Received XXXX; | Revised XXXX; | Accepted XXXX

DOI: xxx/xxxx

RESEARCH ARTICLE

BigDataSDNSim: A Simulator for Analyzing Big Data Applications in Software-Defined Cloud Data Centers

Khaled Alwasel[1,2] | Rodrigo N. Calheiros[3] | Saurabh Garg[4] | Rajkumar Buyya[5] | Mukaddim Pathan[6] | Dimitrios Georgakopoulos[7] | Rajiv Ranjan[1]

[1]School of Computing, Newcastle University, Newcastle upon Tyne, UK
[2]College of Computing and Informatics, Saudi Electronic University, Riyadh, Saudi Arabia
[3]School of Computer, Data and Mathematical Sciences, Western Sydney University, Sydney, Australia
[4]School of Computing and Information Systems, University of Tasmania, Hobart, Australia
[5]School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia
[6]Telstra Corporation Limited, Melbourne, Australia
[7]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia

Correspondence
Khaled Alwasel
Newcastle University, UK.
Saudi Electronic University, Saudi Arabia.
Email: kalwasel@gmail.com

Summary

The integration and cross-coordination of big data processing and Software-Defined Networking (SDN) are vital for improving the performance of big data applications. Various approaches for combining big data and SDN have been investigated by both industry and academia. However, empirical evaluations of solutions that combine big data processing and SDN are extremely costly and complicated. To address the problem of effective evaluation of solutions that combine big data processing with SDN, we present a new, self-contained simulation tool named BigDataSDNSim that enables the modeling and simulation of the big data management system YARN, its related programming models MapReduce, and SDN-enabled networks in a cloud computing environment. BigDataSDNSim supports cost-effective and easy to conduct experimentation in a controllable, repeatable, and configurable manner. The paper illustrates the simulation accuracy and correctness of BigDataSDNSim by comparing the behavior and results of a real environment that combines big data processing and SDN with an equivalent simulated environment. Finally, the paper presents two uses cases of BigDataSDNSim, which exhibit its practicality and features, illustrate the impact of data replication mechanisms of MapReduce in Hadoop YARN, and show the superiority of SDN over traditional networks to improve the performance of MapReduce applications.

KEYWORDS:
Software-Defined Networking (SDN); big data; MapReduce programming model; modeling and simulation; performance optimization, joint-optimization.

1 | INTRODUCTION

As modern applications need to generate and consume massive amounts of data, traditional database systems struggle to cope with such data demand [1]. The main obstacle for these systems is the requirements of excessive time and super-computing capacity. Big data analytics [1] has emerged as the preferred option to effectively analyze large-scale data sets at an astonishing speed. It harnesses the power of clustering commodity hardware to carry out data analysis in a parallel manner. Big data has become the most prominent mechanism to delve into data sets and provide valuable insights, such as detection of emerging risks and threats, predicting behaviors and patterns, and providing business opportunities [2].

To bring big data analytics into existence, several frameworks have been developed and launched in cloud-based environments, such as Hadoop MapReduce[3] and Apache Storm[4]. These frameworks facilitate the process of utilizing parallel programming models and engines, along with meeting different aspects of big data requirements. For instance, the MapReduce programming model can be used to analyze historical data in a few lines of code, while a stream processing model is used to handle a never-ending data stream at a speed of milliseconds[5]. Nevertheless, a big data application can rely on multiple big data engines to handle different aspects of data analysis simultaneously; therefore, big data management systems (BDMS) such as Apache Hadoop YARN[6] have emerged to coordinate and schedule computing resources among big data engines and applications co-hosted on a shared big data cluster.

One critical issue of big data is that every application faces several performance challenges due to its unique context and requirements of data flow and processing[5,7,8,9]. In particular, every MapReduce application has its characteristics and runs across tens of servers distributed in different data center racks; therefore, every application would most often require unique scheduling mechanisms, subject to the given processing and communication requirements and patterns[10]. MapReduce scheduling plays a vital role in achieving performance goals, reducing execution time, minimizing computing costs, and ensuring proper resource utilization and management[11,12]. Surprisingly, scheduling is not a new phenomenon; it has been studied for decades in different contexts[13]. Thousands of algorithms have been introduced where each one tackles a scheduling problem based on specific factors, constraints, and conditions (FCC)[14,15]. This also applies to MapReduce applications, where slight changes in FCC values can lead to a new scheduling problem and require a novel solution.

A large volume of recently published studies proposed solutions concerning MapReduce scheduling for achieving diverse performance goals[10,16]. The limitations of these studies is that the evaluation of the approaches is most often carried out in small-scale infrastructures because real environments can be impractical and difficult to use for reasons related to cost, time, configuration complexity, and behavior instability. Besides, every MapReduce application is required to be configured independently across computing and network layers, which makes the scheduling and optimization of every application a daunting task to accomplish in testbeds. Finding a feasible solution for a scheduling problem usually requires a massive search of all possible solutions, which involves tens of thousands, if not millions, of possibilities and combinations. Such kind of MapReduce problems is determined to be NP-complete[8,17].

The Software-Defined Networking (SDN)[18] paradigm aims to enable dynamic configuration of networks and to overcome the limitations of traditional network infrastructure. The critical difference between traditional networks and SDN is that the control layer of network devices is moved away to an SDN controller. Such a tactic makes networks more robust, simplified, and flexible to changes as the network is controlled from a central point instead of a complex, distributed control mechanism. The control layer seamlessly enforces SDN customized management and routing policies in the data layer using well-defined application programming interfaces (APIs), such as OpenFlow[19]. Several network vendors (e.g., HP, Pica8, Netgear) have introduced numerous network devices that are SDN-enabled to bring better consumer experience. As a result of the attractive features and world-wide adoption, SDN has gained significant attention from researchers working on improving the performance of diverse applications and systems, including MapReduce applications.

With the advent of SDN, a number of studies have leveraged MapReduce in SDN-enabled environments and proposed novel joint-optimization solutions, which notably enhance MapReduce performance[20,21,22]. Still, several challenges need to be tackled to leverage and evaluate the benefits of SDN for supporting the network capabilities for big data applications in MapReduce contexts. One evaluation approach for SDN-enabled MapReduce new solutions is to use live production environments. However, such environments are hard to obtain and configure along with posing an impediment to the evaluation process due to the continuous changing in behaviour. Another superior evaluation approach is to use a simulation tool[23,24,25], which incorporates the concepts of SDN and big data for conducting large-scale infrastructure experiments in easy, repeatable, controllable, and configurable manner. To fill this gap, we present BigDataSDNSim: a new, discrete-event simulation tool designed to enable the modeling and simulation of big data management systems (YARN), big data programming models (MapReduce), and SDN-enabled networks within cloud computing environments. To the best of our knowledge, BigDataSDNSim is the first tool that models and simulates the three merging technologies (MapReduce-BDMS, SDN, cloud) in a single simulated environment. Based on our proposed system-based and mathematical models, the simulator is capable of capturing the key functions, characteristics, and behaviors of the SDN-enabled MapReduce computing environment. It can also model the functionalities of MapReduce applications in line with mimicking diverse SDN capabilities and interactions with BDMS systems in a seamless manner.

