Towards distributed heterogeneous simulation using internet of things

Haseeb, Muhammad; Malik, Asad Waqar; Rahman, Anis ur; Hamayun, Mian Muhammad

DOI:
10.1109/JIOT.2019.2939361

Towards Distributed Heterogeneous Simulation using Internet of Things

Muhammad Haseeb, Asad Waqar Malik, Anis U. Rahman, and Mian M. Hamayun

***Abstract*—Parallel discrete event simulation frameworks have been widely used to analyze the performance of traditional applications under different scenarios. The existing frameworks are designed to work on a cluster and cloud-based computing environments. With the current advances in the internet of things, there is a strong need to revamp such traditional frameworks and make use of the smart connected-devices as an underlying infrastructure to perform simulations. In this study, we propose a new simulation framework, which has been specifically designed to work with diverse heterogeneous devices. The framework allows these heterogeneous mobile devices to participate in a distributed simulation while managing network latency, using device profiles that are maintained by the simulation framework. Moreover, in the proposed framework, random and context-aware simulation task distributions have been explored to manage the devices' sporadic connectivity. Evaluation results using the well-known PHOLD benchmark demonstrate a gain in the overall efficiency of the proposed simulation system.**

***Index Terms*—Discrete event simulation, time warp, internet of things, distributed simulation, master-worker paradigm, parallel computation.**

I. Introduction

Parallel Discrete Event Simulation (PDES) has evolved over the years and is still an active research area due to its significant application in many fields. It is mainly used in fields including automation, auto manufacturing, military training, civil aviation, road traffic monitoring, bio-informatics and other computer-controlled systems [1]. With different implementations of distributed simulations, the objectives and outcomes vary with task requirements of the simulated systems. These objectives include the prediction of possible outcomes, validation of the designed models or to discover the effects of various changes in environments [2]. The existing simulation techniques can be broadly classified as a time-stepped and event-driven approaches. In the time-stepped models, the simulation time advances linearly, whereas, in the event-driven approaches, simulation time advances from one event to the next event. The former approach lacks adaptability, and hence, the latter is widely used due to its adaptability and application in various domains. Furthermore, PDES over a distributed environment requires synchronization models to produce correct results. Therefore, PDES can be further classified into conservative and optimistic approaches. In case of conservative approaches, the causality constraints are always enforced by disseminating control messages across simulation components. In contrast, being optimistic permits violation of local causality constraint with support for rollbacks, in order to undo out-of-order executions [2].

With advancements in system architecture and availability of fast processing units, complex PDES models have been adopted to simulate realistic scenarios such as traffic modeling and weather forecasting. But existing simulation frameworks are designed to work in a cluster-based environment with fixed number of nodes and stable network connectivity throughout the simulation run, that is the entire simulation is performed in a controlled environment. However, with the inception of cloud computing paradigm, researchers have started using cloud infrastructure for running such simulations, presenting new challenges for traditional simulation frameworks. In contrast to the cluster environment, it is difficult to provide a controlled environment for simulation execution in a cloud setting. Moreover, the cloud offers a multi-tenant environment with potentially other applications executing on the same node, on the same rack or even using the same network path and can adversely affect the overall efficiency of the simulation. Therefore, traditional frameworks need modifications to perform well under a shared cloud environment [3].

Similarly, the adoption of IoT has opened up new research opportunities for PDES community. With connected devices termed as smart devices possess computing, networking and storage capabilities. Even though the resources are limited as compared to traditional computer systems but they are capable of running a manageable simulation model. Moreover, these heterogeneous devices provide a low latency infrastructure and access to contextual information for simulation runs. The contextual information about the device's compute capability and network connectivity determines its likelihood of selection for task execution. On the contrary, the traditional simulation frameworks fail to be ported on to such resource-constrained devices. Key issues include response delays, sporadic connectivity, variable processing power, and low device memory.

The use of conservative simulation models over heterogeneous devices increase the simulation time while decreasing the overall efficiency of the system. In this situation, the adoption of traditional optimistic approaches is a better option as it exploits parallelism. However, frequent rollbacks can drain device energy quickly. Moreover, sporadic connectivity in mobile networks can adversely impact simulation results. Furthermore, both types of simulation approaches *i.e.* conservative and optimistic, lack any kind of fault tolerance mechanisms. This makes the simulation environment even more

M. Haseeb, A. W. Malik, A. U. Rahman and M. M. Hamayun: National University of Sciences and Technology (NUST), Islamabad, Pakistan. A. W. Malik and A. U. Rahman: Department of Information Systems, Faculty of Computer Science & Information Technology, University of Malaya, Malaysia. Mian M. Hamayun: University of Birmingham, Dubai, United Arab Emirates. Corresponding author: Asad Waqar Malik.

Manuscript received April 19, 2019; revised August -, -.

TABLE I
COMMONLY USED PDES FRAMEWORKS

PDES Framework	Language	Simulation Model	Context-aware Task Distribution	Supports Dynamic Topology Change	Support for Low Energy Devices
GloMoSim [4]	C-based PARSEC	Hybrid	×	×	×
DaSSF [5]	Java/C++	Conservative	×	×	×
ARTIS GAIA+ [6]	C/MPI	Conservative/Optimistic	×	×	×
LUNES [7]	C	Agent-based	×	×	×
ScipySim [8]	Python	Conservative	×	×	×
ROOT-Sim [9]	C	Optimistic	×	×	×
ErlangTW [10]	Erlang	Optimistic	×	×	×
Spades [11]	Java	Agent-based	×	×	×
GO-Warp [12]	GO	Optimistic	×	×	×

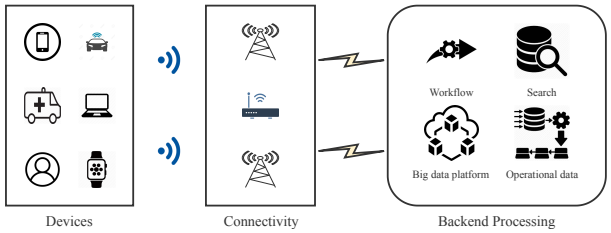

Fig. 1. Typical IoT model where devices are part of a network. The devices share information with a central unit for further data processing.

complex with heterogeneous devices on different networks and participating in the distributed simulation. Thus, increasing the total delay and decreasing the simulation reliability.

Contributions – In this paper, we propose a master-worker framework to support PDES over smart devices. The framework manages heterogeneous resources and provides fault tolerance. The master nodes control all devices/nodes participating in the simulation. The framework allows joining of new devices during the simulation, as well as replacement of existing devices with the newly-joined ones. The key contributions of the proposed framework are listed below:

(i) Proposes a master-worker paradigm with master nodes initiating and tracking simulation on available connected devices. Thus, it supports execution of PDES over heterogeneous devices including low energy devices.
(ii) Manages churn behavior of mobile devices dynamically in order to smooth-out execution of the entire simulation and to provide a better simulation environment based on heterogeneous devices.
(iii) Dynamic management of device connectivity and task distribution resulting in improved overall efficiency, reduced number of rollbacks as compared to random distribution and lesser power consumption. Thus, the improved efficiency can help in better utilization of IoT/ battery-operated devices.

