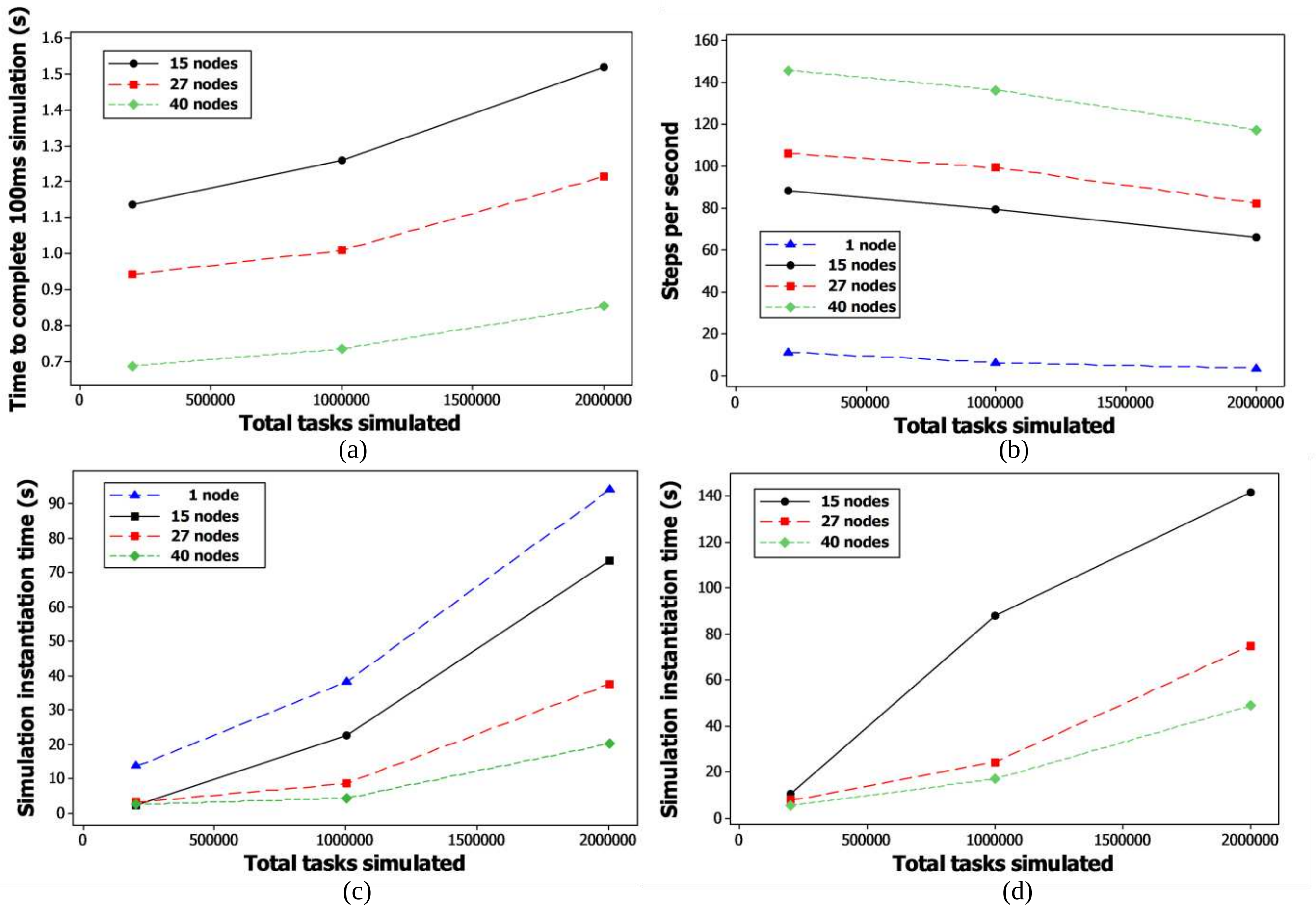


Figure 9. SEED performance evaluation (a) steps per second, (b) simulation execution time, (c) mean instance instantiation, (d) maximum instance instantiation