The main contribution of BigDataSDNSim is the ability to generate different samples of SDN-enabled MapReduce-BDMS infrastructures, along with to reduce the complexity of deploying new solutions. The simulator can predict and quantify the impact of new MapReduce SDN-based solutions and designs running in cloud environments. *The accuracy and correctness of*

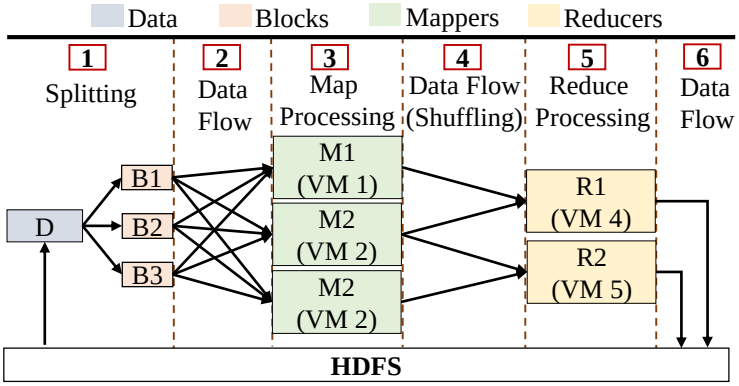


FIGURE 1 An overview model of MapReduce processing and data flow phases

BigDataSDNSim are validated by comparing the behavior and results of a real environment that combines MapReduce and SDN with an equivalent simulated environment given by BigDataSDNSim. Evaluation results demonstrate that BigDataSDNSim is closely comparable with real environments. To gain insight of the features and functionalities of BigDataSDNSim, this paper present two use cases that demonstrate the ability of SDN to improve the performance of MapReduce applications as compared to traditional networks and illustrate the impact of Hadoop Distributed File System (HDFS) replication mechanisms in the overall MapReduce performance. **In summary, the main contributions of this paper are as follows:**

- A new holistic simulation framework that simulates MapReduce applications, BDMS, and SDN-related networks in a cloud-based environment
- Theoretical modeling of MapReduce processing in SDN-aware cloud data centers
- Multiple system models to simulate different samples of MapReduce running in SDN-powered cloud infrastructures
- Validation and comparison of the accuracy and correctness of BigDataSDNSim with a real-world MapReduce SDN-enabled environment

The rest of this paper is organized as follows: Section 2 provides an overview of a MapReduce-BDMS architecture running in an SDN-enabled cloud data center, which is used as a baseline for modeling BigDataSDNSim. Section 3 presents related work and reflects the importance and unique capabilities of BigDataSDNSim. Section 4 elaborates the architecture and design of BigDataSDNSim framework in detail. Section 5 demonstrates the modeling of BigDataSDNSim mathematically and descriptively. Section 6 illustrates the validation of BigDataSDNSim by comparing the results of an identical simulation model given by BigDataSDNSim with the obtained results of a real MapReduce SDN-enabled environment. Section 7 presents two use cases of BigDataSDNSim, which exhibit its practicality and features, illustrate the impact of data replication mechanisms of MapReduce in Hadoop YARN, and show the superiority of SDN over traditional networks in terms of optimizing MapReduce applications. Section 8 concludes the paper and highlights some future work.

2 | OVERVIEW

MapReduce is a programming model that runs in the form of a big data application[3]. Apache Hadoop YARN[6] is considered to be the most dominant framework for building MapReduce applications. A fundamental part of the YARN is HDFS, which is a distributed file storage that distributes data across virtual machines (VMs) in a given big data cluster. It supports a replication mechanism of data sets for fault tolerance purposes, which is set up to three replications by default. On data set submission, HDFS breaks down the data into small data blocks to be replicated and distributed into VMs. HDFS tries to balance the distribution of data blocks among VMs fairly; however, the distribution of a given data block can be later altered if all VMs that contain the data block are busy. By using a data locality technique, YARN copies the functions of map and reduce to all salve VMs so that any VM can be selected to execute a map or reduce function. Whenever a MapReduce application is being executed, YARN would specify it as a MapReduce job, which holds information and records regarding a number of data blocks, the execution time of

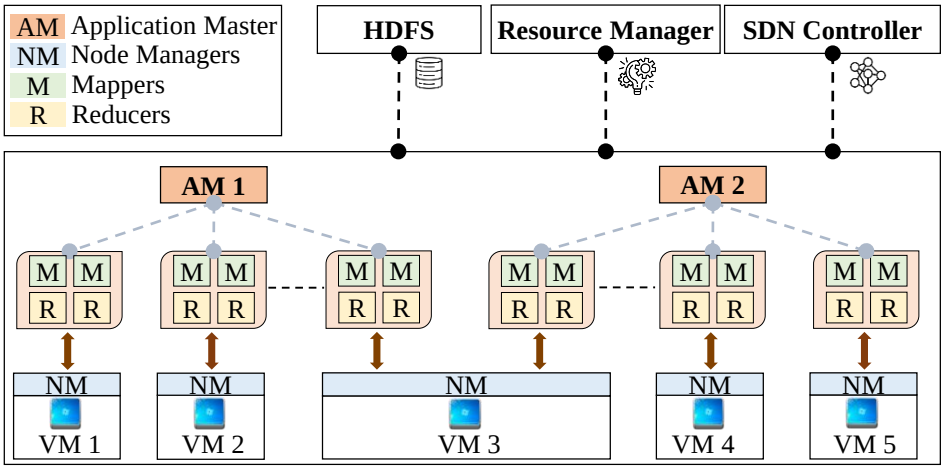


FIGURE 2 Overview of SDN-enabled YARN-related systems

every map and reduce task, etc. So, a MapReduce application and a MapReduce job are interchangeable. To avoid confusion, we use the "MapReduce application" throughout this article.

MapReduce applications can have different building blocks and characteristics based on processing and data flow requirements. The simplest model of MapReduce is illustrated in Figure 1, which consists of six phases, assuming that HDFS, mappers, and reducers reside in different nodes. As it can be seen the HDFS splits the data (D) into several blocks (e.g., B1, B2, B3) where each block is replicated and forwarded to elected VMs via a network layer. For every data block, a single mapper is selected to carry out the processing. Once every mapper completely processes the data, it starts the transmission of the output to respective reducers (e.g., R1, R2) according to the key-value pairs. The MapReduce key-value mechanism requires reducers to wait for all mappers to finish processing so that the output of reducers are accurate. Once every reducer finishes processing, it transfers its output to the HDFS to be combined and reported. The block replication mechanism is also required for reducers to divide their output into small-scale data blocks to be transferred to other elected VMs based on the replication factor.