Organization – The rest of the paper is organized as follows. Section II covers recent research contributions in the area and presents reviews on existing frameworks. In Section III system model is presented. The proposed framework is presented in Section IV. Section V covers the motivation behind this study. Simulation parameters and evaluation setup are discussed in Section VI. Finally, the discussion and conclusions are presented in Sections VII and IX.

II. RELATED WORK

Over the last couple of decades, many simulation frameworks have been proposed to support conservative and optimistic approaches for different testbeds. Fujimoto et al. [13] explains PDES concepts and its real-life applications to give a better understanding to readers and researchers. Moreover, the selection of synchronization techniques with respect to such applications have been discussed in [14]. Time Warp (TW) algorithm is the most commonly used optimistic simulation protocol in various applications [15]. Other variants are discussed in [16] and parameterized TW algorithm [17].

Table I summarizes the commonly used PDES frameworks, in addition to the TW algorithm discussed above. However, these frameworks cannot work over IoT devices, as most of the frameworks are designed to work in controlled environments *i.e.* these frameworks cannot handle/tolerate node/device or packet failures. Therefore, in order to utilize the computing power of nearby IoT devices, we need a new framework that can provide fault tolerance features to PDES. Despite widespread adaption of IoTs in different research domains, the use of heterogeneous devices for PDES has been explored to a very limited extent. Over the last decade, technology has made significant progress in terms of exploiting these smart devices. In the following sub-sections, we cover these PDES contributions including algorithms and frameworks.

PDES Frameworks for Internet of Things – Gabriel et al. [18] use hybrid simulation models for large complex scenarios. The models are capable of exploiting network diversity, which is the case for IoT devices. Fundamentally, the model decomposes a complex simulation scenario into small individual blocks, each working in a synchronous fashion. Furthermore, the solution supports three different simulation models: agent-based, discrete event and script-language; thus, providing a suitable simulation environment for IoT devices. Similarly, Giancarlo et al. [19] propose a hybrid approach exploiting the benefits of agent-based computing. The approach is integrated with a network simulator to investigate the

communication challenges when using IoT devices. The aforementioned contributions use conservative approaches requiring synchronizations before every simulation time advancement. The extensive use of synchronizations limits the opportunities for parallelism in such models. Gluhak et al. [20] reviews existing frameworks with their limitations and concluded that the existing test-bed simulators are ineffective due to the high diversity of IoT devices and associated challenges. Brambilla et al. [21] propose a framework based on DEUS, a discrete event simulator to study large-scale IoT networks. The simulator is developed in Java and benchmarked in an urban IoT environment. The main objective of this work is scalability. Furthermore, the simulator is based on a conservative approach with no mention of time advancement mechanism. Similarly, traditional simulation protocols are tested over standalone battery-operated devices in terms of energy memory and execution of time [22]. Further, to utilize the computing power of IoT devices, Artificial Neural Networks (ANNs) are used for simultaneous data transfer in order to reduce latency and improve network lifetime [23]. To summarize, a comparison of IoT simulation frameworks is presented in Table II.

PDES Frameworks for Cloud-based Simulation Models – On-demand processing power and pay-as-you-go services have changed many aspects of traditional simulation and its related algorithms. Over time, parallel and discrete event simulators have adapted different paradigms to exploit the power of these services. D'Angelo et al. [33] present an approach to shift from traditional parallel computing platforms to cloud computing services by modifying existing techniques. Fujimoto et al. have presented the overall advancements in PDES over the years and discussed the future directions as well as challenges in [34]. Serrano-Iglesias et al. [35] propose DNSE3; a web-service oriented cloud-based network simulator taking benefit of the elasticity and scalability features. The main objective is to run complex large-scale simulations on the pay-per-use cost model. The proposed simulator takes a range of parameters, performs the simulation and on completion, returns results to the end-user. Malik et al. [3] propose a Time-Warp Straggler Message Identification Protocol (TW-SMIP) to support optimistic simulation models in cloud environments. The model dynamically computes barrier points based on the straggler messages. The results show an improved efficiency as compared to the traditional TW algorithm in cloud environments. Similarly, PDES framework for distributed shared memory is presented in [36]. The framework introducing event- and cross-state models with event handlers accessing memory of any process to change the event pointers. However, the proposed solution is tested in a private cloud environment.

Cloud is inherently a multi-tenant environment and the execution of other applications can impact the overall simulation performance. Limited works have focused on performance improvement using continuous system monitoring techniques. For example Peng et al. [37] have proposed a neural network-based system that transforms tasks into an abstract models. Such models can be used to efficiently manage resource allocations in a multi-tenant environment such as cloud.

PDES Framework for High-Level Architecture – High-level architecture (HLA) is an IEEE Std 1516-2000 designed to provide usability and interoperability among existing simulation frameworks. HLA implementation provides integration of existing simulators to provides services for objects, data and federation management [38]. The objective is to build a complex multi-scale simulation. Wang et al. [39] present a rollback back mechanism and its adoption in HLA. In a wide-area network, communication delays affect the overall simulation performance. However, in such networks, the communication costs are usually high and it becomes problematic to synchronize nodes using an optimistic approach. Such network delays contribute to an increase in total rollbacks. Ziganurova et al. [40] discuss general rollback issues and propose a technique to reduce delays using the shortest path approach. However, in most of the cases, it is unrealistic to control the underlying network. Distributed simulation development using HLA is a complex task requiring a sound knowledge of the HLA standard. To handle this challenge, Falcone et al. [41] have proposed HLA Development Kit Framework (DKF) to facilitate the challenging development process.

Literature Review Summary – Parallel and distributed simulation over heterogeneous devices is an emerging area of research. Most of the works proposed in PDES cover execution over a multi-tenant cloud environment with the main focus on improving efficiency. On the other hand, traditional frameworks are designed to work in a controlled cluster environment. In all of the above frameworks, network and node reliability is assumed by default, which is not the case with IoT based frameworks. A few such IoT based simulation frameworks have been proposed in literature (as discussed above). Most of these framework provides a closed simulation environment where the dynamic behavior of IoT devices is not captured. Due to the mobility of IoT devices, the devices connectivity and dis-connectivity can affect the simulation performance and outcomes. Similarly, the available frameworks are developed on top of network simulators, which mainly focus on studying network parameters such as packet delivery ratio and transmission delay. These are relatively different objectives as compared to that of a PDES simulation framework involving IoT devices. Last but not the least, due to the complexity of optimistic simulation over IoT devices, the existing works cover the conservative simulation approaches only. The proposed framework is based on a master-worker paradigm managing dynamic behavior of devices and distributing simulation tasks. This framework considers parameters like context-awareness, network delays, and packet transfer rate for task distribution. The master node manages the execution of the entire simulation including the management of devices churn behaviour. Furthermore, the proposed framework also supports multiple master nodes, where a node can act as a master node for a group of nearby available IoT devices. Thus, the proposed framework provides a highly scalable environment.