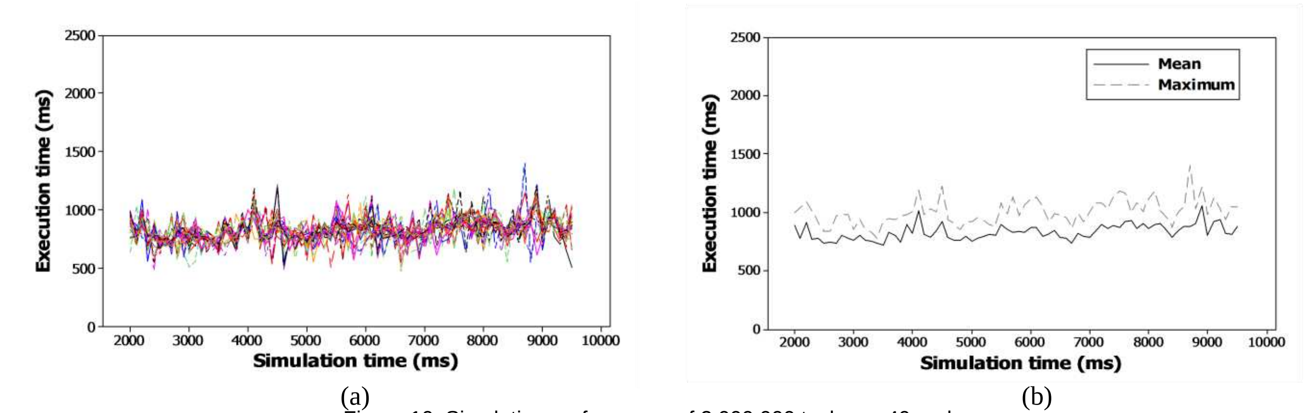


Figure 10. Simulation performance of 2,000,000 tasks on 40 nodes
(a) all instances, (b) mean and maximum.

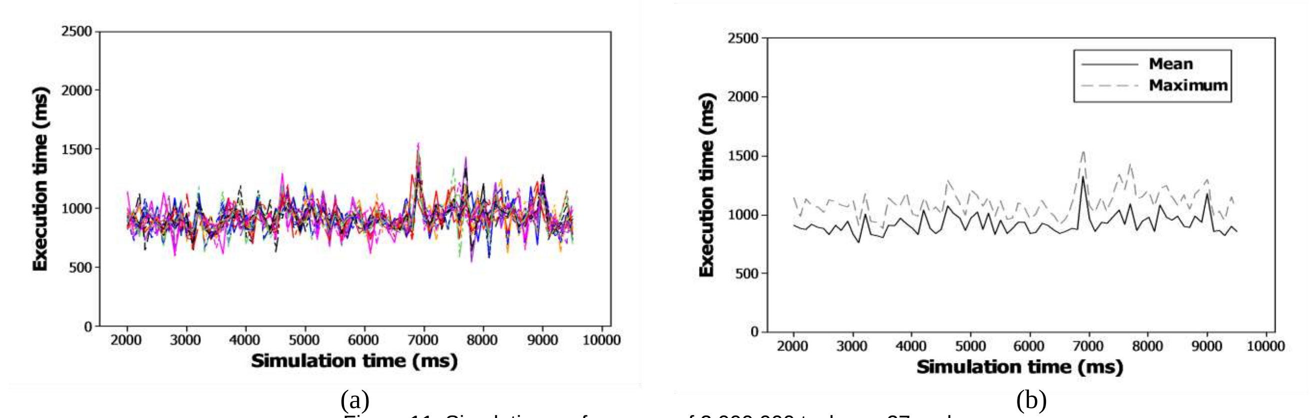


Figure 11. Simulation performance of 2,000,000 tasks on 27 nodes
(a) all instances, (b) mean and maximum.