Figure 2 illustrates an overview of SDN-enabled YARN systems running MapReduce applications. In a given data center, every big data cluster is managed by a resource manager and have a single HDFS shared by all MapReduce applications. The resource manager contains a list of worker VMs where each VM is controlled and monitored by an implemented agent (node manager). For every requested MapReduce application, the resource manager creates a new application master and links requested VM resources to the application master. Once the application master is activated, it copies the code-based functions of its map and reduce tasks to all respective VMs so that every VM can be elected to run the map and/or reduce task, as previously mentioned to ensure the principle of data locality. Following that, the execution logic of MapReduce applications follows the six processing and transmission phases, as mentioned earlier.

The description above of MapReduce, BDMS, and SDN should be considered to derive correct simulation models and ensure the accuracy of simulated results. BigDataSDNSim, therefore, is modeled and designed accordingly with the given descriptions. It also provides abstract layers for enforcing new policy-based solutions. For instance, a MapReduce application might be located on different servers and/or racks of a cloud data center due to insufficient resource capacity in a single server/rack. Such distribution logic requires accurate modeling of abstractions and interfaces so that users of our tool can seamlessly deploy their scheduling policies for MapReduce server-rack placement. In another example, the same MapReduce application might require special QoS requirements (e.g., traffic prioritization, policy-based routing mechanisms) and excessive data transmission on the network layer from one server to another. As SDN-enabled networks are capable of meeting such requirements, BigDataSDNSim is modeled to seamlessly provide easy deployments of QoS requirements on behalf of every MapReduce application.

3 | RELATED WORKS

Several simulation tools were developed due to the surge in adoption of MapReduce and SDN in cloud-based environments. Some of the tools were developed from the ground up while others were developed on top of existing tools. We believe that

there is a clear gap in state-of-the-art simulators in terms of modeling and support for MapReduce applications, BDMS, and SDN in cloud-based infrastructures. This section demonstrates existing simulators in terms of their ability to model and simulate MapReduce-DBMS within SDN-enabled cloud data centers. It then illustrates how our simulation framework fulfills the gap of existing simulation frameworks in a holistic manner. Table 1 illustrates the differences and similarities of existing simulators and emulators as compared to our simulation framework.

There are several simulation tools capable of simulating and modeling the characteristics of cloud and legacy networks. For example, CloudSim[23] is one of the most popular cloud-based tools that allow the modeling of physical and virtual cloud infrastructures using an event-driven architecture. It is capable of simulating and evaluating the performance of cloud infrastructures as well as deploying various provisioning and allocation policies (e.g., VM placement, CPU task scheduling). CloudSim forms the base upon which several simulation tools were developed to fill the gap of networks and applications aspects (e.g., NetworkCloudSim[26], WorkflowSim[27]). GreenCloud[28], iCanCloud[29], and NetworkCloudSim are other cloud-based tools focusing on the perspective of legacy networks in terms of characteristics and communications in cloud data centers.

Mininet[30] is an SDN-based emulation tool that runs on a single device configured with a Linux-based system. It emulates different types of network topologies together with hundreds of virtual hosts and diverse UDP/TCP traffic patterns. The external SDN controller(s) communicates and enforces network policies on Mininet via its unique IP address and OpenFlow APIs. While Mininet is limited to run on a single physical machine, MaxiNet[31] was introduced to enable Mininet to run across multiple physical machines. MaxiNet is capable of emulating large-scale SDN cloud environments along with evaluating new SDN-powered routing algorithms. Moreover, to allow Mininet to mimic the behaviors of MapReduce applications, MRemu[32] was introduced where it is capable of using realistic MapReduce workloads/traces within Mininet environments. It operates on latency periods extracted from MapReduce job traces (duration of tasks, waiting times, etc.).

Similarly, EstiNet[33] is another SDN-based simulation and emulation tool, allowing each simulated host to run a real Linux-based system coupled with several types of real applications. It is capable of simulating hundreds of OpenFlow switches and generating real TCP/IP traffic. CloudSimSDN[34] is an SDN-based cloud simulator that is modeled and implemented on top of CloudSim[23]. It enables the simulation of SDN-network behaviors in a cloud-based environment. The focus of CloudSimSDN is to model power-based management policies to reduce the energy consumption of hosts and network devices. SDNSim[35] is an SDN-enabled simulator that simulates data center elements (e.g. switches, links, and hosts) along with the layer of SDN data plane. The control plane of SDNSim is handled by an external SDN controller, which enforces network decisions and solutions through APIs. Moreover, SDN-Sim[36] is an SDN-based simulation and emulation tool that integrates several frameworks to facilitate the end-to-end performance evaluation of wireless technologies (e.g. 5G). The Virtual infrastructure of SDN-Sim depends on VMWare ESXi servers while its network layer is managed by OpenFlow protocol and Opendaylight controller. SDN-Sim's network can be emulated using GNS3 and Mininet. It depends on MATLAB server to run real world SDN wireless scenarios.

IoTSim[39] is an extension of CloudSim that mimics the characteristics of the MapReduce programming model. It allows simulation and modeling of the old version of the Hadoop framework (e.g., job trackers, task trackers). It provides a simple management mechanism to configure MapReduce applications based on IoT-generated data. Similarly, MR-CloudSim[40], MRSim[41], and MRPerf[42] are other tools that enable the simulation of MapReduce-based applications with different focuses and features. In addition, BigDataNetSim[43] is a simulator designed to evaluate the data placement strategies of HDFS within a dynamic network cluster. It focuses on evaluating solutions for transferring HDFS data to distributed nodes while it neglects the logic and dependencies of MapReduce. MaxHadoop[44] is an emulation tool developed on top of MaxiNet[31] to evaluate the performance of MapReduce strategies within an SDN network environment. The network within MaxHadoop is managed by an external SDN controller, such as Floodlight[38].

While these tools are powerful for simulating cloud-based environments, MapReduce applications, traditional networks, and SDN, BigDataSDNSim differs in supporting a holistic simulation framework that simulates MapReduce applications, BDMS, and SDN-related networks in cloud-based environments. In particular, BigDataSDNSim is capable of modeling and simulating:

- A generic big data approach for executing different big data programming models (e.g., MapReduce, Stream) simultaneously
- MapReduce applications within big data cluster management (BDMS), which is one of the prominent platforms for running different big data models
- behaviors and features of SDN dynamic networks coupled with the coordination and interaction with MapReduce applications within cloud environments

TABLE 1 Comparison of related simulators and emulators
✓* with the help of the INET framework[37]; ✓** with the help of SDN controllers from other projects (e.g. Floodlight[38])