III. System Model

We consider a framework based on an undirected graph $G = (V, E)$ to support distributed simulations in a heterogeneous environment. Here, the device set V comprises of a master

TABLE II
COMPARISON OF DIFFERENT IOT SIMULATION FRAMEWORKS

IoT Simulators	Platform Independent	Open Source	Simulation Model (O/C/H/NA)	Context-aware Distribution	Heterogeneous Device Support	Intended for (R/A/I)*	Support User-defined Scheduling Algo.
SimpleIoTSimulator [1]	×	×	C	×	×	I	×
MobIoTSim [24]	×	✓	C	×	×	A/R	×
IBM BlueMix [2]	Cloud-based	×	NA	×	×	I	×
Google Cloud IoT [3]	Cloud-based	×	NA	×	×	I	×
iFogSim [25]	✓	✓	C	×	×	A/R	✓
Cooja [4]	×	✓	NA	×	×	A,R	×
FogTorch [26]	✓	✓	NA	×	×	A/R	✓
RECAP [27]	-	✓	NA	×	×	A/R	×
EmuFog [28]	✓	✓	NA	×	×	A/R	✓
EdgeCloudSim [29]	✓	✓	C	×	×	A/R	✓
VirtualFog [30]	×	✓	C	×	×	A/R	✓
D2D fogging [31]	-	-	C	✓	×	R	✓
FogNetSim++ [32]	✓	✓	C	Limited	Limited	R	✓

*R: Researchers, A: Academics, I: Industry, O: Optimistic, C: Conservative, H: Hybrid, NA: Not Available

node B supported by N heterogeneous IoT devices referred to as the workers W. The heterogeneous devices W can be of different types such as mobile W_m, static W_s devices, and desktop systems W_d; that is, $V = B \cup (W_m \cup W_s \cup W_d)$. Furthermore, the set of edges E represents the master-to-worker and worker-to-worker links. The nodes can be located at geographically distributed locations.

The master node B offloads simulation events as tasks to its worker nodes that are available or registered with the network. These tasks (events) are initially held within a queue termed as the event queue Q. The queue is maintained at the master node that communicates with its workers through simulation messages M. Once the tasks are assigned, the master node monitors their progress through heartbeat messages M_{hb}. The messages make it possible to manage the activities and status of worker nodes. Additionally, the messages keep track communication delays d_c between the nodes. Note that the workers may be mobile keep changing the delay over time, in turn, affecting the overall simulation system performance. The average delay between the master B and devices W is expressed as the sum of individual link delays $\{d_c^1, d_c^2, \cdots, d_c^N\}$ divided by the total number of worker nodes $|W|$.

$$\mu = \frac{1}{|W|} \sum_{x \in W} d_c^x \tag{1}$$

To balance workload, the proposed framework supports indirect connections among devices V. Therefore, when assigning tasks T, the master node takes into account the number of hops h to the workers in W. Furthermore, the framework supports *super-peer* configuration where worker nodes can act as masters to their nearby devices, to execute any supplementary tasks generated during execution.

The master nodes B offload computations to their worker nodes W, which send back results R upon completion of execution. Most commonly used traffic model such as master/slave configuration is employed, where the master communicates with its slave nodes at any point in time. Notably, there is heterogeneity in such settings, that is due to the different computation capabilities of slaves, the latency/bandwidths of communication channels and/or both. In such heterogeneous environments, heuristics are used as parameters for scheduling to effectively minimize total response times [42].

Initially, the system comprises of one master node. Later on other nodes can act as super-peers (master nodes) for a limited number of devices and tasks. In the former case, M/M/1 configuration is used with queue capacity denoted by K. The mean rate of message arrival is denoted as λ and mean service rate as μ, equaling $1/E[\text{arrival time}]$ and $1/E[\text{service time}]$, respectively. Here, $E[\cdot]$ is the expectation operator. ρ is the ratio of arrival and service rate *i.e.* $\rho = \lambda/\mu$. In the implementation, we have assumed that λ and μ are exponentially distributed. Therefore, the average queue length l_q for a single server queue is,

$$l_q = \frac{\rho^2}{(1-\rho)} \tag{2}$$

Similarly, the average wait time in the queue t_q is,

$$t_q = \frac{l_q}{\lambda} = \frac{\rho}{\mu(1-\rho)} \tag{3}$$

The average waiting time in the system t_w is,

$$t_w = t_q + \frac{1}{\mu} \tag{4}$$

Every request generated from master node is a compute-intensive event ($e \in Q$), which is offloaded using a message

[1]http://www.smplsft.com/
[2]https://console.ng.bluemix.net/
[3]http://www.contiki-os.org/
[4]https://cloud.google.com/solutions/iot/

($m \in M$) to a selected device ($w \in W$). The events offloaded to a particular device are stored in a local queue for unprocessed events Q_U, while processed events are stored in another local queue Q_P.

For all devices $w \in W$ an efficiency η is computed using different node-specific performance parameters, such as the total number events to be executed $m = |Q_U| + |Q_P|$, number of events executed successfully $n = |Q_P|$, network latency σ, and packet transfer rate δ. The efficiency η for any device $w \in W$ is a weighted sum of these parameters given as,

$$\eta = \alpha \cdot \frac{n}{m} + \beta \cdot \frac{1}{\sigma} + \gamma \cdot \begin{cases} 1.0 & \delta \geqslant 3Mbps \\ 0.5 & \text{otherwise} \end{cases} \quad (5)$$

where (α, β, γ) is set to $(0.50, 0.25, 0.25)$, respectively. As discussed in [3], efficiency in distributed simulation is defined as the ratio of committed to total events; therefore, a higher weight is assigned to the success ratio. Moreover, a device having more completed tasks shows its stability which is a critical parameter to compute the efficiency of a node, especially under high mobility environments. Similarly, 25% of weight is assigned to each network feature *i.e.* network latency and transfer rate. However, for PDES environments, network stability in terms of successful task execution is given a higher priority; otherwise, redistribution of tasks not only increases the simulation execution time but also generates more rollbacks; thus, can adversely impact the performance of distributed simulations in the IoT paradigm.

The resulting efficiency value η ranges between $[0-1]$. In an ideal situation, a node with $\eta \approx 1$ is considered to be efficient with high throughput, low latency, and high bandwidth, given the maximum utilization of the node. This value is updated over time using heartbeat messages M_{hb} and stored event logs. When scheduling an event, the nodes are arranged in a descending order based on their availability and the computed value of η. In summary, the mechanism minimizes the overall execution time and improves the reliability of the system.

IV. Proposed Distributed Simulation Framework For Heterogeneous Devices

The proposed DISim framework is designed to support distributed simulation over heterogeneous IoT devices as shown in Figure 2. These devices are available in large numbers but with limited resources and communication capabilities. Each device has its own operating system and set of resources. To achieve interoperability, a middle-ware is designed to facilitate the integration of these IoT devices in the form of a network of nodes. The nodes are permitted to join and leave the network at any time, with a master node keeping a record of their availability using control messages. Above the middle-ware layer, core modules of the proposed framework are implemented comprising of master and worker modules.

The framework implements a dynamic mechanism that manages the joining and leaving of devices at simulation run-time. Thus, it allows handling of newly connected devices to become part of the simulation grid, either directly to the master node or indirectly via other nodes. Similarly, both direct and indirect task distributions are supported as shown in Figure 3.

