Journal of Network and Systems Management (2020) 28:1610–1638
https://doi.org/10.1007/s10922-020-09552-x

MaxHadoop: An Efficient Scalable Emulation Tool to Test SDN Protocols in Emulated Hadoop Environments

Claudio Calcaterra[1] · Alessio Carmenini[2] · Andrea Marotta[2] · Ubaldo Bucci[2] · Dajana Cassioli[2]

Received: 29 January 2020 / Revised: 28 June 2020 / Accepted: 2 July 2020 / Published online: 29 July 2020

Abstract

This paper presents MaxHadoop, a flexible and scalable emulation tool, which allows the efficient and accurate emulation of Hadoop environments over Software Defined Networks (SDNs). Hadoop has been designed to manage endless data-streams over networks, making it a tailored candidate to support the new class of network services belonging to Big Data. The development of Hadoop is contemporary with the evolution of networks towards the new architectures "Software Defined." To create our emulation environment, tailored to SDNs, we employ MaxiNet, given its capability of emulating large-scale SDNs. We make it possible to emulate realistic Hadoop scenarios on large-scale SDNs using low-cost commodity hardware, by resolving a few key limitations of MaxiNet through appropriate configuration settings. We validate the MaxHadoop emulator by executing two benchmarks, namely WordCount and TeraSort, to evaluate a set of Key Performance Indicators. The tests' outcomes evidence that MaxHadoop outperforms other existing emulation tools running over commodity hardware. Finally, we show the potentiality of MaxHadoop by utilizing it to perform a comparison of SDN-based network protocols.

Keywords SDN · Network emulation · Big Data · Hadoop

✉ Andrea Marotta
andrea.marotta@univaq.it

Claudio Calcaterra
claudio.calcaterra@unich.it

Alessio Carmenini
alessio.carmenini@student.univaq.it

Ubaldo Bucci
ubaldo.bucci@graduate.univaq.it

Dajana Cassioli
dajana.cassioli@univaq.it

[1] Department of Economic Studies, University G. D'Annunzio Chieti-Pescara, Pescara, Italy

[2] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy

1 Introduction

The Software Defined Networking (SDN) paradigm has been defined to convert the network environment into a new one, more intelligent and adaptable, to support new applications. This new paradigm introduces a centralized node, named Software Defined Network Controller (SDN-C), which eliminates the vertical integration of legacy networks [1]. This approach increases the flexibility and simplifies the network management by decoupling control plane and data plane. Network nodes are programmable by the SDN-C through the OpenFlow protocol [2]. Programmability of network nodes is granted by the introduction of appropriate levels of abstraction that can be accessed through the use of control interfaces or Application Programming Interface (API).

The interworking of SDN and Hadoop would be very beneficial to optimize the traffic load over network nodes as generated by Hadoop jobs. Its effectiveness has been demonstrated by several studies presented in the literature, as discussed in Sect. 3.

Several simulators and emulators have been proposed for the evaluation of SDN protocols, including investigations of Hadoop operation on SDNs. A good example is *Doopnet* [3]. However, as discussed in Sect. 3, the available tools present some limitations in terms of scalability, requirements on hardware resources and elaboration time.

A good approach is the use of Hardware in the Loop (HiL) to distribute the simulation/emulation load over multiple physical machines using a consolidated SDN emulator, like e.g. MaxiNet [4]. Maxinet is a SDN emulator which partitions the emulated network into *n* portions, and instantiates each one to the available physical machines using the *partitioning library METIS*. This allows the emulation of large-scale SDNs; however, the automatic mapping of network nodes is not appropriate to emulate an instance of a Hadoop Cluster because it lowers the ability to control the memory assignments necessary for the Hadoop job to run properly.

In this paper, we propose a framework to emulate large-scale SDNs without any architectural or topological limits. We exploit the HiL to distribute the emulation load over multiple physical machines. We make it possible to employ MaxiNet, by proposing a methodology to overcome the limitation of METIS' mapping of network nodes to the physical machines.[1] Starting from the identification of the requirements of all involved processes, appropriate settings and optimal mapping are derived. The mapping methodology is presented in the context of Hadoop, but it can be easily generalized to other "heavy" applications. It is crucial to apply the same methodology to any other scenario to be emulated in setups involving containers and HiL, in order to avoid any bias in the emulation results and bottlenecks.

In particular, in this paper we show how to enable the reliable emulation of a Hadoop cluster in MaxiNet; we formalise the memory requirements of Hadoop hosts and the physical constraints of the workers in terms of *design constraints of*

[1] In the following we refer to the *physical machines* as *workers*.

MaxiNet's setup. The distribution of the emulated network nodes is derived and statically mapped onto the workers. Increasing the number of workers provides the desired *horizontal scalability to MaxHadoop*, making it possible to emulate very complex architectures, widely adopted and implemented in today's tera-scale data centers, such as server-centric architectures like BCube and DCell [5], or switch-based networks, like Fat Tree[6]. It is possible to study a MapReduce job in different Fat Tree configurations and different routing protocols managed by the SDN controller[7].

Our tool is of fundamental importance in the context of the new Big Data sources, such as the IoT environment, where the main challenge is the management of huge amounts of data. We provide a scalable and flexible tool to the research community to test new protocols and to explore potential future Hadoop applications on SDNs.

MaxHadoop's scalability is prooved in this paper through tests based on two well known Hadoop benchmarks:

- *TeraSort*, a memory-intensive application that can order any amount of data quickly, in different configurations;
- *WordCount*, a CPU intensive benchmark, that counts the number of occurrences of each word in a given input set.

The results from the benchmarks are grouped to evaluate the main Hadoop and hardware KPIs during the MaxHadoop emulation of a datacenter, hosts, switches, network links and virtual topologies.

The test scenario consists of a tree topology with a depth of 2 with ten hosts distributed over two racks. The performed validation tests show the horizontal scalability of MaxHadoop by increasing the number of workers, its versatility by CPU and Memory intensive tests, and its easiness of configuration by varying network bandwidth and the memory assigned to nodes. The measurements of both the emulated cluster metrics and the utilization of hardware resources (CPU, RAM and network) in the workers allows us to demonstrate that there are no bottlenecks or oversizing of the hardware in the emulation setup. The performed validation tests show MaxHadoop's:

- *horizontal scalability*, by increasing the number of workers,
- *versatility*, by CPU- and memory-intensive tests,
- *easiness of configuration*, by varying network bandwidth and memory assigned to nodes.

Finally, we show the potentiality of MaxHadoop to support research on SDN-based network protocols, by implementing in MaxHadoop a typical use case of interworking of SDN and Hadoop in Data Centers (DC): the impact of different SDN-based routing strategies in a Fat-Tree topology.

The paper is organised as follows. In Sect. 3 we extensively review the state-of-the-art on simulation/emulation tools to support reliable investigations of the interworking of Hadoop and SDNs. In Sect. 1, the basics of MaxiNet and Hadoop are provided. In Sect. 4, our methodology for the correct host-to-memory mapping to

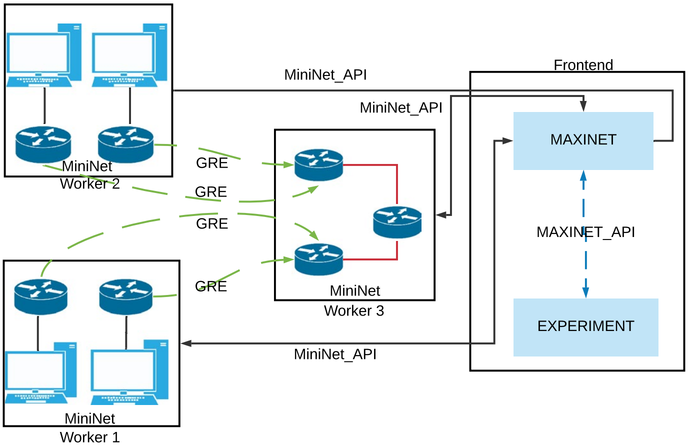

Fig. 1 Maxinet in a nutshell

fulfill the Hadoop requirements is presented. The Sect. 5 describes in details the MaxHadoop framework. Validation tests and performance evaluation of MaxHadoop are presented in Sect. 6. Finally, we draw a conclusion and plan future extensions of the present work in Sect. 7.

2 Background

We kick off with a detailed description of the enabling frameworks underneath MaxHadoop, namely MaxiNet and Hadoop.

2.1 MaxiNet

As we already highlighted, MaxiNet provides the capability of emulating a large-scale SDN network[4], where the throughput and latency on every link can be set by applying the *TCLink class* of MiniNet, being MaxiNet the extension of MiniNet. MaxiNet supports any external SDN controller. The controller is essential for the network operation, especially if multipath protocols, quality of service (QoS) or packet filtering policies are used. Through the controller, it is possible to program the OpenFlow switches to manage the data flows.

As shown in Fig. 1, MaxiNet can be deployed onto a pool of *workers*, that host an unmodified instance of MiniNet each. In practice, the whole network to be emulated is partitioned among different instances of MiniNet by METIS, which is a graph partitioning library. Workers communicate by means of generic routing encapsulation (GRE) tunnels. The whole emulation is controlled by the Frontend centralized API provided by MaxiNet. The Frontend is a specialized worker, which distributes the network to be emulated onto the whole pool of workers. The Frontend stores a list of nodes-to-workers mapping, which enables to access all nodes. MaxiNet scales nicely with an increasing number of physical machines. It provides a command line interface (CLI), as in Mininet, which helps debug the experiments and executes commands within the workers' scope. Then it supports the X-forwarding graphical user interfaces (GUIs) on both the workers and emulated hosts. Maxinet has native monitoring primitives/methods that allow the monitoring of the course of the experiments by means of the observation of CPU, memory and network usage.