Simulators	Features							
	Language	Evaluation Objectives	MapReduce Model	Network Model	BDMS Model	SDN Model	Cloud Model	Dynamic Routing Mechanisms
CloudSim[23]	Java	Performance					✓	
NetworkCloudSim[26]	Java	Performance		Limited			✓	
WorkflowSim[27]	Java	Performance					✓	
GreenCloud[28]	C++/Otcl	Energy		✓			✓	✓
iCanCloud[29]	C++	Performance		✓*			✓	✓*
Mininet[30]	Python	Performance		✓		✓**		✓**
MaxiNet[31]	Python	Performance		✓		✓**	✓	✓**
MRemu[32]	Python	Performance	✓	✓		✓**		✓**
EstiNet[33]	C++ & Python	Performance		✓		✓		✓
CloudSimSDN[34]	Java	Performance & Energy		✓		✓	✓	
SDNSim[35]	MATLAB & Python	Performance		✓		✓	✓	✓
SDN-Sim[36]	MATLAB, Java & Python	Performance		✓		✓		✓
IoTSim[39]	Java	Performance	✓				✓	
MR-CloudSim[40]	Java	Performance	✓				✓	
MRSim[41]	Java	Performance	✓	Limited				
MRPerf[42]	Python	Performance	✓	✓			Limited	
BigDataNetSim[43]	Java	Performance		✓		Limited		✓
MaxHadoop[44]	Python	Performance	✓	✓		✓		✓
BigDataSDNSim (Proposed)	Java	Performance & Energy	✓	✓	✓	✓	✓	✓

- dynamic routing mechanisms based on graph theory to enable any type of network topology to be seamlessly simulated
- model of several policies for SDN, MapReduce, and VM within cloud data centers for multilevel optimization

4 | ARCHITECTURE

This section demonstrates the fundamental functionalities and components of BigDataSDNSim. The modeling logic of our simulation framework is based on the overview in Section 2. Figure 3 presents the key components of our proposed architecture, in addition to a few used elements of CloudSim and CloudSimSDN. The Figure facilitates our simulator's use by categorizing the architecture into two main layers: programming and infrastructure and big data. The detailed description of every component in every layer is discussed later in Section 5. The overall description of each layer is given as follows:

4.1 | Programming layers

The programming layer facilitates the deployment of strategies, policies, and algorithms. It abstracts away the underlying complexities of BigDataSDNSim. It provides the baseline to implement multi-level MapReduce optimizations in SDN-enabled cloud environments. It consists of the following layers:

- *Input*: This layer allows the implementation of different simulation scenarios in a simple manner. The configurations of data centers (e.g., requirements and descriptions of hosts and networks) can be easily defined using a single JSON file. The layer also provides mechanisms for configuring big data clusters, such as the number of required VMs and MapReduce applications. The descriptions of processing and data transmission of MapReduce applications can be submitted in a CSV file – which includes attributes such as start time, and the size of data to be transferred between components (e.g., from HDFS to VMs). By using the input of this layer, BigDataSDNSim instructs its respective components to behave according to the given rules.

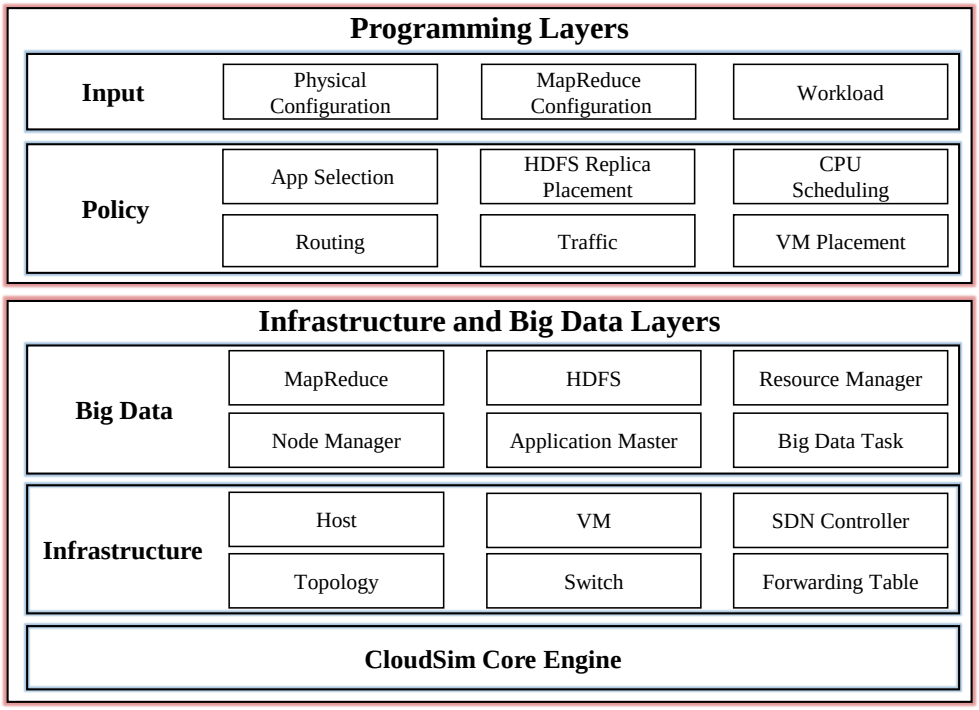


FIGURE 3 Architecture of BigDataSDNSim simulator

- *Policy*: By considering the importance of different policy requirements of MapReduce applications and SDN traffic engineering, this layer is designed to provide a mixed composition of MapReduce and SDN policies. Such policies are key factors for obtaining optimal performance in MapReduce processing and transmission. Therefore, these layers allow the implementation of new algorithms for every listed policy.

4.2 | Infrastructure and big data layers

The Infrastructure and big data layers contain the core, complex components of BigDataSDNSim. In case new functionalities emerge for big data programming models and SDN capabilities and are not supported by BigDataSDNSim, these layers should be extended. In this layer, most of the components communicate with each other using a discrete-event mechanism.

- *Big Data*: This layer holds components responsible for simulating the behaviors and characteristics of MapReduce and BDMS. Big data applications might require cross-engine data executions; thus, this layer integrates big data programming models/engines. Currently, this layer allows the simulation and analysis of the MapReduce model. As maintaining various programming models of big data is essential to support applications that demand different processing mechanisms (MapReduce, stream, etc.), more required components can be added to this layer.

- *Infrastructure*: It contains physical and virtual resources of cloud data centers. With virtualization mechanisms, hosts in BigDataSDNSim are designed to share their resources among multiple VMs, where each VM has its own memory, storage, and processor characteristics. This layer also maintains network and SDN entities. The fundamental functionalities, deployment, and management of network components and SDN are handled in this layer.

- *CloudSim*: This layer is equipped with the core entities, functionalities, and engine of CloudSim, such as resource provisioning and allocation and event processing mechanisms. It provides essential components to simulate cloud data centers. The BigDataSDNSim simulator operates on top of this layer, where entities can easily communicate via a discrete-event mechanism.