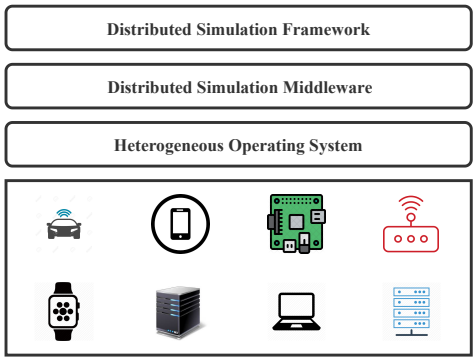

Fig. 2. High-level Architecture of the Proposed Framework

The master node initiates the simulation and manages execution on all of its worker nodes. In addition to executing the tasks assigned by the master node, a worker node can also act as an intermediate node between the master and another worker node, that is acting as a master for all other nodes directly connected to it. Such intermediary nodes are generally termed as *super-peer* nodes, capable of dynamically forming a network of nearby devices used to execute received tasks and/or other sub-tasks.

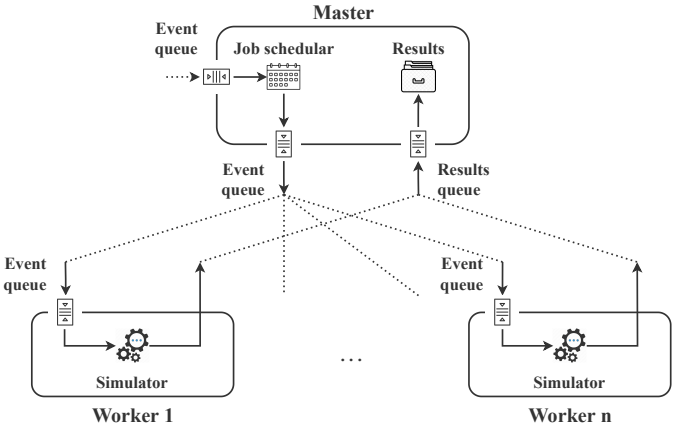

Fig. 3. Master-Worker Model - Worker Node can also be a Master Node

In summary, the master node is at the core of the proposed framework managing the entire simulation tasks and its outcomes. The master node manages the worker nodes, and provides the facilities for new devices to join the framework at any point in time. The framework can be divided into two types of nodes based on their functionality *i.e.* master nodes and worker nodes. The nodes architecture as used by the proposed framework is detailed in the following sections.

A. Master Node

Master node is implemented as a demon process *i.e.* running in the background, and manages the entire framework using worker and super-peer nodes. All processes get initiated by the master node, for instance, to track generated events, to receive processed data, and to monitor worker nodes. The core components of the master node are shown in Figure 4.

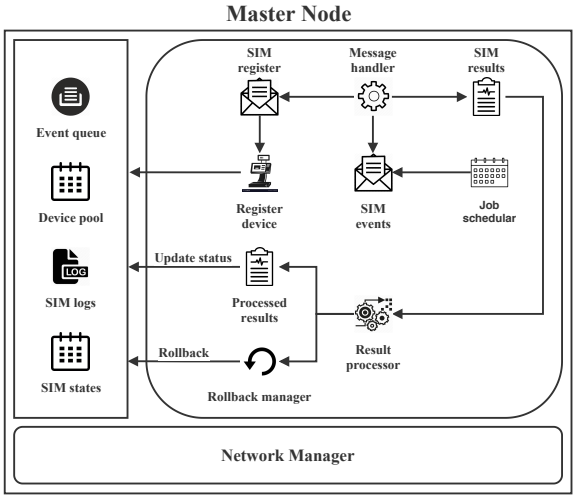

Fig. 4. Master Node's Internal Architecture - Modules and their Interactions

The overall simulation involves all the components shown in the master node, including the management of client-server relationship between master and worker nodes. It is pertinent to note that the master node acts as a client and worker nodes provide the services *i.e.* the execution of simulation tasks. Moreover, all workers directly communicate with their master node. The master node maintains a hashmap data structure based repository of all connected and available worker nodes. The simulation events are distributed among the available worker nodes by the master, with the help of super-peers, wherever needed. The master node also keeps information about the *track record* of all the connected devices/nodes, in order to ensure task assignment to more reliable nodes during the subsequent stages of simulation. The main modules of the master node are detailed below:

*a) **Job Scheduler**:* manages event distribution among available workers. The scheduling is performed based on several parameters including network latency, packet transfer rate, and history of the node. As the events are processed, their states get updated in the repository maintained by the master node. The states are stored to support rollbacks, if needed. Furthermore, executed events are marked successful after storing their outcomes. Note that the scheduler distributes tasks based on context, a key feature of the proposed framework, assigning tasks to workers with the highest response rate and minimal communication delay. Thus, improving the overall simulation system efficiency. The underlying algorithm used to schedule events is presented in Algorithm 1.

Algorithm 1 Job Scheduler

Input S: current simulation
Output w: worker node

SetState(S, START)
while GetState(S) $\neq$ FINISHED **do**
  repeat
    $w \leftarrow$ GetWorker() ▷ Find a suitable worker node
  until w **IS** ELIGIBLE
  $e \leftarrow$ GetEvent()
  Send(e, w)
  SetState(e, SCHEDULED)
end while

*b) **Reducer**:* manages the outcomes of events from all of the involved worker nodes. It gathers and stores them in the repository at the node level. Moreover, the module also coordinates with the scheduler to keep track of events and new assignments. This coordination is performed through the shared repository. As mentioned earlier, the framework supports super-peers, which can act as reducers for their directly-connected devices. The implementation used by the reducer is presented in Algorithm 2.

Algorithm 2 Reducer

Input S: current simulation; w: worker thread
Output e: executed event

while GetState(S) $\neq$ FINISHED **do**
  Wait() ▷ Wait for message
  $e \leftarrow$ ReceiveEvent()
  if $e = \phi$ **then**
    SetState(e, HANDLED)
  else
    SetState(e, EXECUTED)
    SetState(w, AVAILABLE)
  end if
end while

*c) **Network Manager**:* manages network connectivity between the master and its workers. The objective is to provide network interoperability among heterogeneous devices. The underlying mechanism is presented in Algorithm 3. During simulation execution, this module checks different simulation state variables to maintain communication between nodes. On simulation completion, the module closes all network connections with the devices, except, to receive heartbeat messages M_{hb} from the devices to show their availability.

Algorithm 3 Network Manager

Input S: current simulation
Output m: simulation message

SetupServerSocket()
while GetState(S) $\neq$ FINISHED **do**
  Wait() ▷ Wait for connections
  $t \leftarrow$ StartThread() ▷ New thread for connection
  $m \leftarrow$ Recv() ▷ Receive message
  if $m \neq \phi$ **then**
    $s \leftarrow$ GetSender(m)
    Process(m, s)
  end if
end while

*d) **Simulation Module**:* runs a modified TW algorithm over heterogeneous devices. In general, the TW algorithm uses optimistic synchronization to support distributed simulation in a cluster environment, where each Logical Process (LP) executes events independently without coordinating with other LPs [43]. Each LP executes a set of events without requiring synchronization until a causality error occurs. This triggers the rollback mechanism to undo the execution of any out-of-order events. The events are re-executed in a time-stamped order. Moreover, to cancel the generated events, the concept of *anti-messages* is used. In summary, the algorithm is designed to work in a peer-to-peer fashion where the same codes get executed on a large number of processes without synchronization. Clearly, this is unsuitable for the proposed framework where low-energy devices can become part of the distributed

simulation at any instant of time. Moreover, to keep track of device availability, master/worker paradigm is adopted.