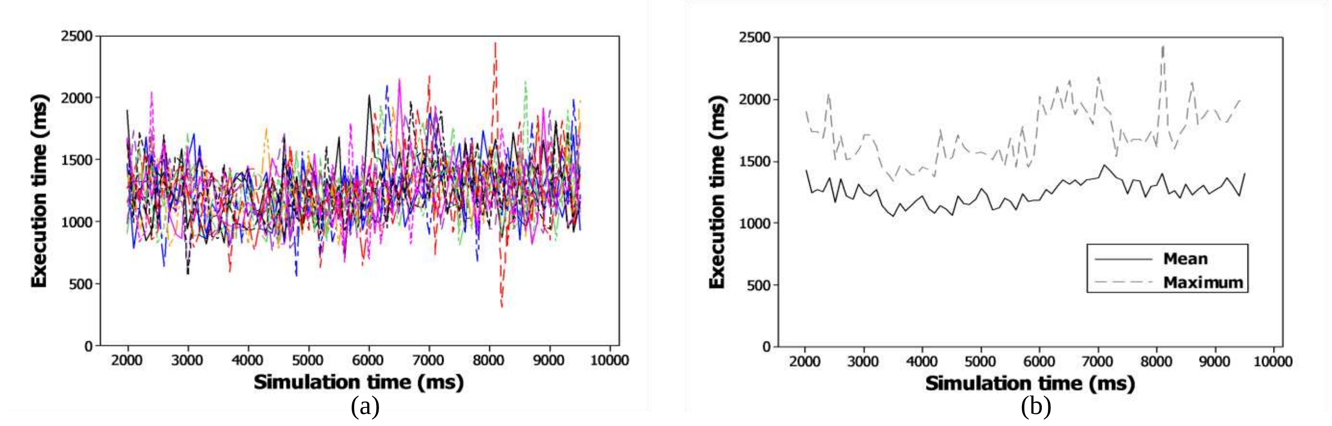


Figure 12. Simulation performance of 2,000,000 tasks on 15 nodes
(a) all instances, (b) mean and maximum.

CPS simulation comprised of 2000 *Virtual Nodes* and between 200,000 and 2,000,000 *Tasks* to be conducted 6.89x to 8.52x relative to real world time when deployed on 40 physical machines, respectively. This result is possible due to SEED enabling a simulation access to additional computing power across multiple nodes. Such an approach intuitively provides advantages over running simulations on a centralized OTS component, exemplified by the same simulation configuration executed on a single machine experiencing significant simulation slowdown at 309.18x relative to real world time.

It is observable that simulation performance increases when SEED has access to larger infrastructure, indicated by an average 45% simulation speed up between 15 and 40 physical machines for all conducted experiments. In relation, results indicate that performance decreases when simulations contain more components as shown in Figure 9(a), with similar levels of slowdown occurring across all infrastructure scale. Such behavior is exemplified within a single node, where increased number of components in a single instance results in significant slowdown between 90.720x – 309.18x, and follows similar time step degradation to larger infrastructure when increasing simulation complexity as shown in Figure 9(b).

The reason for the behavior described above is due to the composition of partitioned *Instances*. Specifically, *Instances* containing a large number of components experience slowdown caused by more frequent and larger volume of state change for executing *Tasks* (task progress time), *Links* (bandwidth) and *Virtual Nodes* (node capacity). As a result, increasing the number of partitioned *Instances* in the simulation results in reduced components per *Instance*, thus reducing simulation slowdown. This behavior is demonstrated in Figure 9(b), which shows overall performance degradation in simulation steps/s in the presence of increased components within the simulation.

Furthermore, we observe that simulation instantiation time is affected by the size of the physical infrastructure and simulation scale. As shown in Figure 9(c-d), while instantiation times of smaller simulations across all infrastructure configurations is very similar at 2.9 seconds for 200,000 *Tasks*, I*nstance* instantiation time becomes more significant in larger scale simulation reflected by instantiation time of 37.3 and 73.4 seconds for 2,000,000 simulated *Tasks* across 1 and 40 physical machines, respectively. The exception to this however is the single node, which requires a substantial amount of time to instantiate a single *Instance*, ranging between 13.4 – 94.04 seconds for different infrastructure size. Such a result is worth noting, as the feasibility of developing practical distributed simulation as a SaaS model must consider this effect in respect to desired QoS specified by users.

Ideal scalability of SEED would result in a 7.73x slowdown on 40 physical nodes when simulating 2 million tasks by extrapolating recorded slowdown on a single node. In reality an actual slowdown of 8.52x occurs due to additional network traffic between Instances and the *Clock Manager*. This result indicates that increasing system scale even further will result in network synchronization eventually becoming a debilitation to simulation execution.