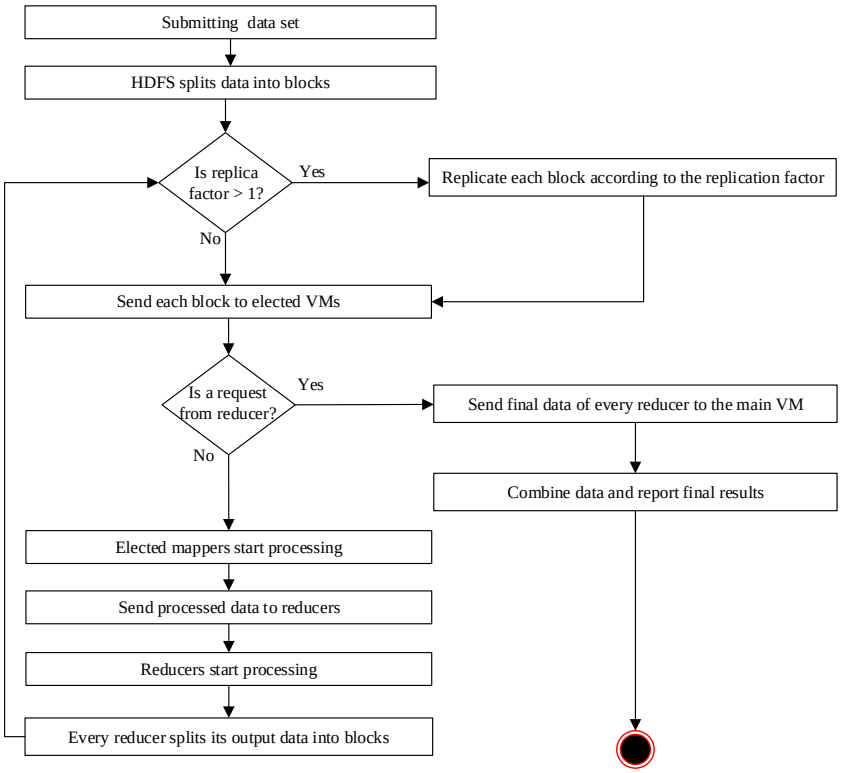


FIGURE 4 BigDataSDNSim MapReduce-HDFS Model

5 | SIMULATION MODELING AND DESIGN

This section illustrates the modeling and design of our simulation framework. It first demonstrates its theoretical modeling of BigDataSDNSim. Next, it illustrates multiple proposed designs of BigDataSDNSim, which includes system, policies, and interactions.

5.1 | Theoretical model

According to the YARN framework[6], the first step required for execution of MapReduce applications is the determination of HDFS data block size. HDFS can be defined as a distributed file storage that breaks down every received data set into small-scale data, referred to as a block, and stores all the blocks on distributed VMs. In a MapReduce YARN-based application, data is broken down into blocks for two times. First, when the HDFS splits a submitted data set into several HDFS blocks to be forwarded to elected VMs, which in turn are assigned to elected mappers for processing. Note that HDFS would only instruct one of the elected mappers to carry out the execution while the others are in a standby mode. The second time is when reducers finish executing where every reducer will split its output into several reducer blocks and distribute the blocks into elected VMs. We denote the former to "HDFS block" and the latter to "reducer block." The two types of blocks use the same block factor bs_c for determining their data block size.

Every generated HDFS block is first copied to elected VMs based on a given replication factor bs_c. One of the elected VMs is selected to run a new mapper for every received HDFS block, while other VMs are in a standby mode. This technique is known as "data locality" where computation is moved to a given data location (e.g., HDFS blocks) instead of vice versa, which helps in reducing MapReduce network loads. Moreover, the output of every reducer is divided into other blocks (reducer blocks) where each block is replicated and copied to elected VMs based on the same given replication factor bs_c. The HDFS replication mechanism ensures that the processing time of HDFS blocks by mappers is decreased while the reducer replication ensures that the final output of a MapReduce application, which combines data from all reducers, is available at all times. The modeling of BigDataSDNSim MapReduce-HDFS is illustrated in Figure 4.

By default, the HDFS engine assigns a default data size bs_d of 128MB to all HDFS and reducer blocks. Every HDFS data block is the input data for every mapper. To tune the number of required mappers, the block size bs_n can be configured. Equation 1 is used to select the overall block size where c is a decision variable to denote the demand for changing the default data block size. Let $H = \{1, 2, \dots, U\}$ be a set of HDFS blocks where each $h \in H$ has a data block size, denoted as D_h. Given the size of the MapReduce submitted data set ds, Equation 2 is used to compute the total number of HDFS blocks $|H|$. The total number of HDFS blocks $|H|$ is inversely proportional to bs_c, represented as $(|H| \propto (bs_c)^{-1})$. YARN applies division by repeated subtraction to determine the size of every HDFS block h, which means that all elected VMs that store HDFS blocks obtain the same size for all the blocks except for the last one. To properly determine the data size of every block $d \in D_h$, Equation 3 is used.

$$bs_c = \begin{cases} bs_d, & \text{if } c = 0 \\ bs_n, & \text{if } c = 1 \end{cases} \tag{1}$$

$$|H| = ceil\left(\frac{ds}{bs_c}\right) \tag{2}$$

$$D_h = \begin{cases} bs_c, & \text{if } h < |H| - 1 \\ ds\%bs_c, & otherwise \end{cases} \tag{3}$$

Every reducer needs to obtain intermediate data from every mapper. The output size of every mapper cannot be exactly determined because every mapper may produce different patterns of output. For the sake of simplicity, we assume that all of the reducers of a MapReduce application $a \in A$ obtain an equal size input from every mapper. Let $M = \{1, 2, \dots, Q\}$ be a set of mappers where each $m \in M$ has a data output, denoted as Out_m. The total number of mappers $|M|$ is equal to the total number of HDFS blocks $|H|$. Let $R = \{1, 2, \dots, W\}$ be a set of reducers where each $r \in R$ obtains input data, denoted as In_r. Equation 4 is used to estimate the data input size In_r for every reducer $r \in R$ by dividing the output size of every mapper $m \in M$ by the total number of reducer $|R|$. For every MapReduce application, there must be at least one reducer; therefore, BigDataSDNSim assigns a single reducer for every requested MapReduce application. However, the number of reducers can be changed if required.

$$In_r = \left(\frac{Out_m}{|R|}\right), \forall m \in M \tag{4}$$

The execution time of every mapper and reducer depends on the number of instructions that is required to be executed on their VMs, which is given in Million Instructions (MI). The execution time also depends on the speed of the central processing unit (CPU) of VMs, which is measured in Million of Instructions Per Second (MIPS). In discrete-event simulators, modeling the speeds, overheads, and sharing factors of CPU, memory, and a hard drive is hard, if not impossible. To correctly capture this type of model, a configurable parameter α is used to capture hidden overheads of VMs when needed. To compute the processing capacity for every VM assigned for any mapper or reducer, Equation 5 is used where C_{vm} is the processing capacity of every $vm \in VM = \{1, 2, \dots, E\}$, $mips(vm)$ is the MIPS speed of vm, and $cpu(vm)$ is the number of CPU cores of vm. Let P_m to be a processing demand of every $m \in M$, G_r be a processing demand of every $r \in R$, and $t \in \{M, R\}$ be a MapReduce task. The execution time of every task t can be computed using Equation 6 where $E(t, vm)$ denotes the execution time of t executed in a VM vm.

$$C_{vm} = mips(vm) * cpu(vm) * \alpha, \quad \alpha \in [0, 1] \tag{5}$$

$$E(t, vm) = \frac{P_t}{C_{vm}} \tag{6}$$

Every mapper and reducer might have different execution times due to CPU sharing mechanisms among all map and/or reduce tasks. In addition, the policy design (e.g., HDFS replica placement, CPU scheduling policies) plays a vital role on the execution time. The feasible option for computing the execution time of a given set of mappers and reducers belonging to a given MapReduce application is to observe the highest execution time of mappers and reducers. To estimate the total execution time $\mathcal{ET}(a)$ of every MapReduce application $a \in A$, Equation 7 is used.