Unfortunately, MaxiNet is not aware of the capabilities of workers. Thus, the automatic mapping could exceed the performance limits of the workers in the presence of the heavy load of Hadoop processes. To overcome this limitation, we introduce in Sect. 4 the host-to-memory mapping method to calculate either the number of workers needed to emulate the Hadoop cluster or the memory required by the containers used in the emulation.

Another relevant feature of the proposed emulation framework, worth to highlight, is the support of Docker containers. Docker overcomes the MiniNet limitation of emulating distributed environments, such as Hadoop, over a single file system. Docker is an open-source project for the creation of distributed systems, allowing different applications or processes to work autonomously on the same physical machine or different virtual machines[8]. The application and its dependencies are packaged up as a lightweight, portable, self-sufficient container. Since MiniNet uses a single Linux kernel for all virtual hosts, which share the host file system and process ID (PID) space, the use of Docker Containers, by, e.g., Containernet[9], provides a complete machine isolation between the virtual hosts and the machine's file system, allowing the installation of a real Hadoop cluster.

2.2 Hadoop

Hadoop is composed of two main elements, which are: Hadoop Distributed File System (HDFS) and Yet Another Resource Negotiator (YARN) framework. The HDFS is designed to be deployed onto low-cost hardware, to provide high fault-tolerance and to sustain batch-processing. Also, it has been demonstrated that HDFS is a good provider for those applications that require high-throughput and low-latency data access. Generally, the data sets of applications running on top of HDFS have a size ranging from gigabytes to terabytes. The HDFS cluster is composed by one *master*, named *NameNode* (NN), and usually one *DataNode* (DN) per node. The master manages the filesystem namespace, regulates the access to the files and executes operations like opening, closing, and renaming files and directories. Another important capability is the mapping of blocks to the DataNodes, where the *blocks* result from the file splitting procedure usually operated by the filesystem. The DataNodes perform the operations required by both the NameNode and the filesystem client.

Since version 2.0, Hadoop comes along with YARN which distributes into separate daemons the functionalities of resource management and job scheduling and monitoring. The YARN architecture is composed by four main elements: the ResourceManager (RM), the NodeManager (NM), the ApplicationMaster (AMt), and the Container (C). The RM is defined as the arbitrator of cluster's resources allocation to competing applications. The RM is composed of two modules, namely the Scheduler (SC) and the Applications-Manager (AMg). The SC is in charge of the resources allocation process, whereas AMg is responsible of accepting job submissions and for negotiating the container of AMt. The NM is a per-node daemon that manages the containers and monitors their resource usage, in terms of CPU, disk, network, and memory utilization, and provides a periodic report to the RM scheduler. The AMt is a component associated with a single application. The application is expressed as either a single job or a Directed Acyclic Graph (DAG) of jobs. The AMt negotiates the resources with the RM and works with NMs to perform and monitor the tasks. The Container is a collection of physical resources such as CPU cores, RAM, and disks on a single node.

An example of YARN application is the MapReduce job. MapReduce is a data massive generation and processing technique executed on a distributed cluster. It is based on the *divide and conquer strategy*. The MapReduce job is completed in three phases, namely the *Map* (M), *Shuffle* (S), and *Reduce* (R). The M phase takes as input a dataset and, based on programmer's decisions, produces a list of (key, value) pairs. This list is shuffled and sorted in the S phase, whose outcome is reduced in the R phase.

3 Related Work

The interworking of SDN and Hadoop would be very beneficial to optimize the traffic load over network nodes as generated by Hadoop jobs. Examples are the OFScheduler[10], a dynamic OpenFlow-based network communications optimizer for MapReduce operations, and the PANE controller[11], which assigns to Hadoop the network bandwidth required to run its operations (i.e., the jobs) in the shuffle phase and to write the final output in the Hadoop Distributed File System (HDFS). The interaction between Hadoop job scheduler and the SDN control plane using a method for aggregating optical links in intermediate devices has been demonstrated in[12] and effectively satisfies the bandwidth requests.

Over the past decade, cloud computing and related ICT technologies have developed rapidly, and in the interesting study by Abbasi et al. [13], the Software-Defined Cloud Computing (SDCC) paradigm emerges as an approach "software-defined" for automating the process of optimal cloud configuration by extending virtualization concept to all resources in a data center such as infrastructure, network, storage, control, protection and service level agreement.

Complete data centers transformation to SDCC principles will take years. During this process, several research works focus on implementation challenges in DC transformation, primarily in terms of programmability, scalability, interoperability, and security. Although the SDCC is in the early stages of development, global virtualization giants like VMware have adopted SDCC concepts with proprietary solutions and are able to support open cloud solutions, as Kubernetes (Tanzu) [14] and Docker (vSphere Integrated Containers)[15] including a wide range of virtualization services, management and orchestration platforms, storage resource managers, and hybrid-cloud deployment solutions.

The evolution of cloud services towards network emulation is described by Lai et al.[16]. This cloud-based network platform uses technologies such as NFV and

SDN to the network emulation domain and aims at providing to the users the "Network Emulation as a Service" (NEaaS), which can be conveniently deployed on both public and private clouds. To emulate networking nodes in a hybrid manner, Docker containers are used (representative of lightweight virtualization technology) as a supplement to virtual machine (VM) (heavy virtualization technology).

Zulu et al. [17] shows a simple hybrid implementation of the technology (cloud and on-premises), applied to a realistic programmable network that uses an SDN network with OpenDayLight controller hosted on Amazon Web Services. This controller is used as a control plane for a Mininet switch that allows communication between MiniNet hosts and a web server hosted on the Emulated Virtual Environment-Next Generation (EVE-NG).

So far, SDNs studies have been based on a plethora of simulators.

NS-3 simulator embeds a project which supports OpenFlow version 0.89[18], against the current version 1.5. Unfortunately, NS-3 cannot yet involve any external SDN controller.

Another good simulator and emulation platform is **EstiNet**[19], that supports both OpenFlow and different types of SDN controllers and offers a complete graphical user interface. However, EstiNet is an expensive commercial product and its underlying cloud-service framework, called "Simulation as a Service", needs either a service-clouding hardware platform or a cloud provider, which introduces additional costs and overhead.

A simulation framework for SDN-enabled cloud is **CloudSimSdn** [20], based on **CloudSim**[21]. It is one of the most famous cloud simulators for centralized management of cloud resources by a controller and offers the abstraction of the control plane and data plane. However, experiments are highly demanding in terms of elaboration time and hardware resources.

A SDN emulator which overcomes the cost-problem and introduces the easy management without rivals is **Mininet**[22], where the network is instantiated by the user through simple python scripts. An example Mininet framework to test a SDN network with a POX controller in the cloud has been presented in[23]. However, the main limitation of MiniNet is the scalability, because the emulation load is constrained to the hardware resources of the machine. A hand-crafted version "MiniNet Cluster Edition," which enables the distribution of the network emulation over a cluster of machines, has been proposed, but is still at a prototype-stage.

Finally, an emulation framework that allows the deployment of Hadoop clusters on a SDN is **Doopnet**[3]. It uses Docker containers as the virtual hosts and natively supports SDN in conjunction with external OpenFlow controllers (such as Floodlight, chosen for testing). Doopnet is an interesting framework because it is designed to integrate experimental testbeds in the real-world to evaluate new algorithms. However, it has some limitations: it runs on a single physical machine, limiting the size of the emulated network topology, it requires the installation of Dockernet[24] to manage the docker environment and requires large computing and memory capacity on the single physical machine to simulate a Hadoop complex infrastructure. Even in this case, scalability cannot be easily achieved, i.e., either a manual and sophisticated error-prone procedure is required, or the hardware upgrade.

vSDNEmul[25] was recently presented and represents a relevant solution for SDN emulation. It uses Docker containers to represent the elements in the emulated network. It does not make use a network emulator (like e.g., Mininet), but SDN switches performed as Docker container for an interaction and execution of more realistic scenarios. For the simulation of complex environments, vSDNEmul requires an intensive use of CPU and RAM, which can reach up to + 230% and + 2000% of Mininet requirements, respectively. vSDNEmul represents an emulator of boundary between the physical world (on-premise) and the cloud, because it can integrate the emulation with cloud solutions, such as Kubernetes[26], Swarm[27] and Containerd[28].

4 Host-to-Memory Mapping Methodology

In light of the above description of the Hadoop architecture, it is evident that the Hadoop processes (in particular YARN) require the assignment of specific amounts of physical memory and computational resources in the workers in order to run unbiased emulations. Beyond the possibility of increasing of the CPU and RAM available on the single worker, MaxHadoop guarantees the emulation scalability by allowing to scale out the intensive use of computational and memory resources by increasing the number of workers to distribute the workload.

Concerning the use of computational resources, MaxHadoop offers the virtualization of applications through Docker that allows Hadoop tasks to be distributed evenly over all available core CPUs. In fact, by default, all Docker containers get the same proportion of CPU cycles and, on a multi-core system, the shares of CPU time are distributed over all CPU cores. Theoretically, it is possible to set advanced kernel-level features like the CPU scheduling and prioritization, or assign specific CPUs or cores a container can use. However, incorrect settings of these values can cause the host system to become unstable or unusable. Instead, the default configuration for computing resources provides a perfectly balanced distribution for all containers, which is the best possible configuration because all Hadoop nodes, except the master, have the same requirements in terms of computational resources, as the NM and DN and MR processes. We demonstrate in Sect. 6 that it is possible to use the default parameters without penalizing the performance.