The TW protocol is modified to work in a master/worker paradigm. The resulting implementation minimizes the TW overhead like memory usage and Global Virtual Time (GVT) calculation cost. In the proposed framework, the history of events and simulation states are stored at the master node; thus, rollback can only occur at the master node. Moreover, the rollback mechanism on devices can quickly drain their energy, thus it makes these devices unsuitable for optimistic simulation. Further, storing states and event history at a master node can ease the GVT computation without the involvement of worker nodes/devices. Thus, the above modifications allow a device to participate in the simulation at any time instant. On successful execution of an event, the GVT is checked and updated accordingly. In case there are causality errors, mainly due to network latency and/or device specifications, these are handled by subsequent rollbacks initiated by the master node. For this purpose, the master node uses a repository containing information about all running and executed events. The modified TW algorithm supporting optimistic simulation over low energy devices arranged in a master/worker paradigm is presented in Algorithm 4.

Algorithm 4 Modified TW using Master/Worker Paradigm

Input W: worker pool; t_{end}: simulation end time
Output S: simulation state variables

$t_{GVT} \leftarrow 0$ ▷ Timer for GVT computation
$Q_U \leftarrow Q_P \leftarrow \{\phi\}$ ▷ Unprocessed/Processed messages queue
Start t_p
while $t_p < t_{end}$ **do**
  Enqueue(Q_U, j) ▷ Enqueue job from buffer
  $m \leftarrow$ Dequeue(Q_U)
  if m **IS** REG_MSG **then** ▷ Registration message
    $count \leftarrow$ RegisterDevice()
  end if
  if m **PROCESSED then**
    Enqueue(W, GetDevice(m))
    $S \leftarrow$ UpdateStates()
  end if
  if $antimessage$ **ARRIVED then**
    Annihilate() and Rollback()
  end if
  if $W \neq \phi$ **then**
    $w \leftarrow$ Dequeue(W) ▷ Get available device
    $e \leftarrow$ Dequeue(Q_U) ▷ Get unprocessed event
    SendToDevice(w, e)
  end if
  if t_{GVT} **EXPIRED then**
    Reset(t_{GVT})
    FlushEvents(Q_p, ComputeGVT())
  end if
end while

B. Worker Node

The worker node executes events assigned to it by the master node. As mentioned earlier, any device in the distributed environment can act as a worker *i.e.* it can be a mobile or static device with varying processing and network connectivity features. The main objective of using these nodes is to utilize the processing power available on these devices. The basic architecture of such a worker node is presented in Figure 5. In case of worker nodes, the network manager is responsible for establishing connections and sending regular heartbeat messages M_{hb} to their master node, which may be a super-peer. Using a local simulation module, the worker node maintains an events queue and a message log to complete assigned tasks using the worker node Algorithm 5. The functionality of different modules of a worker node is detailed as follows:

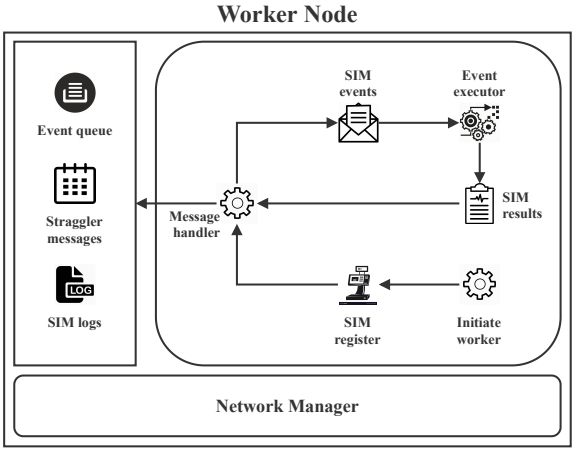

Fig. 5. Worker Node Internal Architecture - Modules and their Interactions

Algorithm 5 Worker node – Simulation Model

Input S: current simulation
Output e: executed event; S_v: State variables

$t \leftarrow$ InitThread() ▷ Initialize worker thread
success $\leftarrow$ Register(t) ▷ Send register message to master
if success **then**
  SetupSocket() ▷ To receive messages from master
  $Q \leftarrow \{\phi\}$ ▷ Incoming message queue
  while GetState(S) = RUNNING **do**
    $m \leftarrow$ Recv() ▷ Receive message
    if m **IS** SIM_END **then**
      $Q \leftarrow \{\phi\}$ and SetState(S, COMPLETED)
    else ▷ Normal message
      if antimessage **ARRIVED then**
        $m \leftarrow$ Dequeue(Q)
      else
        Enqueue(Q,m)
        $m_t \leftarrow$ Dequeue(Q) ▷ Pop top event
        Process(m_t)
        Send(B, UpdateStates(m_t))
      end if
    end if
  end while
else
  Quit() ▷ Simulation not running on master
end if

*a) **Message Handler**:* deals with all communications between the master and worker nodes. The communication happens as different types of messages like simulation time-stamp messages and messages related to registration with the master nodes. In the later case, as worker nodes can become part of the simulation grid at any time instant. For this purpose, a worker node needs to send only a register message with information of its available resources. Over the course of simulation execution, the message handler receives different messages and handles them accordingly. Once the simulation is completed, the worker node receives an end of the simulation message from master, that is the worker keeps its communication channel open until it receives this message.

*b) **Event Queue**:* maintains the events received from the master node, for execution on the worker node. It communicates with the message handler to add or remove events from the queue, particularly the case when straggler messages are received. The events in this queue are arranged using time-stamp messages, in their chronological order. The worker process picks up the earliest event for execution, as per the time-stamp order.

*c) **Event Executor**:* gets the top event (earliest) from the queue, executes it and forwards the outcomes in the form of state variables (S_v) to the result manager for onward submission to the master node (B). The worker module takes into account the available resources when executing events. This is done according to the requirements of the simulation.

*d) **Result Manager**:* prepares the results as per agreed format between master and worker node. This module is added to provide interoperability among heterogeneous devices. Thus, at the time of worker node registration, the master and worker node also agree on the format of result exchange.