$$\mathcal{ET}(a) = \max\{E(t, v)\} + \max\{E(t', v)\}, \quad \forall t \in M, \forall t' \in R, \exists v \in VM \tag{7}$$

By default, every MapReduce application $a \in A$ creates three replicas for every block to ensure fault tolerance in case of a VM failure. The replication is performed for every HDFS block and reducer block. Equation 8 is used to select the number of required replicas $\Phi \in \mathbb{N}$ where Ω is the requested replication factor and $|VM|$ represents the total available number of VMs. Ω cannot be higher than $|VM|$. Let each $\mathbb{R}_h \in \mathbb{N}$ be the set of replicas for each HDFS block $h \in H$. Given the set of the data of HDFS blocks D_h and the set of HDFS replica size $|\mathbb{R}_h| = \Phi$, every HDFS block $d \in D_h$ is mapped to every corresponding replica $o \in \mathbb{R}_h$ by using a matrix, denoted as $\mathbb{U}_h = D_h \times \mathbb{R}_h$. The data of every replica $u \in \mathbb{U}_h$ will be then transferred from the HDFS to every elected VM.

$$\Phi = \begin{cases} \Omega, & \text{if } \Omega < |VM| \\ |VM|, & otherwise \end{cases} \tag{8}$$

Similarly, every reducer divides and replicates its output into a number of reducer blocks based on the overall given block size bs_c. Every block is then replicated according to the replication factor Φ and forwarded to elected VMs. The replication is required to ensure that the final output of every MapReduce application is not lost when some VMs are not available. Let Out_r be a set of reducer output and $B_r = \{1, 2, \dots, P\}$ be a set of reducer blocks for each $r \in R$ where each $b \in B_r$ has some data, denoted as $d \in D_r$. The total number of reducer blocks $|B_r|$ is inversely proportional to bs_c, represented as $(|B_r| \propto (bs_c)^{-1})$. To determine the total number of blocks $|B_r|$ created by every reducer, Equation 9 is used. To estimate the data size of every reducer block $d \in D_r$, Equation 10 is used. Let Each $\mathbb{R}_r \in \mathbb{N}$ be the set of replicas for each reducer block $b \in B_r$. Given the set of the data of HDFS blocks D_r and the set of reducer replica size $|\mathbb{R}_r| = \Phi$, the data of every HDFS block $d \in D_r$ is mapped to every corresponding replica $n \in \mathbb{R}_r$ by using a matrix, denoted as $\mathbb{L}_r = D_r \times \mathbb{R}_r$. The data of every replica $l \in \mathbb{L}_r$ is then transferred from a VM that contains a corresponding reducer $r \in R$ to every other elected VM.

$$|B_r| = ceil\left(\frac{Out_r}{bs_c}\right), \forall r \in R \tag{9}$$

$$D_r = \begin{cases} bs_c, & \text{if } r < |B_r| - 1 \\ Out_r \% bs_c, & otherwise \end{cases} \tag{10}$$

A network channel must be established to transfer data from a source VM to a destination VM if they reside in different hosts. The modeling of channels prevents VMs from overloading any given link existing in its route, which would lead to network congestion. Let $L = \{1, 2, \dots, K\}$ be a set of links where each $l \in L$ has a bandwidth BW_l and a number of associated channels NC_l passing through. Let $C = \{1, 2, \dots, Z\}$ be a set of channels traveling through some links where each $c \in C$ obtains a bandwidth BW_c based on the smallest bandwidth of links existing on the route of the channel c. To compute BW_c for every channel c, Equation 11 is used where $BW_c(s, d)$ is the bandwidth of channel c from a source VM s to a destination VM d and BW_l is the bandwidth of a link l that the channel c traverses through and NC_l is the number of channels traveling and sharing a link l.

$$BW_c(s, d) = \frac{min\{BW_l(s, d)\}}{NC_l}, \quad s, d \in VM, \exists l \in L \tag{11}$$

For any data to be transferred via an SDN-enabled network, it must be encapsulated inside a network flow. We avoid using network packet modeling to prevent our simulation tool from generating millions of network packet objects, which would overload the computing resources of a machine that runs our simulator. Define NF_c as a number of flows that a channel $c \in C$ has and $F = \{1, 2, \dots, U\}$ as a set of flows where each $f \in F$ has a network bandwidth, denoted as BW_f. As every channel bandwidth BW_c can be shared by flows that travel from a source VM s to a destination VM d, the bandwidth of every flow f can be computed using Equation 12.

$$BW_f(s, d) = \frac{BW_c(s, d)}{NF_c(s, d)}, \quad \exists c \in C \tag{12}$$

Every MapReduce application requires to transfer data at different times. The transmission of data can be summarized as follows: (1) from HDFS to elected VMs, (2) from mappers to reducers, (3) from reducers to other elected VMs, and (4) from reducers to a VM that reports the final MapReduce results. Let $x \in \{\mathbb{U}_h, Out_m, \mathbb{L}_r, Out_r\}$ be data where each x has an associated

flow f to transfer the data of x from one VM to another via the SDN-enabled network. To compute the transmission time of every data x, Equation 13 is used. The total transmission time $\mathcal{TT}(a)$ of a MapReduce application $a \in A$ is calculated using Equation 14. The overall completion time $\mathcal{CT}(a)$ of a MapReduce application a, which is composed of all executions and transmissions, is computed using Equation 15.

$$\mathcal{T}(x) = \frac{x}{BW_f(s,d)}, \quad \exists s, d \in VM, \exists f \in F \tag{13}$$

$$\mathcal{TT}(a) = \max\{T(x)\} + \max\{T(x')\} + \max\{T(x'')\} + \max\{T(x''')\}\}, \quad \forall x \in \mathbb{U}_h, \forall x' \in Out_m, \forall x'' \in \mathbb{L}_r, \forall x''' \in Out_r \tag{14}$$

$$\mathcal{CT}(a) = \mathcal{ET}(a) + \mathcal{TT}(a) \tag{15}$$

Depending on the value of replication factor, the generated data of every MapReduce application $a \in A$ is different. To compute the total generated data $TD(a)$ of every a, Equation 16 is used. The data characteristics generated by different types of MapReduce applications differ from one another. For example, mappers in page-ranking MapReduce-based applications often have a larger size of outputs as compared to the size of their inputs, while word-count MapReduce-based applications follow the opposite[45]. The modeling logic of BigDataSDNSim allows different types of MapReduce applications to be seamlessly simulated by tuning the values of $x \in \{\mathbb{U}_h, Out_m, \mathbb{L}_r, Out_r\}$.

$$TD(a) = \sum_{\forall x \in \mathbb{U}_h} x + \sum_{\forall x' \in Out_m} x' + \sum_{\forall x'' \in \mathbb{L}} x'' + \sum_{\forall x''' \in Out_r} x''' \tag{16}$$

5.2 | Implementation

As mentioned earlier, BigDataSDNSim is a discrete-event simulator. Every element that requires communication and cooperation with other elements (e.g., data center, resource manager, SDN controllers, etc.) must do so by issuing events that are processed and delivered by the simulation engine. Every event may contain actions to be carried out and data to be used by destinations. The following describes and indicates the property of every component developed in BigDataSDNSim:

- *ResourceManager*: This entity is responsible for the configuration, deployment, and scheduling of worker nodes' resources among competing big data applications. It carries out provisioning mechanisms for new big data applications in a given big data cluster. It tries to reserve the required resources using VM usage statistics. If cluster's resources are insufficient, requests are held in a waiting queue. Once resources become available, the required resources are reserved. The queue is based on a first-come-first-served mechanism. This entity can be instructed with different policies if required.