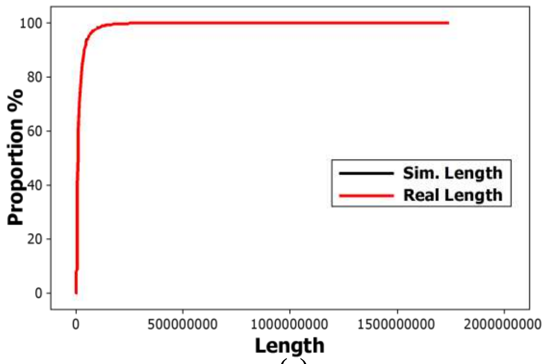
Due to the tight synchronization of instance execution in accordance to the *Master Clock*, the simulation must wait for the slowest instance to reach the global clock simulation time before advancing. We observe that increasing infrastructure size reduces the mean and standard deviation of simulation progression time as shown in Figure 10-12(b), which shows the mean and maximum *Instance* execution time per time step. It is observable that at smaller infrastructure size, deviation of simulation execution time increases dramatically in comparison to larger infrastructure size, reflected by a decreasing standard deviation in larger infrastructure as shown in Table 2 and depicted in Figure 10-12(a). While it is observable that this deviation results in a portion of instances to execute faster per time step, it also results in slower instance execution as shown in Figure 9(b). Such behavior results in slower simulation execution holistically due to the simulator being unable to progress until all *Instances* synchronize with the global *Master Clock* time step value. Such behavior does not appear to occur within a single server, as there is no network jitter within the simulation, expressed as a reduced deviation for completing 100ms simulation execution as demonstrated in Table 2.

From the experiments conducted, the actual partitioning of the simulation into *Instances* takes less than 0.5 seconds, while the scheduling of these Instances into the simulator infrastructure is dependent on the size of partitioned simulation and file transfer speed between physical nodes.

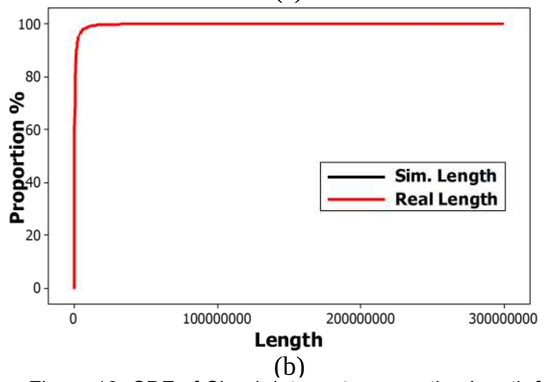
7 Practicality of SEED

In order to demonstrate SEED's effectiveness, we have implemented a simulation of a production CPS. To achieve this, we simulated operational characteristics from a real world production Cloud computing datacenter [34] from using prior statistically validated models developed within [33] and [38] for task cluster resource usage and length as well as server hardware characteristics, respectively. Using SEED, 100 *Virtual Nodes* and 1000 *Tasks* were executed 100 times on a single quad-core Intel machines @ 3.40GHz CPU running Windows 7 in order to validate accuracy of generated outputs in comparison to the empirical data.

Figure 13 and 14 demonstrate the accuracy of simulation outputs in terms of both simulation tasks components generated and their operational characteristics in terms of execution length, respectively. It can be observed in Figure 13(a-b) that the Cumulative Distribution Function (CDF) for execution length for Task type 1 and 3 is statistically similar, represented by a Lognormal distribution with a location



(a)



(b)

Figure 13. CDF of Cloud datacenter execution length for (a) Task type 1, (b) Task type 3.