V. A Motivating Application

Distributed simulation over embedded heterogeneous systems is an emerging area with many applications where devices are used to process real-time information to predict future states. Similarly, in many applications, a network of sensors is used to dynamically monitor changes in a physical system. In a real environment, the sensors/devices can be used to run distributed coordinated simulation where real-time information is used to project the outcome using models such as dead reckoning and etc. Consider a collection of Unmanned Aerial Vehicles (UAVs) monitoring urban traffic, forest fire, or a chemical accident. These small battery-operated UAVs are assigned a specific geographical area to monitor and communicate with a centralized system in case of an anomaly and/or provide periodic updates. The distributed simulation model implemented at each UAV can take real-time information to predict traffic congestion or the direction of a forest fire. The simulation model adopted for such scenarios is similar to time warp protocol. In case the prediction is beyond some predefined criterion, the entire simulation rollbacks and computes new states. Such applications are classified as Dynamic Data-driven Application Systems (DDAS), as they rely on data collected from various channels, which gets processed to define future tasks accordingly. However, energy is a major concern for applications relying on UAVs. Over the years, significant research has been undertaken to optimize or extend the battery life of UAVs. The techniques such as UAV swapping, battery hot-swapping, and wireless power transfer have been proposed and extensively used [44]. In UAV swapping, the low-power UAVs are sequentially switched out with fully charged UAVs. However, in battery hot-swapping, fully-charged batteries are plugged in; thus, reducing the charging time. In wireless charging, electromagnetic field (EMF) charging is used to transfer energy quickly over a short range. Thus, UAVs can be charged in minutes by hovering over the grid. Whereas, in non-EMF systems, solar radiation is used to charge the UAVs; however, for solar charging, high altitude UAVs are suitable.

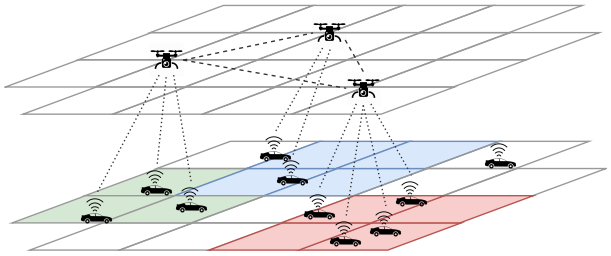

Fig. 6. Traffic monitoring system based on UAVs to predict future states using real-time and shared information in a distributed simulation environment.

VI. Simulation Results

In order to benchmark the proposed distributed simulation framework over heterogeneous devices, the PHOLD benchmark application is used as an instrumentation tool. Table III summarizes the master and worker node specification used to create a heterogeneous architecture environment. The simulation parameters used for the evaluation are listed in table IV.

TABLE III
System Specifications: Master / Worker Nodes

Parameters	Master node	Device type 1	Device type 2
CPU	i5 Hexa-core	Octa-core	Quad-core
Clock speed	2.8GHz	1.2GHz	1.4GHz
RAM	16GB	3GB	2GB
Storage	256GB	32GB	16GB
OS	Windows	Android	Android
OS version	8.1	5.1	6.0
Manufacturer	Intel	Huawei	Samsung

TABLE IV
Simulation Parameters

Parameters	Values
Number of processes	1024
Master node	Dedicated system
Worker nodes	Handheld devices and dedicated nodes
Communication	TCP
Simulation message size	4096 bytes
Implementation language	Java Android
Total execution time	1440 mins
Simulation model	Optimistic approach
GVT computation	5 mins
Simulation topology	Grid

Scheduling Algorithms – Event scheduling for heterogeneous devices plays an important role in improving the efficiency of a PDES platform. In this study, we have implemented two scheduling algorithms: (a) randomized scheduler selects a free worker node randomly for event execution, and (b) the context-aware scheduler selects a worker node based on certain parameters, for example packet transfer rate, network delay, channel quality and overall node efficiency. The efficiency of a device is computed using eq. 5. This value is updated at the master node over time using heartbeat messages M_{hb}. Thus, in order to schedule an event, the available devices are arranged in a descending order of their efficiency *i.e.* their computed value of η. The topmost device is selected from the pool for event execution.

Queuing Nodes and Grid Network Simulation – A benchmark application is implemented in this study, which is similar

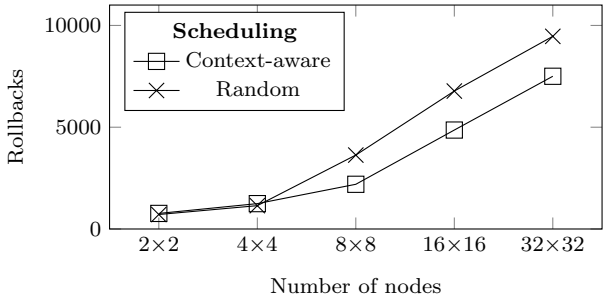

Fig. 7. Rollbacks for 2×2, 4×4, 8×8, 16×16, and 32×32 worker nodes over the entire simulation time.

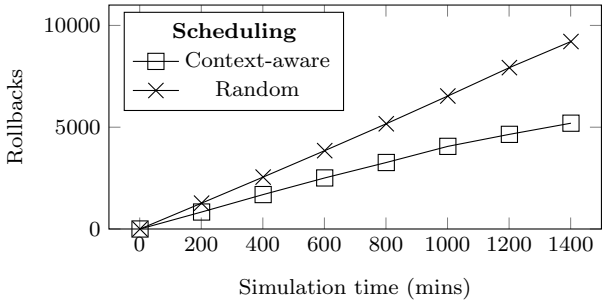

Fig. 8. Rollbacks for 32×32 grid network with respect to simulation time. Note that the simulation ran for 24 hours.

to the one discussed earlier as a motivating example. The application is a simulation of queuing network in a two-dimensional grid where each process is assigned a specific area on the grid. Jobs are maintained at the master node whereas the worker nodes are logical processes allocated in the grid. The minimum service time t is ensured through the scheduling algorithms. Figure 7 represents the execution of the queuing network for grid sizes: 4 (2×2), 16 (4×4), 256 (16×16), and 1024 (32×32) nodes. Here, the performance of the scheduling algorithms in terms of rollbacks over different grid sizes is presented. Similarly, Figure 8 presents the performance of the scheduling algorithms with 1024 nodes (32×32) reported over a time span of 24 hours (1440 minutes).

Figure 9 shows the efficiency comparison of scheduling algorithms for PDES tasks over IoT devices. Recall that efficiency is the ratio of committed events to the total number of events. The result shows that the context-aware model performs better compared to random device selection strategy. Furthermore, the overall efficiency achieved is around 80%. These results demonstrates that using more information improves the efficiency of the distributed simulation.

Power consumption in embedded systems plays an important role in considering the overall performance. During the simulation execution, the power usage is benchmarked on worker nodes. Figure 10 represent the power consumption of a device used to simulate 1024 (32×32) nodes in a grid network. The power consumption has highly related to the number of events processed during the simulation. Furthermore, the number of event rollbacks is a critical factor that determines the performance *i.e.* lesser the rollback events means less power consumption and improved simulation efficiency. The overall power consumption during the simulation varies between 30-35% on mobile devices used for simulation. In comparison, the context-aware approach performs slightly better than the random approach in terms of power consumption, thanks to the better rollbacks-to-event ratio.