- *ApplicationMaster*: It is designed to manage the application life-cycle of big data programming models. A new application master is initiated for every new big data application. The initiation of every application master is carried out by a resource manager. The current development of this entity is based on the MapReduce programming model.

- *NodeManager*: This entity is in charge of controlling and monitoring VM resources. Every node manager is coupled with a single VM to track and report the status of its VM. It informs the actual usage of the VM to a resource manager. Such update is used to properly allocate a cluster's resources among big data applications, alleviating resource contention.

- *NetworkNIC*: It is an interface equipped in every node. It is responsible for establishing and maintaining the network connection of a given node. It is similar to the network interface card (NIC) embedded in most of today's devices. The key feature of the interface is to allow the modeling of southbound network management protocols (OpenFlow and Open vSwitch[46] (OVS)) in switches and hosts. Such protocols enable SDN controller(s) to have a full network control of nodes, build routing and forwarding tables, and capture a global network view.

- *SDNController*: This is an entity designed to mimic the behavior of an SDN controller. It provides abstractions for programming and monitoring networks dynamically. By gathering the information from each node (switches and hosts), the controller builds dynamic routing for each host, VM, or application based on a given routing algorithm. The controller is developed to seamlessly shape network traffic for each MapReduce application based on a given traffic policy. The

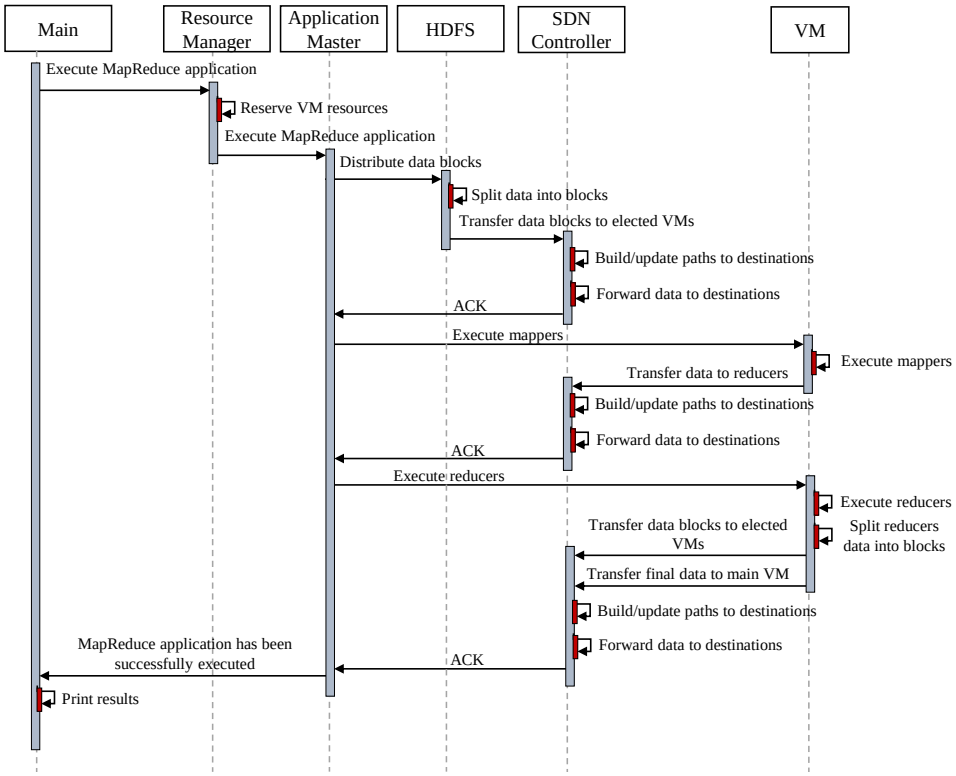


FIGURE 5 Underlying interactions of BigDataSDNSim (overview)

network can be controlled and managed by more than one SDN controllers if required. Network optimization and the reconfiguration of a given network can be easily achieved by implementing smart routing and traffic policies.

- *HDFS*: Every data submitted to a MapReduce application is stored in a distributed manner by the HDFS. The entity divides the data set into several blocks based on given block size. Once data blocks are determined, it copies them to elected VMs via a network infrastructure. The election of VMs is based on a given HDFS replica placement policy.
- *BigDataTask*: This entity can be shaped in different ways based on a selected big data application. The current design of BigDataTask represents the modeling of mappers and reducers. In the case where a new big data model is required (e.g., stream), the entity can be inherited to obtain the common features that all other big data tasks have in common.
- *Topology*: It is responsible for maintaining a network graph and relations among switches and hosts. An SDN controller uses this entity to obtain network graphs and relations and adjust the network graph if required. Moreover, the SDN controller uses this entity to build routing and forward tables among VMs and applications.
- *Flow*: This is used to facilitate the process of network traffic modeling. A network flow is initiated for every traffic from a source component to a destination component (e.g., hosts, VMs, MapReduce applications). An SDN controller shapes the network traffic of every flow according to given routing and traffic policies.

Figures 5 demonstrates an overview of the interactions and workflows among entities throughout the runtime of BigDataSDNSim. The simulator's functionalities are classified into four phases: building required infrastructure, establishing requested MapReduce application(s), carrying out task processing and data transmission, and finally reporting the results of every MapReduce application. The infrastructure is built by parsing a configuration file provided in the JSON format. Once BigDataSDNSim obtains the file, it initiates the corresponding objects of hosts, switches, and network links. At the same time, it establishes the required components, such as the SDN controller and resource manager, and builds a required network topology. Once the resource manager is active, it couples every VM with a node manager for monitoring and reporting purposes.

Shortly after BigDataSDNSim starts, it instructs the resource manager to initiate every requested MapReduce application. The resource manager would create a single application master for each required MapReduce application. Once every application master is active, every application is fed with the configuration requirements in a CSV file containing different information such as start time, number of mappers, size of flows, and amount of MIs. Each MapReduce application is composed of computational tasks and data flow transmissions, which are carried out in a specific order. The current application logic supported in BigDataSDNSim is based on two processing activities and four transmissions, as depicted earlier in Figure 1. For the processing stage to take place, two requirements must be met: (i) the execution logic of map and reduce tasks must be placed into VMs, and (ii) mappers and reducers must acquire the whole required data to start execution. The transmission stage, on the other hand, includes four sequence activities (1) dividing initial data into HDFS blocks and transferring the blocks from HDFS to elected VMs; (2) transferring the output of mappers to reducers; (3) dividing the output of reducers into blocks and transferring the blocks to elected VMs; (4) transferring the output of reducers to a VM, which reports the final MapReduce results.