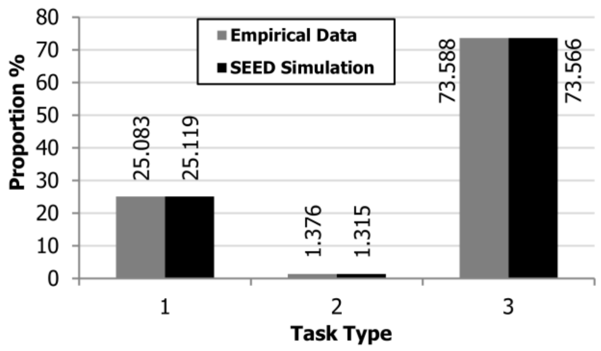


Figure 14. Comparison of task proportions in a Cloud datacenter.

and scale value difference less than 0.01. Furthermore, Figure 14 demonstrates that simulated outputs from SEED are statistically similar to the empirical data, indicated by 0.1% difference between empirical and simulated data.

In terms of performance, it was possible to complete simulation execution within 24 seconds. This is primarily due to the small component size and low fidelity level within an *Instance* (as discussed in Section 2.1), however for the purposes of studying scheduling practices and their effect on server capacity and utilization, SEED provides a means to effectively run substantial amounts of simulated CPS operation in a short time frame. While additional complexity, components and their respective interactions introduced into the simulator will result in slow down, this effect can be mitigated due to SEEDs ability to scale to multiple physical nodes as demonstrated in Section 6.2.

8 CONCLUSION

In this paper we have presented SEED: a novel service-oriented approach for effective large-scale Cyber-Physical System simulation. The process and architecture used for automated partitioning, instantiation and execution of CPS simulation over a distributed computing infrastructure is described in detail. Unlike other approaches, SEED is capable of enforcing event-based synchronous simulation across loosely-coupled OTS distributed environments with no assumptions concerning underlying hardware, as well as minimal user interaction through the use of XML-based protocols. The approach has been empirically demonstrated to effectively simulate CPS operational behavior through experiments conducted at different simulation sizes and infrastructure scale, as well as capable of simulating operational behavior of a real production CPS. A number of conclusions can be made:

SEED provides an effective means to achieve distributed simulation. Experiments demonstrate that SEED is capable of simulating 2,000 nodes executing 2,000,000 tasks, slowdown between 6.4x and 15x relative to real world time, and best case jitter of 0.277s per simulation time step. Such a result represents substantial simulation speed-up and interactivity in comparison to current approaches, and allows the ability to simulate large-scale CPS operation in order to study workload characteristics, resource management and scheduling of large-scale CPSs, exemplified from the simulation of a production Cloud datacenter.

Important trade-offs must be considered when designing and deploying distributed simulators As demonstrated from experiments conducted, simulation performance is directly correlated to simulation size and computing infrastructure scale. While larger infrastructure scale results in increased simulation speed, it also results in higher expenditure and operational costs. On the other hand, while using smaller infrastructure results in reduced expenditure, it causes simulations to require longer periods of time to complete execution resulting in increased energy costs, as well as detrimental performance due to increased computation complexity per instance and jitter in simulation synchronization. This suggests that it should be possible to derive an optimal balance between these two options, with respect to a fulfilling user QoS demands if the simulator is provided as SaaS (or another type of system design and functional goal).

The proposed approach is a valuable step in providing distributed simulation of large-scale CPS environments as a service, and abstracting the user away from the underlying hardware to run their simulation. Future work will include studying and implementing simulation heuristics prior to execution to optimize simulation performance, running different types of simulations using emulated and real system components, as well as deploying SEED in even larger infrastructure. Furthermore, as the approach has been demonstrated to accurately simulate a real production CPS environment, we are currently extending SEED in order to evaluate a number of proposed resource management policies under different operational scenarios driven by dynamic user and application behavior.