Concerning the physical memory, the default configuration process of Maxinet assigns the memory to the different nodes according to a fair share of physical resources, hence the default settings of Maxinet could introduce an undesired bias in the emulation if the allocated physical memory was not adequate to support the Hadoop processes. In this section we analyse in detail the memory requirements of the relevant Hadoop processes and provide a methodology for the correct sizing of the emulation setup. We also provide a configuration algorithm to spread the MaxHadoop emulation setup over an adequate number of physical machines, guaranteeing the regular operation of a real Hadoop cluster constituted by a variable number of nodes. Obviously, the amount of resources, in terms of RAM and CPU, available in the physical machines shall match overall the requirements imposed by the Hadoop cluster.

The process of Hadoop memory allocation for MapReduce jobs consists of two phases. In the first phase YARN get assigned an amount of memory M_y and a number of CPU cores for each container. In the second phase, the application's resource requests (e.g. MapReduce) are negotiated based on the available resources on each NodeManager of the cluster, then the MapReduce job runs on YARN utilizing its distributed resources to execute its Map and Reduce tasks. Hadoop provides a series of files to manage the memory's and CPU's configuration assigned to the processes. The yarn-site.xml file defines the YARN settings in terms of the following four parameters:

– *yarn.nodemanager.resource.memory-mb* is the amount of physical memory (RAM) that can be allocated to YARN containers on a single node. The default value is **8192 MB**.
– *yarn.scheduler.maximum-allocation-mb* and *yarn.scheduler.minimum-allocation-mb* indicate the maximum and minimum memory allocation allowed to a single container, respectively. The default values are **8192 MB** and **1024 MB**, respectively.
– *yarn.nodemanager.resource.cpu-vcores* is the number of CPU cores that can be allocated to YARN containers. The default value for this parameter is **8**.

The file *mapred-site.xml* defines the MapReduce settings, expressed by the following six parameters:

– *mapreduce.map.memory.mb* and *mapreduce.reduce.memory.mb* are the physical memory values required by the ResourceManager for each Map and Reduce task, respectively; the default values are both **1024 MB**.
– *mapreduce.map.java.opts* and *mapreduce.reduce.java.opts* are the Java Virtual Memory (JVM) heap size for the Map and Reduce tasks, set to **80%** of *mapreduce.map.memory.mb* and *mapreduce.reduce.memory.mb*.
– *mapreduce.map.cpu.vcores* and *mapreduce.reduce.cpu.vcores* are the number of virtual cores to request for each map and reduce task. The default value for both parameters is **1**.

Furthermore, there are other fundamental processes running in a Hadoop cluster; their memory usage values, obtained by launching the "ps" and "jps" commands inside the Dockers containers, are listed in Table 1.

For the correct execution of the jobs, MaxHadoop requires the use of machines with adequate computational capacity, especially the physical memory. As described above, YARN manages the physical resources of the cluster (virtualized by Docker) and assumes default values for all memory-related Hadoop parameters. Unfortunately, if a non-appropriate configuration of the memory-related properties is set, the cluster will start successfully, but won't work properly and efficiently, introducing an undesired bias in the emulation results. In fact, the Docker containers, where Hadoop nodes run, are a process group spawned into independent namespaces by the Linux kernel, which is responsible of the resource scheduling on the hardware using, by default, the virtual memory subsystem. If the containers saturate the

Table 1 Memory usage of Hadoop processes

Node	Process	Memory (MB)
Master	NameNode	350
	SecondaryNameNode	250
	ResourceManager	400
	JobHistoryServer	400
	WebAppProxyServer	200
Total master memory M_{ma}		1600
Slave	DataNode	300
	NodeManager	400
Total slave memory M_{sl}		700

available physical memory, they would use the swap file, allowing the processes to run properly but penalizing execution times. Therefore, it is fundamental to optimise these configuration settings.

Since one Hadoop cluster is usually composed of a single master node and n slave nodes, the total number of nodes, i.e. Docker containers, is $n+1$. Let w be the number of heterogeneous physical workers available in the lab, where MaxHadoop framework has to be installed. Hence, each worker is assigned a number of slaves n_k for $k = 1, \dots, w$, with $n = \sum_{k=1}^{w} n_k$, and one worker is assigned the master node. Without loss of generality we assume that the master is hosted in the *worker 1*. The maximum memory requirement for the entire Hadoop environment M_h is given by:

$$M_h = \sum_{k=1}^{w} \left(M_y + M_{sl}\right) * n_k + M_{ma} \tag{1}$$

where M_y is the amount of physical memory (RAM) that can be allocated to YARN containers on a single node; $M_{ma} = 1600$ MB and $M_{sl} = 700$ MB are the memory (RAM) amounts required by the master node and each slave node, respectively (see Table 1).

Let $M_i = M_{tot}^{(i)} - M_{os}^{(i)}$ be the memory available at the $i-th$ worker, for $i = 1, \dots, w$, where w is the number of workers, $M_{tot}^{(i)}$ is the total RAM installed in the $i-th$ worker, and $M_{os}^{(i)}$ is the memory occupation of the OS and other processes in the $i-th$ worker including, e.g., MaxiNet.[2]

To guarantee good performance to the Hadoop cluster and in particular to YARN, the MaxHadoop emulation setup should dispose of an amount of available memory such that the following condition holds

$$\sum_{i=1}^{w} M_i \geq M_h \tag{2}$$

[2] $M_{os}^{(i)}$ can be obtained with the *free* command.

with the constraints $M_1 = \left(M_y + M_{sl}\right) \times n_1 + M_{ma}$ and $M_i = \left(M_y + M_{sl}\right) \times n_i$ for $i = 2, \dots, w$.

From the above equations we may calculate the maximum number of slaves that can be included in the cluster as given by:

$$n = \left\lfloor \frac{M_1 - M_{ma}}{M_{sl} + M_y} \right\rfloor + \sum_{i=2}^{w} \left\lfloor \frac{M_i}{M_{sl} + M_y} \right\rfloor \tag{3}$$

where $\lfloor \cdot \rfloor$ indicates nearest integer smaller (or equal) than the argument.

The above expressions formalize the memory balance required by the emulation setup to design meaningful experiments in light of their objective and constraints. For instance, solving Eq. (2) in terms of w, we obtain the number of workers with given characteristics required to host a MaxHadoop emulation of a Hadoop cluster of certain dimensions. Solving Eq. (2) in terms of M_y, the memory allocation for the YARN containers is obtained, as constrained to the number and characteristics of the workers, i.e.

$$M_y = \frac{\sum_{i=1}^{w} M_i - M_{ma}}{\sum_{k=1}^{w} n_k} - M_{sl} \tag{4}$$

and so on.

Following the above mathematical calculations, the MaxHadoop emulation setup can be appropriately configured according to the Algorithm 1, where the notations are the same used in Eqs. (1)–(4).

```
i := 0
begin
    while $i < w$ OR $n > 0$ do
        if $i$ != 0 then
            $j \longleftarrow M_i/M_{sl}$
            // number of hosts mappable on the $i$-th worker
            map the hosts from $k$-th to $(k + j)$-th on the $i$-th worker
            $k \longleftarrow k + j$
        else
            map the master node on the $i$-th worker
            $k \longleftarrow (M_i - M_{ma})/M_{sl}$
            // number of hosts mappable on the first worker
            map the first $k$ hosts on the $i$-th worker
            $n \longleftarrow n - i$
            // remaining nodes to map
        link the hosts to the respective switch
        map on the $i$-th worker the switches connected to this set of hosts
        $i \longleftarrow i + 1$
    complete the connection between the switches
```

Algorithm 1: MaxHadoop balanced configuration.

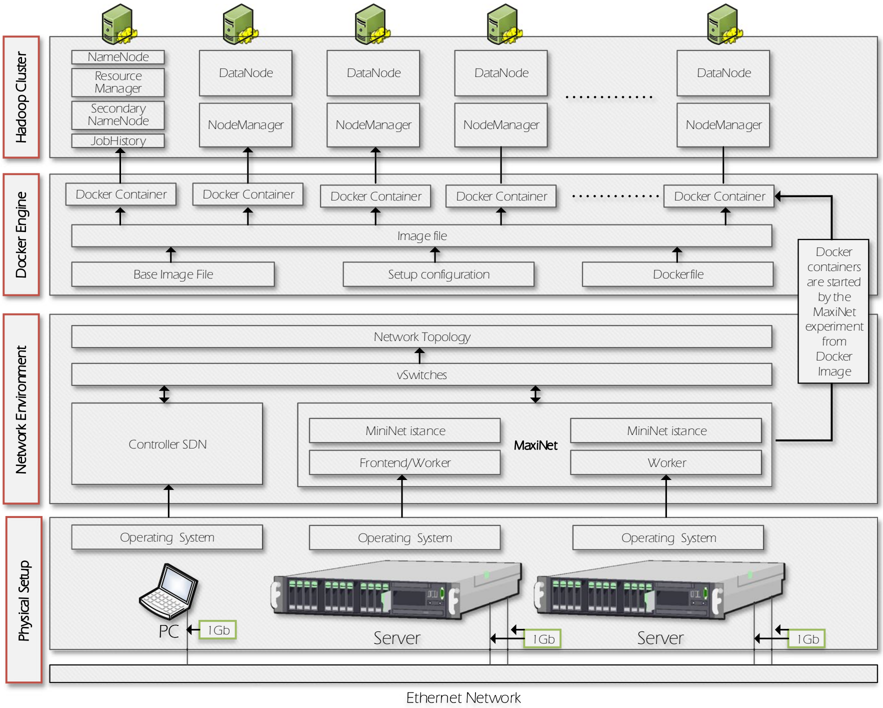

Fig. 2 MaxHadoop architecture

5 The MaxHadoop Emulator

5.1 MaxHadoop Architecture

The MaxHadoop architecture, as shown in Fig. 2, consists in four layers, which are detailed in the following, scrolling the architecture bottom-up.