VII. Discussion

Traditional frameworks lack the support for heterogeneous devices to become a part of the distributed simulation network. In contrast, the proposed framework is designed to permit joining of heterogeneous devices with varying degree of computational resources and subsequently participating in the distributed simulation. Moreover, as the devices are dispersed

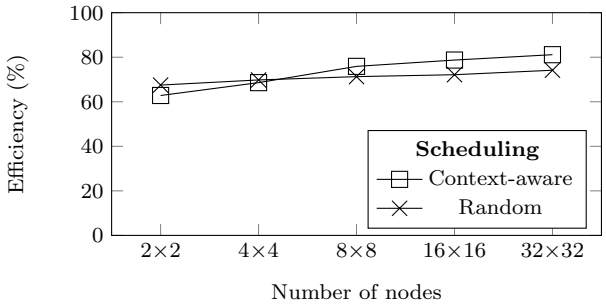

Fig. 9. Efficiency of event scheduling on nodes for 2×2, 4×4, 8×8, 16×16, and 32×32 grid networks.

over multiple networks, which requires extensive modifications to the traditional systems in order to deal with the caveats and problems of a large-scale heterogeneous system.

There is a significant difference between the computational capabilities of a desktop machine as compared to a hand-held mobile device, making it difficult to synchronize processes. Therefore, a comprehensive approach is necessary to take care of varying computational resources and network delays. The goal is to utilize the available resources in an efficient manner while completing a task. In the proposed framework, statistics regarding each of the worker nodes and employ locality information is maintained to ensure efficient usage of simulation system resources. It is evident from the results that the use of such scheduling algorithms have significant impact on the performance of a distributed simulated system.

All in all, this study outlines the ground realities and issues faced when working with an environment of diverse compu-

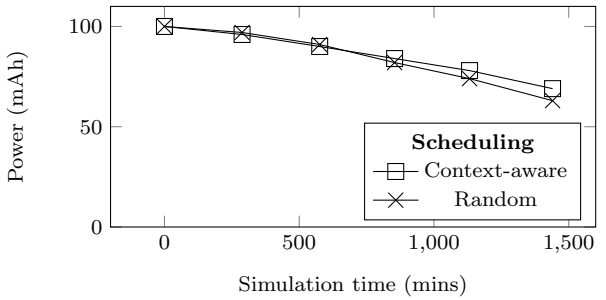

Fig. 10. Power consumption for 32×32 grid network with respect to simulation progress. Note that 100% battery is used up during the runs.

tational resources. The proposed framework demonstrates that context-aware scheduling performs well by maintaining node profiles improving simulation performance. Furthermore, the framework can be extended to implement AI-based scheduling using node profiles and simulation states. In the future, we are looking to add more sophisticated scheduling techniques into the framework and support for devices with specialized computing capabilities like graphics processors.

VIII. Availability

The framework is an open source project and is anonymously available on a source repository hosted by GitHub at https://github.com/mhaseeb-seecs/dsim-simulator. We expect contributors to add new methods, models and ideas to the framework. Nevertheless, the authors have an interest in developing new modules for simulation management and scheduling, and support specialized devices.

IX. Conclusions

The proposed PDES framework addresses the limitations of traditional simulation frameworks and has been designed to include hand-held devices. The framework dynamically manages the churn behavior of these devices as well as their sporadic connectivity. More specifically, the framework has been designed to use heterogeneous devices that are usually dispersed over multiple networks and encounter frequent connectivity issues. Moreover, the proposed implementation supports widely used hand-held devices as well as desktop platforms, in order to provide an inclusive simulation framework. It provides a set of basic simulation tools with parameters to enhance the performance of distributed simulations. The framework manages fault tolerance and uses context-aware services to improve the efficiency of distributed simulation over IoTs.

In terms of framework's evaluation, the proposed context-aware feature is compared with random task distribution over IoT devices. Experimental results show a 28% reduction in the overall rollbacks when simulated over a time span of 24 hours. Moreover, above 80% efficiency is observed when using context-aware task distribution, which includes the dynamic management of devices' churn behavior. Similarly, as the power consumption is directly proportional to total number of events processed at any device, the overall power consumption is reduced using the proposed framework due to lesser number of rollbacks. This feature is especially important for battery operated IoT devices participating in the PDES simulation. We strongly believe that this framework can serve as a foundation for researchers to design simulations and run them across a network of heterogeneous IoT devices in a distributed environment. This effort makes the overall process a cost-effective one as it is able to deal-with node failures during simulation runs.

In future, we would like to adopt the dynamic task migration feature using context-awareness and device constraints. Moreover, artificial intelligence based techniques can be incorporated for selecting the most suitable worker nodes for executing simulation events.

Acknowledgment

The work was supported by the Faculty Program, University of Malaya, under Grant GPF019D-2019.

References

[1] Q. He, M. Ammar, G. Riley, and R. Fujimoto, "Exploiting the predictability of tcp steady-state to speed up network simulation," *Performance Evaluation*, vol. 58, no. 2-3, pp. 163–187, 2004.

[2] R. M. Fujimoto, K. S. Perumalla, and G. F. Riley, *Network simulation*. Morgan & Claypool Publishers, 2006, vol. 1, no. 1.

[3] A. Malik, A. Park, and R. Fujimoto, "Optimistic synchronization of parallel simulations in cloud computing environments," in *2009 IEEE International Conference on Cloud Computing*. IEEE, 2009, pp. 49–56.

[4] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment," *UCLA computer science department technical report*, vol. 990027, no. 1999, p. 213, 1999.

[5] T. J. Delve and N. J. Smith, "Use of dassf in a scalable multiprocessor wireless simulation architecture," in *33rd Conference on Winter simulation*. IEEE Computer Society, 2001, pp. 1321–1329.

[6] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello, "Scalable and efficient parallel and distributed simulation of complex, dynamic and mobile systems," in *2005 Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-PERF'05)*. IEEE, 2005, pp. 136–145.

[7] G. D'Angelo and S. Ferretti, "Highly intensive data dissemination in complex networks," *J. Parallel Distrib. Comput.*, vol. 99, pp. 28–50, 2017.

[8] A. I. McInnes and B. R. Thorne, "Scipysim: towards distributed heterogeneous system simulation for the scipy platform (work-in-progress)," in *2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International, 2011, pp. 89–94.

[9] A. Pellegrini, R. Vitali, and F. Quaglia, "The rome optimistic simulator: Core internals and programming model," in *4th International ICST Conference on Simulation Tools and Techniques*, 2011, pp. 96–98.

[10] L. Toscano, G. D'Angelo, and M. Marzolla, "Parallel discrete event simulation with erlang," in *1st ACM SIGPLAN workshop on Functional high-performance computing*. ACM, 2012, pp. 83–92.

[11] P. F. Riley and G. F. Riley, "Spades—a distributed agent simulation environmentwith software-in-the-loop execution," in *2003 Winter Simulation Conference S. Chick, PJ Sánchez, D. Ferrin, and DJ Morrice, eds*, 2003, pp. 817–825.

[12] G. D'Angelo, S. Ferretti, and M. Marzolla, "Time warp on the go," in *5th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS '12. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012, pp. 242–248.

[13] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, no. 10, pp. 30–53, 1990.

[14] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, 1999.

[15] H. Rajaei, R. Ayani, and L.-E. Thorelli, "The local time warp approach to parallel simulation," in *ACM SIGSIM Simulation Digest*, vol. 23, no. 1. ACM, 1993, pp. 119–126.