References

[1] W. Mueller, M. Becker, A. Elfeky, A. DiPasquale, "Virtual prototyping of Cyber-Physical Systems," Design Automation Conference (ASP-DAC), pp.219-226, 2012.
[2] A. M. Law, W.D.Kelton, "Simulation, Modelling and Analysis: Third Edition", McGraw-Hill Series in Industrial Engineering and Management Science, 2000.
[3] Huang et al., "Development of an Automated Testing System for Vehicle Infotainment System.", Intl. Journal of Advanced Manufacturing Technology, pp. 233-246, 2010.
[4] S.H. Choi, A.M.M. Chan, "A Virtual Prototyping System for Rapid Product Development", Computer-Aided Design, Vol. 36, Issue 5, pp. 401-412, 2004.
[5] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," Softw. Pract. Exp., vol. 34, no. 7, pp. 653–673, 2004.
[6] E. Weingartner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators," IEEE International Conference on Communications, pp. 1–5, 2009.
[7] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley, "Large-scale network simulation: how big? how fast?," in 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS), pp. 116–123, 2003.
[8] M. Rosenblum, S. A. Herrod, E. Witchel, A. Gupta, "Complete computer system simulation: the SimOS approach," IEEE Parallel Distrib. Technol. Syst. Appl., vol. 3, no. 4, pp. 34–43, 1995.
[9] R. Malhotra, "Study and Comparison of Various Cloud Simulators Available in the Cloud Computing," SIJ Trans. Comput. Sci. Eng. its Applications (CESA), vol. 1, no. 3, 2013.
[10] S. Mohapatra, P. Kanungo, "Performance analysis of AODV, DSR, OLSR and DSDV Routing Protocols using NS2 Simulator," Procedia Eng., vol. 30, pp. 69–76, 2012.
[11] Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating data Grids: an extension to GridSim," Concurr. Comput. Pract. Exp., vol. 20, no. 13, pp. 1591–1609, 2008.
[12] M. Turner, D. Budgen, P. Brereton, "Turning Software into a Service," in IEEE Computer journal, vol. 36, pp.38-44, 2003.
[13] M. P. Papazoglou, W.-J. van den Heuvel, "Service-Oriented Computing: State of the Art and Open Research Challenges," Computer (Long. Beach. Calif)., vol. 40, no. 11, pp. 38–45, 2007.
[14] G. Christine and G. Emilie, "Modelling of distributed system in one single simulation model: a way to study communications within distributed systems," IEEE Conference on Emerging Technologies and Factory Automation, 2, vol. 1, pp. 697–703, 2005.
[15] J. Memon and W. U. Rehman, "Simulation on Single Server & Distributed Environment (It's Comparison & Issues)," World J. Eng. Technol., vol. 1, no. 2, pp. 23–25, 2013.
[16] W. T. Tsai and R. Paul, "Modeling and Simulation in Service-Oriented Software Development," Simulation, vol. 83, no. 1, pp. 7–32, 2007.