Layer I: Physical Setup It gathers the workers that form the cluster for the computations and a laptop where the controller SDN is installed. In this layer the workers are connected to each other using the physical Ethernet network connection.

Layer II: Network Environment It corresponds to the emulation of the network environment, i.e., the SDN in MaxiNet. The centralization of network control in the SDN controller allows the SDN controller to implement common network services, such as routing, multicast, access control, bandwidth management, quality of service, optimization of switch workloads, and other policies through the use of APIs. MaxHadoop supports any external SDN controller, which centrally manages the paths of data flows, including the paths between Mappers and Reducers. For our experiments we chose Ryu[29]. It can be implemented on a dedicated hardware device.

The installation of MaxiNet on the Linux machine takes place through a simple script. Maxinet creates topologies and runs virtual hosts, network devices and links, instantiated by simple python scripts that make the physical cluster similar to a real network. The configuration of the environment is set through the MaxiNet configuration file which defines all parameters, including the default OpenFlow controller, and specifies the location of the frontend server and the workers properties.

Layer III: Docker Engine On top of MaxiNet is the Docker environment. MaxHadoop uses Docker containers to emulate the hardware and the software to be evaluated, in place of the default MaxiNet shell-based hosts. Docker is available cross-platform, supports hardware resources to emulate different devices, and is a lightweight option for an isolated environment to execute code. To swap out the MiniNet hosts with a Docker container we use a virtual Ethernet pair. This way, MaxiNet can also run Docker containers instead of the default shell-based hosts. To run Hadoop on emulated hosts, we have used a “Plug and Play” Docker image having Hadoop installed on Ubuntu. We have built the image starting from Hadoop source code version 2.9.0, and compiled inside a container. With this we have created a source Hadoop Docker Image for all cluster nodes, considerably reducing the size of the image compared to the installation from the Hadoop binary file.

Layer IV: Hadoop Cluster It emulates a cluster Hadoop in a “Fully-Distributed” mode[30], where the daemons run on a cluster of machines, composed of Docker containers and configured with a master/slave architecture to distribute and process data between the various nodes. Hadoop master and slave nodes run within different Docker containers, NameNode and ResourceManager run within Hadoop-master container and DataNode and NodeManager run within Hadoop-slave container.

5.2 Emulation Output Parameters

MaxHadoop may monitor many Hadoop metrics. The most common metrics are the amount of transferred data and the time spent by all flows generated during a phase of Shuffle in a MapReduce job, performed on a topology enabled for SDN. To access information about MapReduce flows we use the Hadoop metrics that are published with the REpresentational State Transfer (REST) API. The REST API provides counters, attempts, and configuration information about the jobs and tasks. To complete Hadoop cluster processes, we have enabled the proxy server that is embedded within the ResourceManager service of YARN to access the MapReduce ApplicationMaster REST APIs[31]. The detection of Hadoop metrics via REST API can be done either on-line during the execution of the Job, by issuing queries to the Hadoop Application Master; or off-line at the end of the job, by issuing queries to the History Server of the Hadoop cluster or accessing its logs. In particular, our framework can capture real-time information about running MapReduce tasks and produce an output using the Application Master REST APIs.

Moreover, the Hadoop Job HistoryServer stores data of task-level details that maintains information of MapReduce applications executed over the cluster. Network flows can be captured by using either the SDN REST-API services or the Hadoop metrics/counter related to MapReduce jobs. The SDN REST-API services proactively pull global network traffic information based on any given port number. For instance, data shuffling traffic is captured during the MapReduce Shuffle phase using port number 50010.

The MapReduce framework exposes a number of parameters to track statistics on MapReduce job execution. On-Line detection is applied over a time window closely related to the Job. At the end of the Job, any communication on the sockets is closed. The metrics can be pulled in real time via REST API calls, by polling the ResourceManager while waiting for the Job to run. When a Job is detected in “Running” status it is possibile to obtain, in real time, the following metrics from the ApplicationMaster:

- Task Name;
- Task Type;
- Host and Rack;
- State of the Task;
- Amount of data to be transferred to the Reducer.

These data can be given in input to traffic engineering techniques to optimize the routing, redirect some traffic and reduce the Shuffle phase duration. For the off-line detection of information related to MapReduce job we propose the use of Rumen[32], a framework that mines JobHistory logs to extract relevant data, store them in an easily-parsed, condensed format or digest and to generate statistics from Hadoop jobs useful for performance optimization and simulation.

In MaxHadoop it is also possible to monitor **physical hardware resources**, used by the workers running the experiment, to detect CPU usage, memory utilization and network data rate. This information can be used to detect bottlenecks in the hardware resources used for the testbed or if those resources are underutilized.

Although the MiniNet/MaxiNet network is isolated from the LAN and from the Internet by default, in MaxHadoop we create a specialized host emulated at the frontend which tunnels SSH from the frontend to the emulated hosts. This solution grants the access to the Hadoop cluster from the physical frontend machine, via network, like in a real environment.

A hybrid architecture composed of an environment of physical servers dedicated to Hadoop and a emulator of virtualized nodes on Docker containers, such as MaxHadoop, could provide a very interesting solution. For example, if we detect a saturation of the resources of the physical cluster through the REST API queries, we could activate small virtualized computing centers that are normally dedicated to test environments, to increase the computing capacity. All this assumes proper configuration of IP addressing and virtualized environment name resolution. In MaxHadoop the name resolution has been simplified using the “hosts” file instead of the DNS Server for the configuration.

5.3 MaxHadoop Workflow

The MaxHadoop workflow for a test execution is depicted in Fig. 3. The first step is to create the Docker image to generate Docker containers. In this phase we set the number of hosts and other information useful for the setup (daemons, memory, rack awareness, web interfaces, numbers of hosts, etc.), through the scripts. It is possible to set an arbitrary number of hosts by correctly configuring the network environment for the MaxiNet experiment. Every change in the number of hosts involves a new compilation of the Docker image, i.e., the Hadoop environment configuration file and the file for resolving node names have to be edited and updated.

The next step is the setup of the routing protocols and the start of the SDN controller. With the MaxiNet experiment setup we create a Python script that defines the Docker containers for the Hadoop cluster and the network topology in detail, with a number of OpenFlow switches and links to the same switches and hosts. After the execution of the Python script, the experiment is started with the network environment and the Hadoop cluster, virtualized within Docker containers. So, virtual switches and hosts are mapped over one or more workers. At this point, it is no longer possible to change the cluster settings and topology. Mapping and hostname mapping are passed to the constructor of the experiment class to generate an instance and then it goes to the Command Line Interface (CLI). The role of nodes within the cluster is determined by the IP addresses and hostnames dynamically assigned by

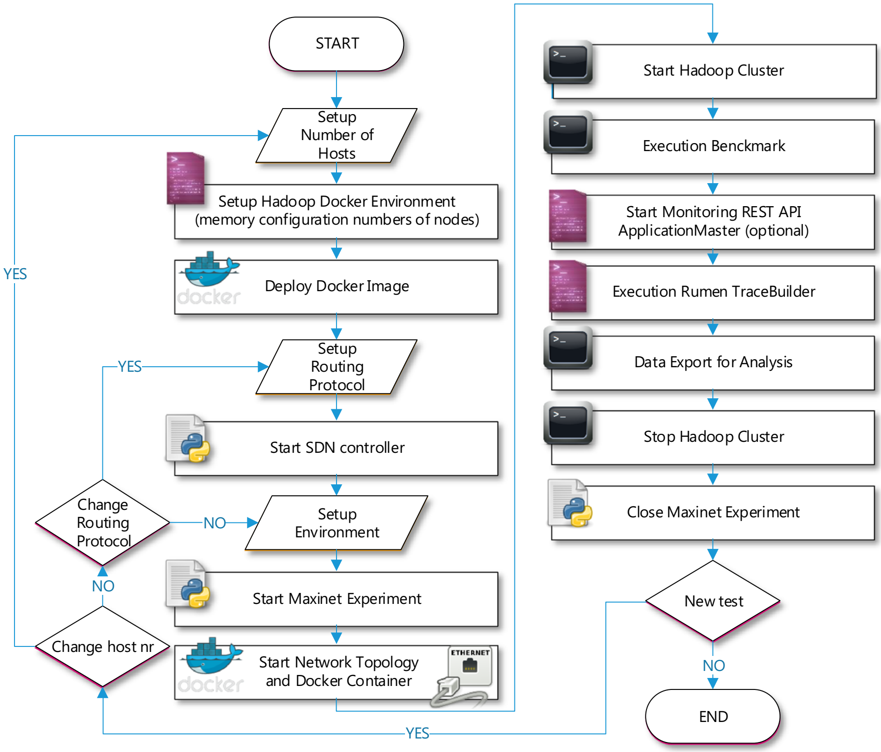

Fig. 3 The MaxHadoop workflow

the script. The use of Docker containers is simple. The MaxiNet class, for importing Docker machines, is invoked by the Python script that specifies the name of the Docker Image, the IP address, and the Hostname of the Docker container. It is also possible to specify limitations on the resources used (CPU and memory) through additional parameters. At this point it is possible to launch the benchmark and collect the information through TraceBuilder. When the benchmark is done, task-level details are extracted by Rumen from Hadoop JobHistory, such as the Map tasks, Reduce tasks, and the nodes where they are executed. Information about Map tasks and the amount of Shuffle data transferred to the Reducer nodes is aggregated.