[16] D. Jefferson and R. Fujimoto, "A brief history of time warp," in *Advances in Modeling and Simulation*. Springer, 2017, pp. 97–134.

[17] A. C. Palaniswamy and P. A. Wilsey, "Parameterized time warp (ptw): an integrated adaptive solution to optimistic pdes," *J. Parallel Distrib. Comput.*, vol. 37, no. 2, pp. 134–145, 1996.

[18] G. D'Angelo, S. Ferretti, and V. Ghini, "Distributed hybrid simulation of the internet of things and smart territories," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 9, p. e4370, 2018.

[19] G. Fortino, R. Gravina, W. Russo, and C. Savaglio, "Modeling and simulating internet-of-things systems: a hybrid agent-oriented approach," *Comput. Sci. Eng.*, vol. 19, no. 5, pp. 68–76, 2017.

[20] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 58–67, 2011.

[21] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli, "A simulation platform for large-scale internet of things scenarios in urban environments," in *1st International Conference on IoT in Urban Space*, 2014, pp. 50–55.

[22] F. Maqbool, A. W. Malik, I. Mahmood, and G. D'Angelo, "Seecssim-a parallel and distributed simulation framework for mobile devices," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2018, pp. 1–7.
[23] E. Di Pascale, I. Macaluso, A. Nag, M. Kelly, and L. Doyle, "The network as a computer: A framework for distributed computing over iot mesh networks," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2107–2119, 2018.
[24] T. Pflanzner, A. Kertész, B. Spinnewyn, and S. Latré, "Mobiotsim: towards a mobile iot device simulator," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2016, pp. 21–27.
[25] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[26] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1185–1192, 2017.
[27] J. Byrne, S. Svorobej, A. Gourinovitch, D. M. Elango, P. Liston, P. J. Byrne, and T. Lynn, "Recap simulator: Simulation of cloud/edge/fog computing scenarios," in *2017 Winter Simulation Conference (WSC)*. IEEE, 2017, pp. 4568–4569.
[28] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress (FWC)*. IEEE, 2017, pp. 1–6.
[29] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," vol. 29, no. 11. Wiley Online Library, 2018, p. e3493.
[30] J. Li, J. Jin, D. Yuan, and H. Zhang, "Virtual fog: A virtualization enabled fog computing framework for internet of things," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 121–131, 2018.
[31] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3887–3901, 2016.
[32] T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan, "Fognetsim++: A toolkit for modeling and simulation of distributed fog environment," *IEEE Access*, vol. 6, pp. 63 570–63 583, 2018.
[33] G. D'Angelo and M. Marzolla, "New trends in parallel and distributed simulation: From many-cores to cloud computing," *Simulation Modelling Practice and Theory*, vol. 49, pp. 320–335, 2014.
[34] R. M. Fujimoto, "Research challenges in parallel and distributed simulation," *ACM Trans. Model. Comput. Simul.*, vol. 26, no. 4, p. 22, 2016.
[35] S. Serrano-Iglesias, E. Gómez-Sánchez, M. L. Bote-Lorenzo, J. I. Asensio-Pérez, and M. Rodríguez-Cayetano, "A self-scalable distributed network simulation environment based on cloud computing," *Cluster Comput.*, vol. 21, no. 4, pp. 1899–1915, Dec. 2018.
[36] M. Principe, T. Tocci, A. Pellegrini, and F. Quaglia, "Porting event &cross-state synchronization to the cloud," in *2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 2018, pp. 177–188.
[37] G. Peng, H. Wang, J. Dong, and H. Zhang, "Knowledge-based resource allocation for collaborative simulation development in a multi-tenant cloud computing environment," *IEEE Trans. Serv. Comput.*, vol. 11, no. 2, pp. 306–317, 2018.
[38] M. Gütlein, R. German, and A. Djanatliev, "Towards a hybrid cosimulation framework: Hla-based coupling of matsim and sumo," in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2018, pp. 1–9.
[39] X. Wang, S. J. Turner, M. Y. H. Low, and B. P. Gan, "Optimistic synchronization in hla-based distributed simulation," *Simulation*, vol. 81, no. 4, pp. 279–291, 2005.
[40] L. Ziganurova and L. N. Shchur, "Synchronization of conservative parallel discrete event simulations on a small-world network," *Phys. Rev. E*, vol. 98, no. 2, p. 022218, 2018.
[41] A. Falcone, A. Garro, S. J. Taylor, and A. Anagnostou, "Simplifying the development of hla-based distributed simulations with the hla development kit software framework (dkf)," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2017, pp. 1–2.
[42] J.-F. Pineau, Y. Robert, and F. Vivien, "The impact of heterogeneity on master-slave scheduling," *Parallel Comput.*, vol. 34, no. 3, pp. 158–176, 2008.
[43] D. Jefferson, B. Beckman, F. Wieland, L. Blume, and M. DiLoreto, "Time warp operating system," in *SOSP '87 Proceedings of the eleventh ACM Symposium on Operating systems principles*. ACM, 1987, pp. 77–93.
[44] B. Galkin, J. Kibilda, and L. A. DaSilva, "Uavs as mobile infrastructure: Addressing battery lifetime," *IEEE Commun. Mag.*, 2019.

Muhammad Haseeb received the master's degree in parallel and distributed systems from National University of Sciences and Technology (NUST), Pakistan in 2019 and Bachelor degree in computer science from Bahauddin Zakariya University (BZU), Pakistan in 2015. His main research interests include distributed simulation, cloud/fog computing, internet of things, machine learning, automated systems, and natural language processing.

Asad Waqar Malik is an Assistant Professor at the Department of Computing (DOC), NUST School of Electrical Engineering and Computer Science (SEECS). Besides, he is also working as Senior Lecturer at the Department of Information Systems, Faculty of Computer Science & Information Technology, University of Malaya, Malaysia. He finished his Ph.D. with majors in parallel and distributed simulation/systems from National University of Science and Technology (NUST), Pakistan in 2012. His primary area of interest includes distributed simulation, cloud/fog computing, and internet of things.

Anis ur Rahman received the master's degree in parallel and distributed systems from Joseph Fourier University, France, and the Ph.D. degree in computer science from Grenoble University, France, in 2013. He is currently an Assistant Professor with the School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan. Besides, he is also working as Research Fellow at the Department of Information Systems, Faculty of Computer Science & Information Technology, University of Malaya, Malaysia. His main research interests include modeling of visual attention by assessing the different mechanisms guiding it, salient multi-object image and video segmentation and tracking, and efficient implementations of large-scale scientific problems on commodity graphical processing units.

Mian Muhammad Hamayun completed his PhD in Computer Science (System Modeling and Simulation) from University of Grenoble, France, in 2013. His PhD research focused on Native Simulation and Modeling of Multi-Processor System-on-a-Chip (MPSoC) using Hardware-Assisted Virtualization techniques. He is working as a Lecturer at the School of Computer Science, University of Birmingham Dubai, United Arab Emirates. His research interests include Virtualization Techniques, GPGPU Computing, SystemC/TLM based Virtual Prototyping, Cloud Computing and Internet of Things (IoT).