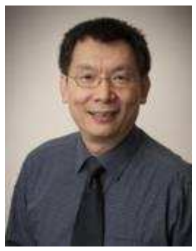
[17] R. Jain, "The art of computer systems performance analysis" John Wiley and Sons Inc., pp. 391–504, 2008.
[18] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job Scheduling for Multi-User MapReduce Clusters," Technical Report No. UCB/EEC2S-2009-55, 2009.
[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab," ACM SIGCOMM Comput. Commun. Rev., vol. 33, no. 3, pp. 3-12, 2003.
[20] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," ACM SIGOPS Oper. Syst. Rev., vol. 36, pp. 255-270, 2002.
[21] P.S. Magnusson, "Simics: A full system simulation platform. Computer", IEEE Computer Journal, pp. 50–58, 2002.
[22] S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, "ProtoFlex," ACM Trans. Reconfigurable Technol. Syst., vol. 2, no. 2, pp. 1–32, 2009.
[23] H. Kurahata, T. Fuji, T. Miyamoto, and S. Kumagai, "A UML Simulator for Behavioral Validation of Systems Based on SOA," Int. Conf. Next Gener. Web Serv. Pract., pp. 3–10, 2006.
[24] S. K. Garg, R. Buyya, "NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations," 2011 Fourth IEEE Int. Conf. Util. Cloud Comput., pp. 105–113, Dec. 2011.
[25] E. Weingaetner, H. Lehn, and K. Wehrle, "A performance comparison of recent network simulators," IEEE Intl. Conf. on Communications, pp. 1-5, 2009.
[26] A. Bashar, "Modeling and Simulation Frameworks for Cloud Computing Environment: A Critical Evaluation," in International Conference on Cloud Computing and Services Science, pp. 1–6, 2014.
[27] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in Proc. On USENIX Symp. On Operating Systems Design and Implementation, pp. 351–366, 2006.
[28] M. Hsieh, J. Meng, M. Levenhagen, K. Pedretti, A. Coskun, and A. Rodrigues, "SST + gem5 = A Scalable Simulation Infrastructure for High Performance Computing," Proc. Fifth Int. Conf. Simul. Tools Tech., pp. 196-201, 2012.
[29] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, pp. 1–12, 2010.
[30] M. Lacage and T. R. Henderson, "Yet another network simulator," Proceeding from 2006 Work. ns-2 IP Netw. simulator - WNS2 '06, Article. 12, 2006.
[31] A. Nunez, J. Ferna, and J. Carretero, "New Contributions for Simulating Large Distributed Systems," 2010 IEEE/ACM 14th Int. Symp. Distrib. Simul. Real world time Appl., pp. 227–230, 2010.
[32] R. N. Calheiros, R. Ranjan, A. Beloglazov, A. F. De Rose, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software Practice & Experience, vol. 41, pp. 23–50, 2011.
[33] I.S. Moreno, P. Garraghan, P. Townend, J. Xu, "Analysis, Modeling and Simulation of Workload Patterns in a Large-Scale Utility Cloud," IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp.208-221, 2014.
[34] C. Reiss, J. Wilkes, J. Hellerstein, "Google Cluster-Usage Traces: Format Schema," Google Inc., Mountain View, CA, USA, White Paper, 2011.
[35] J. C. Mogul, "Internet subnets.", in Internet Engineering Task Force RFC 917, 1984, http://tools.ietf.org/html/rfc917.
[36] A. Barrat, et al. "The architecture of complex weighted networks." Proceedings of the National Academy of Sciences of the United States of America pp. 3747-3752, 2004.
[37] R.M Fujimoto, "Distributed simulation systems," Simulation Conference, 2003. Proceedings of the 2003 Winter , vol.1, no., pp.124-134, 2003.
[38] P. Garraghan, I.S. Moreno, P. Townend, J. Xu, "An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment," IEEE Transactions on Emerging Topics in Computing, vol.2, no.2, pp.166-180, 2014.
[39] C. Reiss , A. Tumanov , G. R. Ganger , R. H. Katz , M. A. Kozuch, "Heterogeneity and dynamicity of Clouds at scale: Google trace analysis", Proceedings of the Third ACM Symposium on Cloud Computing, pp.1-13, 2012.
[40] I.S. Moreno, J Xu, "Neural Network-Based Overallocation for Improved Energy-Efficiency in Real world time Cloud Environments," IEEE 15th Intl. Symposium on Object/Component/Service-Oriented Real-time Distributed Computing (ISORC), pp.119-126, 2012.

Peter Garraghan is a Research Fellow in the School of Computing, University of Leeds and a visiting researcher at Beihang University, China. He has industrial experience building large-scale systems and his research interests include distributed systems, Cloud computing, large-scale simulation, data analytics and energy-efficient computing.

David McKee is a PhD student in the School of Computing, University of Leeds, UK. He received the M.Eng. degree in Computer Systems and Software Engineering from the University of York, UK. His research interests include real world time service-orientation as well as large-scale distributed system simulation.

Xue Ouyang is a PhD student in the School of Computing, University of Leeds. She received her B.Eng. degree in Network Engineering and M.Eng. degree in Software Engineering from National University of Defense Technology, China. Her primary research interest lies in improving service performance within distributed systems.

David Webster is a Research Fellow working in the School of Computing at the University of Leeds. He has published over 20 peer-reviewed papers in the fields of distributed systems since 2004. David's primary research focus is in the area of service-orientation in distributed systems and mechanisms to handle their evolvable nature.


Jie Xu is Chair of Computing at the University of Leeds and Director of the UK EPSRC WRG e-Science Centre. He has industrial experience in building large-scale networked systems and has worked in the field of dependable distributed computing for over 30 years. He is a Steering/Executive Committee member of IEEE SRDS, ISORC, HASE, SOSE, etc. and a co-founder of the IEEE Conference on Cloud Engineering (IC2E). He has led or co-led many research projects to the value of over $30M, and published over 300 research papers.