6 Validation and Tests

We validated MaxHadoop through a series of experiments. In the following, we review the settings of the experimental setup and describe the validation tests. Finally, we discuss the presented results.

Table 2 Configurations of the experimental environment

	Doopnet	MaxHadoop	
Physical machines	One HP server (DL380-G7) with 96 GB memory, 24 Intel Xeon CPU cores and 1TB hard drive	One Fujitsu Primergy server (RX300 S5) with 72 GB memory, 8 Intel Xeon CPU cores and 900 GB hard drive	Two Fujitsu Primergy servers (RX300 S5) with a total of 144 GB memory, 16 Intel Xeon CPU cores and 1.8 TB hard drive, with 2 Gbps network connection
Operating system	Ubuntu 14.04.03 LTS	Ubuntu 16.04 LTS	
Hadoop version	2.7.1	2.9.0	
Docker version	1.9.1	18.03.1-ce	
Controller SDN	FloodLight	Ryu	
Link speed	100 Mbps		
Benchmark	Wordcount: the input files are IETF RFCs (429 MB)		
Reducers number	4		

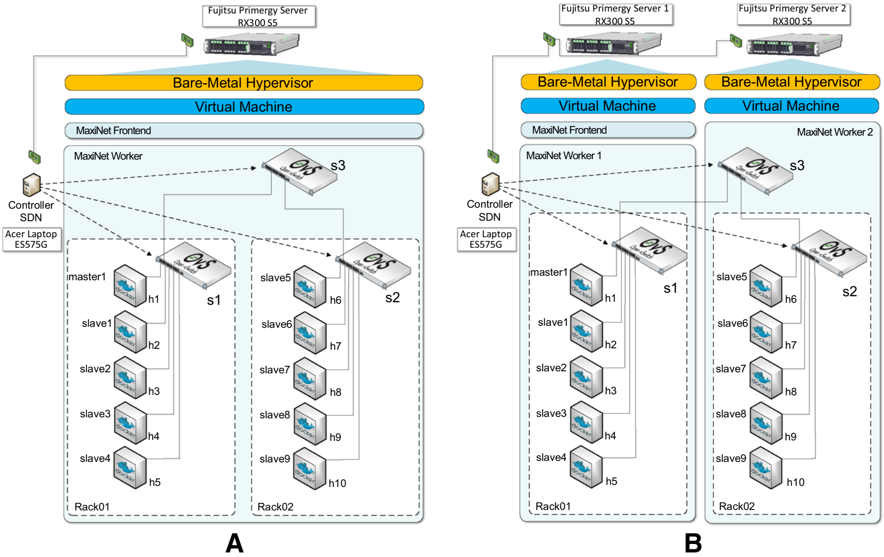

Fig. 4 MaxHadoop validation tests setup: **a** one worker; **b** two workers

6.1 Setup Settings

MaxHadoop is conceived to share the emulation load over multiple workers, whose number can be extended according to the extent of the network to be emulated and the characteristics of the physical machines used. To avoid bottlenecks, two identical workers have been used here. The experimental setup is described in Table 2. The SDN controller is hosted in a laptop Acer ES575G with 12 GB memory, 1 Intel Core i7-7500U 2, 70 GHz and 1 TB hard drive. OpenFlow 1.3 is adopted for the interaction between the SDN controller and the switches. The network topology is emulated on the workers, through MaxiNet, utilizing virtual machines on top of VMware ESXi 6.5 Hypervisor. This hypervisor is not really necessary for the experiment, but it greatly simplifies the portability of the entire environment on different hardware without changing the configuration and parameters of the emulation. Typical experienced CPU overhead for a general-purpose server workload on an ESXi Hypervisor is around 1–5%, with 5–10% memory overhead. With 64-bit CPUs that support the most recent CPU hardware virtualization extensions, it is possible to reduce the overhead to 1% [33]. Hence, the total overload is considered wholly negligible in relation to the advantages obtained.

The setup settings of MaxHadoop are provided manually through a python script based on Algorithm 1. The emulated cluster consists of 10 hosts organized into two racks with links at 100 Mbps as depicted in Fig. 4 for either one or two workers.

The memory that YARN and the MapReduce applications can use in the workers and the load's distribution over them is set according to the Eq. (2) The configuration with one worker, with $M_i = 73728$ MB, $M_{os} = 1700$ MB, $M_y = 6114$ MB and a number of slaves $n = 9$ and one master, satisfies Eq. (2),

Table 3 Memory settings

Parameter	Doopnet/ MaxHadoop	MaxHadoop max memory tests
hadoop-env.sh		
HADOOP_HEAPSIZE	500	500
HADOOP_NAMENODE_INIT_HEAPSIZE	500	500
yarn-env.sh		
YARN_HEAPSIZE	500	500
YARN_RESOURCEMANAGER_HEAPSIZE	500	500
YARN_NODEMANAGER_HEAPSIZE	500	500
YARN_TIMELINESERVER_HEAPSIZE	500	500
mapred-env.sh		
HADOOP_JOB_HISTORYSERVER_HEAPSIZE	250	250
yarn-site.xml		
yarn.nodemanager.resource.memory-mb	6114	12288
yarn.nodemanager.resource.memory-mb	6114	12288
yarn.scheduler.minimum-allocation-mb	256	1024
yarn.scheduler.maximum-allocation-mb	6114	6114
mapred-site.xml		
mapreduce.map.memory.mb	1024	2048
mapreduce.reduce.memory.mb	1024	4096
mapreduce.map.java.opts	-xmx819m	-xmx1638m
mapreduce.reduce.java.opts	-xmx819m	-xmx3277m

which gives 72280 MB ≥ 62926 MB. In the case of two workers, the Eq. (4) gives that each YARN node can be assigned M_i = 15128 MB. We set the Hadoop and YARN parameters as presented in Table 3, to make a fair assessment of scalability of MaxHadoop by doubling the computational capacity and the memory. Other settings are dictated by the specificity of the validation tests, which are organized as follows:

- **Benchmarking Tests**: we performed CPU-intensive and memory-intensive tests, with one or two workers, using *WordCount* (WC) and *TeraSort* (TS). These tests are intended to show the versatility of our emulator and to measure the use of physical resources to assess possible bottlenecks.
- **Comparison with a Similar Emulator**: we replicated in MaxHadoop the same scenarios (including the memory configuration) that have been used for the tests in Doopnet[3] to compare the execution time, using either one or two workers. This test is intended to validate the scalability and to demonstrate that the use of machines with lower performance does not lead to an undesirable increase of the execution time. We use both the above-mentioned benchmarking tools, and make multiple experiments with two workers by increasing

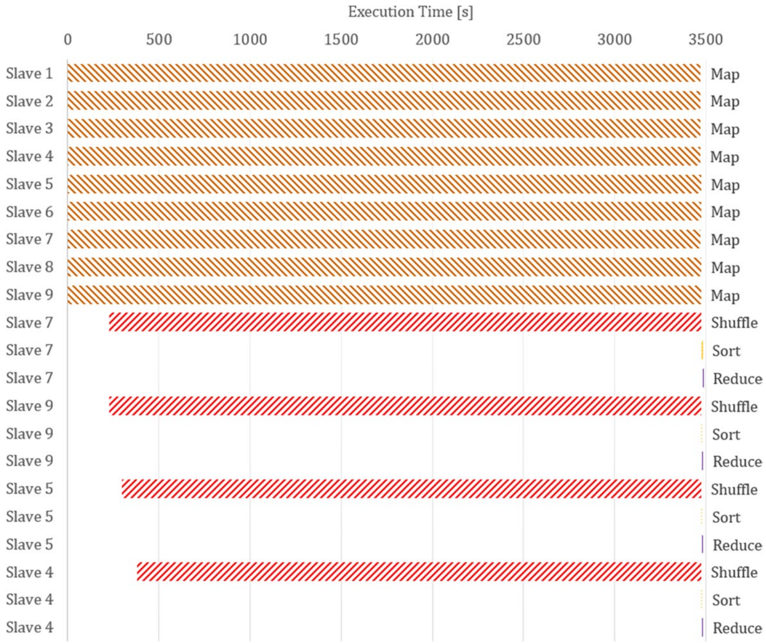

Fig. 5 Execution time in the WordCount test with two workers running a MapReduce job

either the network connection bandwidth or the memory assigned to Hadoop services. These tests are intended to show the effects of emulation settings on the execution time.

6.2 Benchmarking Tests

In the WC benchmark test with 2 workers, data processing is performed on the Mapper nodes that hold the input block to be processed, according to the Hadoop “data locality” and “rack awareness” and only the final result is transferred to the Reducer which utilizes few resources. This minimizes the use of network bandwidth between workers by transferring blocks inside a rack. As shown in Fig. 5, all the Map tasks commit most of the execution time and the Sort and Reduce phases are negligible, i.e. the use of physical memory during the test run is minimal. Indeed, the use of resources during the WC benchmark test is shown in Fig. 6.

Figure 6a evidences that the WC benchmark is totally CPU intensive: the use of the CPU of the two workers is 100% for the whole duration of the test. This means that our choice of setting no constraints to limit a given Docker container’s access to the host machine’s CPU cycles leads to an optimal and full usage of the CPU resources according to the default settings.

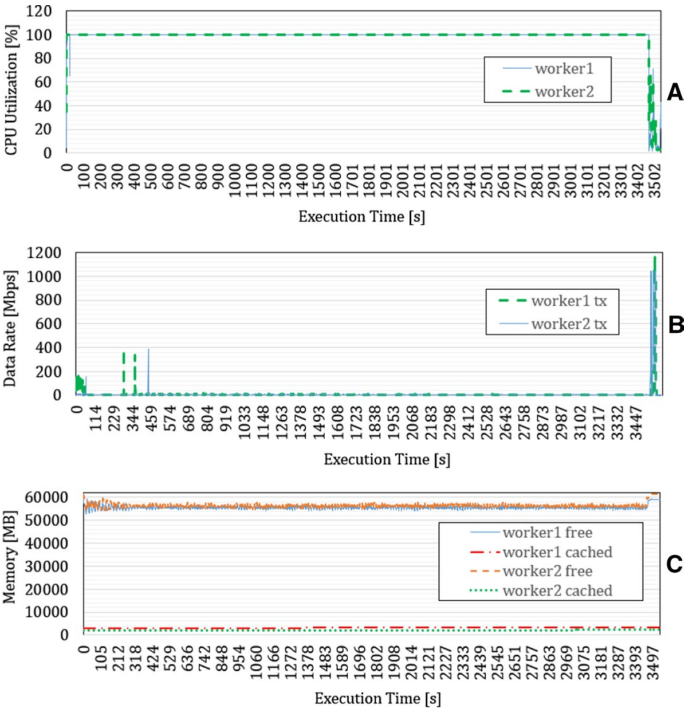

Fig. 6 WordCount test with two workers: **a** CPU utilization; **b** Network connection utilization; **c** Memory usage

The data transfer between the two workers is low, as shown in Fig. 6b. The memory used by the two workers is shown in Fig. 6c, where we distinguish between the *free memory*, that can be used by YARN containers, and the *cached memory* primarily used by HDFS. The free memory has a low usage. Buffered memory used for disk write operations was constant 0 and was not included in the Fig. 6c.

TeraSort exhibits a different behavior than the WC benchmark in the use of physical resources. We may identify two phases: the initial one is CPU and memory-intensive, whereas the second is only memory-intensive. The first phase corresponds to the Map Task and the second one to the Reduce Task of the MapReduce job, as shown in Fig. 7. This is evident in Fig. 8a that shows the use of the CPU in the two workers. The data transfer from the *worker 1* to *worker 2* shown in Fig. 8b is due to the Shuffle phase that sends the HDFS blocks from the Mappers to the sole Reducer (slave 6) to process the final result through the following phases of Sort and Reduce. Figure 8c shows the use of RAM by the workers. We note that during the Reduce phase in worker 2 there is an intense use of free memory by the slave 6 and the cached memory by HDFS. The bottlenecks are created by the CPU-intensive usage in the initial phase and by the network saturation during the transfer of HDFS blocks in the Shuffle phase.

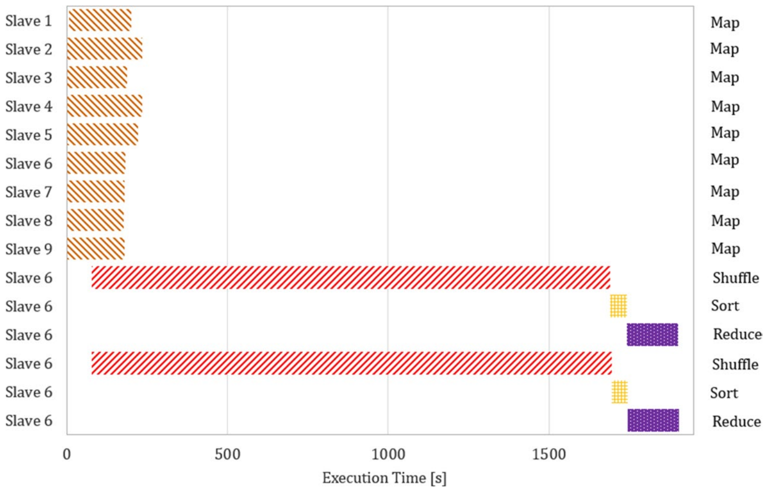

Fig. 7 Execution time of MapReduce job over two workers for TeraSort benchmark with an input data set of 20 GB

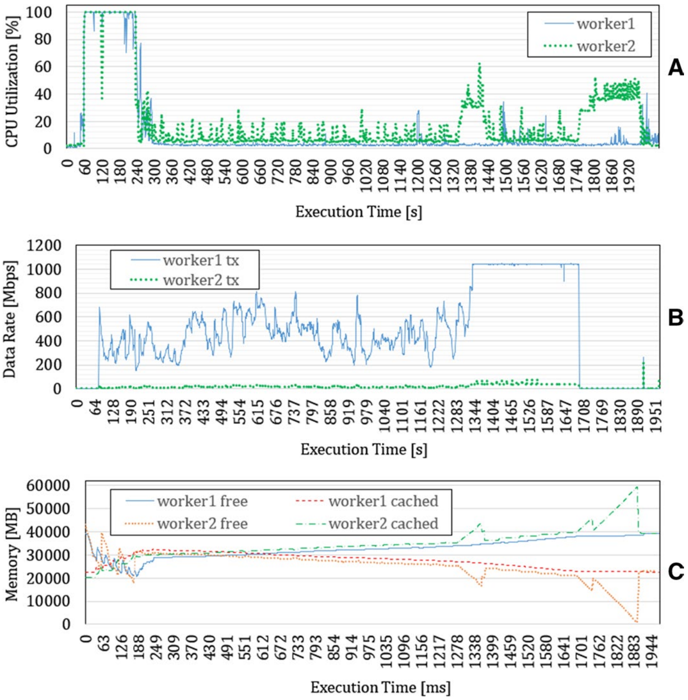

Fig. 8 TeraSort test with two workers: **a** CPU utilization; **b** Network connection utilization; **c** Memory usage

Table 4 Comparison of the execution times of WordCount and TeraSort benchmarking tests obtained by the emulation of the same scenarios in Doopnet and MaxHadoop

Test	Benchmark	Workers	Link speed [Mbps]	Memory [MB]	QoS	Background traffic	Doopnet	MaxHadoop
1	WC	1	100	6114	No	No	5675 s	6749 s
2	WC	2	100	6114	No	No	–	3485 s
3	WC	1	100	6114	No	Yes	7542 s	7917 s
4	WC	2	100	6114	No	Yes	–	4132 s
5	WC	1	100	6114	Yes	Yes	6609 s	6871 s
6	WC	2	100	6114	Yes	Yes	–	3523 s
7	TS	2	100	6114	No	No	–	1905 s
8	TS	2	100	6114	No	Yes	–	2948 s
9	TS	2	100	6114	Yes	Yes	–	2309 s
10	WC	2	1000	6114	No	No	–	3454 s
11	TS	2	1000	6114	No	No	–	567 s
12	WC	2	100	12288	No	No	–	3418 s
13	TS	2	100	12288	No	No	–	1295 s

In MaxHadoop additional results are obtained by varying the link capacity and the memory settings

6.3 Comparison with a Similar Emulator

We launched the benchmarks WC and TS in three different scenarios: (a) in the presence of Hadoop traffic only, (b) in the presence of Hadoop traffic and background traffic generated by *iperf* tool with *iperf server* on the odd Docker containers and *iperf clients* on the even Docker containers, (c) Hadoop traffic and background traffic with a mechanism for the QoS enforcement in the switches to schedule the iperf traffic to the low priority queue. This policy grants to the Hadoop traffic the 99% of the link capacity while competing with the background traffic. Openflow 1.3 is utilized for the interaction with L2 switches running Open vSwitch 1.10. Traffic prioritization is implemented utilizing *min-rate* and *max-rate* properties.

The comparison of the benchmarks execution times in Doopnet and MaxHadoop with different configurations is summarized in Table 4 for the thirteen tests performed. The execution time of MaxHadoop for Test 1 using one worker and WC is higher than that of Doopnet by 19%, whereas it is 39% lower in Test 2 (two workers), i.e. doubling the computing power brings an improvement of 48%.

Higher execution times are experienced in the tests with WC with background traffic and no QoS (Tests 3 and 4) since iperf shares with Hadoop both the network bandwidth and host's computational power. MaxHadoop with one worker (Test 3) runs the test in 7.917 seconds, 5% more than Doopnet and in 4.123 seconds with two workers (Test 4), 45% less than Doopnet. The improvement of MaxHadoop with 2 workers compared to the use of only one, is equal again to 48%. Introducing also the QoS (Tests 5 and 6), the bandwidth for iperf is limited to 1% of the link, and the execution time of MaxHadoop on one worker is 4% higher than Doopnet, and drops by 47% with two workers. The improvement with two workers is 49%. Actually, the WC benchmark run is CPU-intensive throughout its lifetime. By increasing the number of

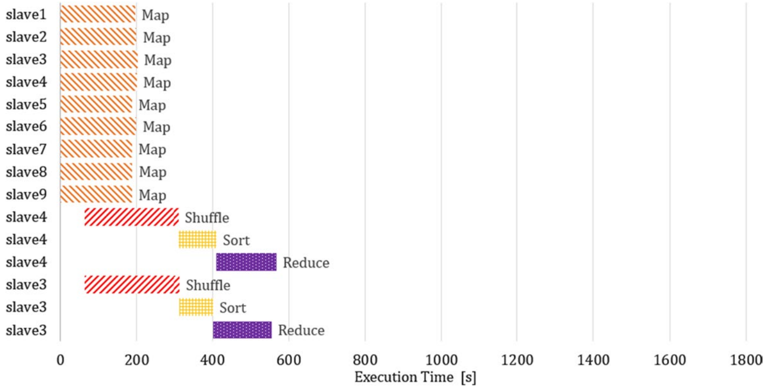

Fig. 9 Execution time of MapReduce job with the network connection speed at 1Gbps

workers from one to two, we observe a decrease in the job execution time of 48-49% because it doubles the computing power (from 8 to 16 CPUs available for calculation).

In general, we may conclude that MaxHadoop on two workers offers improved performance with respect to Doopnet simulator, although less CPU cores are available for MaxHadoop, i.e., 16 CPUs overall on two workers, against the 24 CPU for Doopnet.

The same tests have been done with the TS benchmark. In TS it is possible to select the amount of data to be ordered for the tests; we have used a data set of 20 GB. The TS benchmark execution times are also reported in Table 4. TS benchmark takes 1.905 seconds in the presence of Hadoop traffic only (Test 7). Adding the background traffic generated by iperf tools, the execution time rises to 2948 seconds (+ 55%) (Test 8). Introducing the QoS queueing mechanism (Test 9), the task execution time is 2309 seconds, 21% higher than in the case of Hadoop traffic only and 22% lower than in the presence of background traffic. Obviously, this reduction of the MapReduce job execution time is due to the Hadoop traffic prioritization with respect to the other traffic generated by *iperf*. Increasing the network connection bandwidth in MaxHadoop from 100 Mbps to 1 Gbps (Test 11) we obtain the execution times shown in Fig. 9 for the TS benchmark on two workers. A significant reduction of the execution times of the MapReduce job is observed with respect to the results shown in Fig. 7. The execution times of WC and TS with two workers with a link speed of 1000 Mbps are reported in Table 4, i.e., Tests 10 and 11. For the WC benchmark the execution time reduction is 1% compared to Test 1. The improvement is negligible because that job uses mainly the CPU processing power. Test results change for the TeraSort benchmark with a job execution time in Test 11 of 70% lower than in test 7. The increase of link speed leads to a considerable reduction in the time required to execute the shuffle phase (Fig. 9) responsible for the transfer of data from the Mappers to the Reducers (slave4 and slave3). Tests 12 and 13 present the performance of MaxHadoop when the maximum available memory capacity is assigned to Hadoop services and YARN for the WC and TS benchmarks, respectively. The time reduction for WC (Test 12) is 2% lower than Test 1 and for TS (Test 13) is 32% lower than Test 7. The increase in the available memory involves

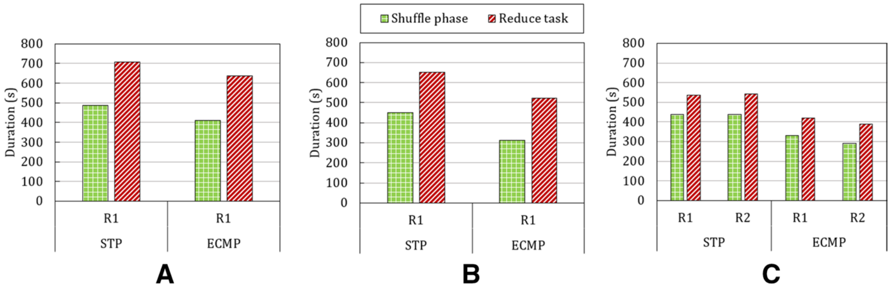

Fig. 10 Terasort execution time. **a** 1 reducer, 8 hosts; **b** 1 reducer, 16 hosts; **c** 2 reducers, 8 hosts

the use of multiple YARN containers, and therefore additional hardware resources to run applications or, as in this case, the MapReduce job of TeraSort.

6.4 Network Protocols Comparison Use Case

In this section we report an emulation's outcome obtained utilizing MaxHadoop emulation environment. Starting from the setup described in Sect. 6.1, we utilize MaxHadoop to perform a comparison of Hadoop perfomance utilizing different SDN-based routing strategies in a Fat-Tree topology.

We compare the performance of the SDN implementation of two well-known network protocols: Spanning Tree Protocol (STP) and Equal Cost Multi Path (ECMP). STP is designed to reduce a mesh topology to a single spanning tree eliminating all loops in the topology. Furthermore, in case of a node failure or breakdown in data path, it will reconfigure automatically the spanning tree to provide a service of fault tolerance. ECMP, instead, allows the choice of multiple next hops with equal cost. The ECMP spreads the traffic loads with the implementation of multiple equal cost shortest paths. This protocol, through the use of a hash function, spreads the forwarding traffic toward a destination on a set of forwarding ports with the same cost, rather than implementing a spanning tree for each destination.

The SDN control is implemented utilizing Ryu Controller and OpenFlow 1.3. Networkx Python Library is utilized to create a graph representation of the network and to calculate all shortest paths between pairs of nodes.

We compare the performance of the two proposed routing algorithms by measuring the time taken by the Reducer task and the Shuffle phase to complete the Terasort benchmark operation. The reduce task time includes the shuffle phase as well as sort and reduce operations time. We evaluate the performance by varying the number of Hadoop hosts in the network, the number of Reducer nodes. The amount of data to be ordered by TeraSort is set to 10GB. It is worth to recall that the Shuffle phase strongly loads the network because it includes the data transfer from "mapper nodes" to the "reduce node."

Figure 10 shows the performance of the two network protocols in different scenarios. In a scenario with 1 reducer node and 8 Hadoop hosts, the Shuffle phase reduces the execution time by 15% with ECMP, compared to the same phase with STP (Fig. 10a). One can exploit the configurability of MaxHadoop by increasing the number of Hadoop hosts. As shown in Fig. 10b, ECMP reduces by 32% the execution time of shuffle phase with respect to STP. In this case, ECMP outperforms STP protocol due to the higher number of flows controlled by the SDN controller. Finally, in Fig. 10c we consider the scenario with 8 hosts and 2 reducers. In this case the execution time reduction achieved with ECMP is 36%. The significant difference of performance between ECMP and STP evident in Fig. 10c is explainable by the better capability of ECMP to manage the higher number of flows generated with 2 reducers.

7 Conclusion

This document proposes MaxHadoop, a emulation framework to emulate Hadoop environments on SDN networks using commodity hardware. The strengths are: *the scalability of the environment* with the number of physical machines (workers) composing the setup, that allows the simulation of large clusters with many nodes and complex data center infrastructure; *the ease of network environment setup*, because it is possible to emulate switches and routers according to different topologies with the same simplicity of MiniNet. With the same simplicity, it is possible to start Docker containers from previously archived images.

This tool will allow researchers to work on emulated data center infrastructures where algorithms and protocols can be tested easily and at low cost. Also, it is possible to test new network protocols or variants of existing ones and to test Hadoop ecosystem solutions, like e.g. Pig, Hive, Spark, etc. We validated the performance of MaxHadoop using two typical benchmarks. The test results show that the performance of Hadoop can be improved by increasing memory allocated to nodes when memory is the bottleneck or by increasing the speed of network links. We have also seen that using QoS, centrally managed by the SDN controller, can improve Hadoop's performance. From the same controller it is possible to manage the entire network and gather information to dynamically change the configuration to improve the performance of the hosted environments.

Finally, we provided a performance comparison of different SDN-based network protocol implementation, in order to show the potentiality of our emulator. In this direction we will work in the future. Since the Shuffle phase in Hadoop uses the network very intensively, we expect that the SDN paradigm can guarantee the adequate flexibility in network configuration by providing sufficient redundancy and appropriate alternative paths in case of network congestion due to the flows from Mappers to Reducers, leading to a reduction of the execution time of the Shuffle phase. To do this, we will use the REST API queries to the Hadoop environment in real time to provide the knowledge of the Jobs status to the SDN controller, which, along with the knowledge of the network topology and the employment status, can dynamically manage the reallocation of resources to optimize the MapReduce job flows. In a future work we will also evaluate the execution of the same tests performed in this document with the Spark platform, perhaps using a cloud solution such as

Kubernetes in order to perform a large-scale simulation with an SDN network distributed on-premise and in cloud.

Acknowledgment Open access funding provided by Università degli Studi dell'Aquila within the CRUI-CARE Agreement.

References

1. Elalaoui, A.E., Benzekki, K., Fergougui, A.E.: Software-defined networking (SDN): a survey. Secur. Commun. Netw. **9**, 5803–5833 (2016)
2. Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J., McKeown, N., Anderson, T.: Openflow: enabling innovation in campus networks. Int. J. Parallel Programm. **38**, 69–74 (2008)
3. Fang, G., Lee, B., Qiao, Y., Wang, X.: Doopnet: an emulator for network performance analysis of Hadoop clusters using Docker and Mininet. In: IEEE Symposium on Computers and Communication (ISCC), pp. 784–790, (2016)
4. Schwabe, A., Wallaschek, F., Zahraee, MH., Karl, H., Wette, P., Dräxler, M.: MaxiNet: Distributed emulation of software-defined networks. In: 2014 IFIP Networking Conference, pp. 1–9, (2014)
5. Lebiednik, Brian., Mangal, Aman., Tiwari, Niharika.: A survey and evaluation of data center network topologies. arXiv preprint arXiv:1605.01701. (2016)
6. Vahdat, A., Al-Fares, M., Loukissas, A.: A scalable, commodity data center network architecture. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication, pp. 63–74, (2008)
7. Calcaterra, C., Carmenini, A., Marotta, A., Cassioli, D.: Hadoop performance evaluation in software defined data center networks. In: Proc. of the 2019 International Conference on Computing, Networking and Communications (ICNC), pp. 18–21, (2019)
8. Docker Inc. Docker, (June 2018). https://www.docker.com/index.html
9. Peuster, M.: Containernet home page. https://containernet.github.io/ (2018)
10. Yao, B., Guo, M., Li, Z., Shen, Y.: Ofscheduler: a dynamic network optimizer for MapReduce in heterogeneous cluster. Int J Parallel Programm. **43**, 472–488 (2015)
11. Ferguson, Andrew D., Guha, Arjun., Liang, Chen., Fonseca, Rodrigo., Krishnamurthi, Shriram.: Participatory networking: An API for application control of SDNs. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, SIGCOMM '13, pp. 327–338, New York, NY, USA, Association for Computing Machinery. https://doi.org/10.1145/2486001.2486003. (2013)
12. Wang G., Ng. TSE, Shaikh, A.: Programming your network at run-time for big data applications. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, pp. 103–108, (2012)
13. Abbasi, A.A., Abbasi, A., Shamshirband, S., Chronopoulos, A.T., Persico, V., Pescapè, A.: Software-defined cloud computing: a systematic review on latest trends and developments. IEEE Access **7**, 93294–93314 (2019)
14. VMware. VMware Tanzu Kubernetes Grid. https://tanzu.vmware.com/kubernetes-grid. Accessed 18 Apr 2020
15. Colbert, K.: Introducing vSphere 7: Modern Applications and Kubernetes. https://blogs.vmware.com/vsphere/2020/03/vsphere-7-kubernetes-tanzu.html. Accessed 18 Apr 2020

16. Lai, J., Tian, J., Zhang, K., Yang, Z., Dingde, J.: Towards a cloud-based network emulation platform. In: Mobile Networks and Applications, Network Emulation as a Service (NEAAS) (2020)
17. Zulu, L.L., Ogudo, K.A., Umenne, P.O.: Emulating software defined network using mininet and opendaylight controller hosted on amazon web services cloud platform to demonstrate a realistic programmable network. In 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), pp. 1–7, (2018)
18. Henderson, T.R., Lacage, M., Riley, G.F., Dowell, C., Kopena, J.: Network simulations with the ns-3 simulator. SIGCOMM Demonstrat. **14**(14), 527 (2008)
19. Yang, C., Wang, S., Chou, C.: Estinet openflow network simulator and emulator. EEE Commun. Mag. **51**, 110–117 (2013)
20. Calheiros, RN., Ji, X., Yoon, Y., Buyya, R., Son, J., Dastjerdi, AV.: Cloudsimsdn: Modeling and simulation of software defined cloud data centers. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 475–484, (2015)
21. Beloglazov, A., Rose, C.A.F.D., Buyya, R., Calheiros, R.N., Ranjan, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resourceprovisioning algorithms. Softw Pract Exp **41**, 23–50 (2011)
22. McKeown, N., Lantz, B., Heller, B.: A network in a laptop: rapid prototyping for software defined networks. In: Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, pp. 1–6, (2010)
23. Adami, D., Chiaro, AD., Giordano, S., Santos, A., Teixeira, J., Antichi, G.: Datacenter in a box: test your SDN cloud datacenter controller at home. In: 2013 Second European Workshop on Software Defined Networks, pp. 99–101, (2013)
24. Zhu, R.: Dockernet source code. https://github.com/vfreex/dockernet
25. Farias, Fernando N. N., Junior, Antônio de O., da Costa, Leonardo B., Pinheiro, Billy A., Abelém, Antônio J. G.: vsdnemul: A software-defined network emulator based on container virtualization, (2019)
26. Cloud Native Computing Foundation. kubernetes. https://kubernetes.io. Accessed 18 Apr 2020
27. Docker. Swarm. https://docs.docker.com/engine/swarm. Accessed 18 Apr 2020.
28. Cloud Native Computing Foundation. containerd. https://containerd.io. Accessed 18 Apr 2020
29. Ryu SDN Framework. Ryu SDN Framework Community. https://osrg.github.io/ryu/. (2017)
30. Apache Software Foundation. Apache hadoop 2.9.0 hadoop cluster setup. https://hadoop.apache.org/docs/r2.9.0/hadoop-project-dist/hadoop-common/ClusterSetup.html. (2017)
31. Apache Software Foundation. Apache hadoop 2.9.0—mapreduce application master rest APIs. https://hadoop.apache.org/docs/r2.9.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredAppMasterRest.html. (2017)
32. Apache Software Foundation. Rumen. http://hadoop.apache.org/docs/r1.2.1/rumen.html. (2018)
33. Li, Z., Kihl, M., Lu, Q., Andersson, J. A.: Performance overhead comparison between hypervisor and container based virtualization. In: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), pp. 955–962, (2017)

Claudio Calcaterra is a system administrator and solution architect in a multisite military datacenter in geographic replication. He has expertise on Software Defined Data Center (SDDC) architectures on converged and hyper-converged Vmware infrastructures. He received his PhD in Information and Communication Technology from the University "G.d'Annunzio" Chieti-Pescara, in 2019, with studies related to Big Data application on simulated SDN. He is a member of the Unit of Molecular Neurology at CeSI-MeT, Center of Excellence of Studies on Aging and Translational Medicine of the University of Chieti-Pescara.

Alessio Carmenini is student of the master degree in Telecommunication Engineering at University of L'Aquila. In the 2018 he received his Bachelor degree in Computer Science from the University of L'Aquila. From 2018 until 2020 he worked as research assistant at University of L'Aquila performing research on SDN and Cyber-security.

Andrea Marotta is research fellow at the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila, Italy. He received his M.Sc. degree in Computer Engineering and his Ph.D. in Information and Communications Technology from the University of L'Aquila, in 2015 and 2019, respectively. During his graduate studies, he was visiting student at Group for Research on Wireless (GROW) at Instituto Superior Técnico/University of Lisbon, performing research on C-RAN deployment optimisation. In 2017 he spent one year as visiting Ph.D. at the research Institute of Communication, Information and Perception Technologies (TeCIP) of Scuola superiore di studi universitari e di perfezionamento Sant'Anna in Pisa performing research on 5G Radio Access Network design. In 2019 he spent a research period at the Computer Networks Lab (NetLab) of the University of California, Davis, performing research on slice reliability. He actively participated in the TPC of IEEE conferences, such as IEEE 5G World Forum (WF-5G) and IEEE International Forum on Research and Technologies for Society and Industry (RTSI). He performs research on Mobile Networks Reliability, 5G Optical-Wireless Convergence, 5G Software Defined Access, CoMP Coordinated Scheduling.

Ubaldo Bucci is a Ph.D student working at the Department of Information of Engineering and Computer Science of the University of l'Aquila, Italy. He received his master degree in Computer science in 2018 at the same university. He is specializing in Network Slicing and Deep Reinforcement Learning. In particular his main interest is the usage of Deep Reinforcement Learning to solve Radio Access Network related problems of optimization or automation.

Dajana Cassioli is *Associate Professor* of Telecommunications Engineering at the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila, Italy. Her main research interests are in the fields of measurement and modeling of wireless channels, wireless and mobile communication technologies, 5G/beyond 5G Networks and Network Security. She is the Chair of the WIE AG of the IEEE Italy Section, and Past Chair of the IEEE Joint VT06/COM19 Italy Chapter. She is the Coordinator of the University of L'Aquila Node of the National Laboratory of Cyber-Security of the National Inter-University Consortium for the Computer Science (CINI). In 2010 and 2016, she has been awarded the *ERC Starting Grant VISION* (Video-oriented UWB-based Intelligent Ubiquitous Sensing) and the *ERC Proof-of-Concept Grant iCARE* (MobIle health-Care system for monitoring toxicity and symptoms in cAncer patients Receiving diseasE-oriented therapy). She is author and co-author of tens of publications on prestigious International Journals and Conferences. She served as the Industry Co-Chair of PIMRC 2018, WIE Chair for RTSI 2018, RTSI 2019, RTSI 2020, MELECON 2020 and 2020 IEEE Int. Workshop for Industry 4.0 and IoT, and as TPC member of several International Conferences, like the ICC, PIMRC, VTC, GLOBECOM. She is an *Associate Editor* of the IET Electronic Letters and IEEE Communications Letters, and an *Executive Editor* of the Internet Technology Letters, John Wiley & Sons Ltd and Transactions on Emerging Telecommunications Technologies, John Wiley & Sons Ltd.