Any element (e.g., HDFS, mapper, reducer) that requires data transmission must create a network flow and inject its data to it. Next, it encapsulates the flow inside an event and sends the event to an SDN controller through the implemented discrete-event mechanism. When the SDN controller receives the event, it updates the progress of existing network flows and removes completed ones along with idle network channels. Next, the SDN controller updates or builds forwarding tables based on VM-to-VM communications, with respect to the implemented SDN routing algorithms. If there is no existing route between a source and a destination, the SDN controller will build a route, add a forwarding rule to every node along the route (switches and hosts), create a channel, and encapsulate the received flow inside the channel. In order for the SDN controller to track the completion of flows, it calculates the earliest finish time eft of existing flows, creates an event with an invoking time of eft, and sends the event internally to itself to be intercepted according to the given time of eft. In general, the eft can be computed using Equation 17 where $f_r(j)$ is the remaining data of the j_{th} flow to be transferred and $c_{bw}(j)$ is the flow bandwidth of the j_{th} flow.

$$eft = \min\left\{\frac{f_r(j)}{f_{bw}(j)}\right\} \tag{17}$$

Once there is no more activity or event to take place, the simulation concludes and results are reported. The report structure reflects the information of MapReduce applications, performance measurement of transmission and processing, and information of SDN forwarding tables. The overall output illustrates every MapReduce application's status, such as submission time, queuing delay, start time, etc. The processing result shows the performance metrics of mappers and reducers (e.g., VM's ID, start time, execution time), while the transmission result demonstrates the statistics of every connection (IDs of source and destination, size of flows, start time, transmission time, etc.). The information of SDN forwarding tables shows the list of traversing nodes for every flow and any changes made to the flow's bandwidth in terms of size and time throughout the transmission period.

5.3 | Policies design

Developing new policies that dynamically respond to changing behaviors and optimize the performance of different MapReduce applications running in SDN-enabled clouds is essential. Implementing such policies within BigDataSDNSim was a complex endeavour due to complex interactions among BigDataSDNSim's components. Therefore, we modeled and developed our simulator to support host-network policies so that different MapReduce policies can be seamlessly implemented. Figure 6 shows a sample of different policies currently developed in BigDataSDNSim. The policies are categorized into five groups: app selection, HDFS replica placement, VM-CPU scheduling, routing, and traffic. Several polices are developed for each group. The significance of each group is described as follows:

- *Application selection*: Resources reserved for a big data cluster may be limited. Therefore, an application selection policy is essential to determine the selection criteria based on given QoS (e.g., deadlines). A resource manager queues incoming MapReduce applications and schedules them based on resource availability and given selection policy. There are two different application selection policies implemented within BigDataSDNSim: prioritization and first come, first served.
- *HDFS replica placement*: The replication mechanism is an integral part of MapReduce production systems to ensure reliability and performance. There are many existing policies with different goals to tune the replication mechanism according to a given criteria and constraints (e.g., rack-aware replica placement policy). Therefore, we modeled the HDFS replica placement policy and also included examples of general policies (e.g., round-robin, least used VM, most used VM). New HDFS replica placement policies can be implemented by extending this component.

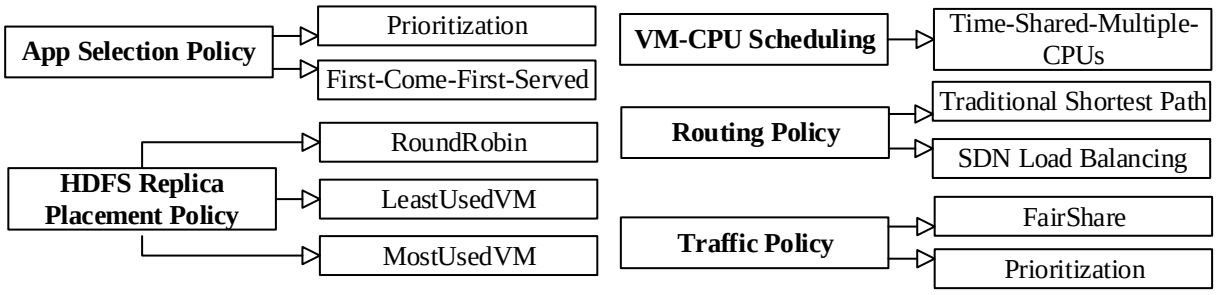


FIGURE 6 Modeling of policies in BigDataSDNSim

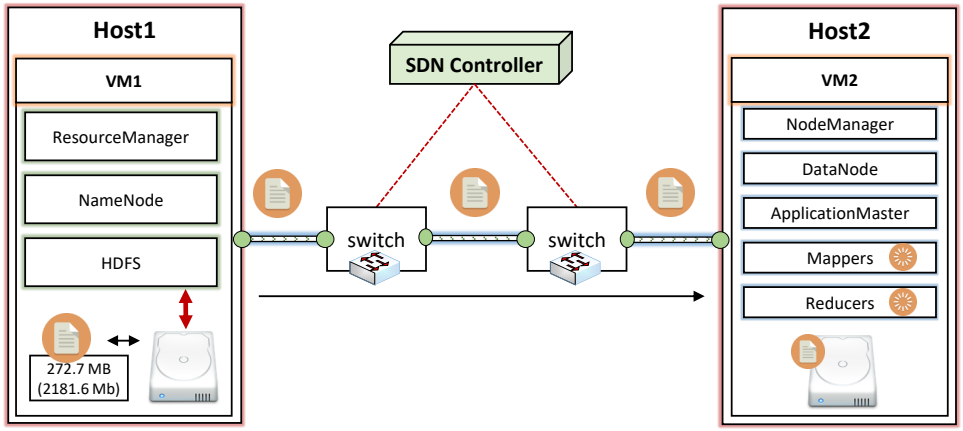


FIGURE 7 Validation setup of the real experiment

- *VM-CPU scheduling*: Multiple tasks can be scheduled to be executed in a single VM. The scheduling criteria can be different from one MapReduce application to another. The VM-CPU scheduling was originally implemented in CloudSim. The overall MapReduce processing performance depends on this policy; therefore, we implemented a new VM-CPU scheduling policy called Time-Shared-Multiple-CPUs to enable the use of multiple CPU cores by mappers and reducers.

- *Routing*: In general, the structure of every network is represented as graphs. The network and routing mechanisms of BigDataSDNSim are modeled in a dynamic manner to establish pairwise relationships among different entities (e.g., switches, hosts, VMs, and applications). By such dynamic modeling, BigDataSDNSim enables any network type to be simulated and pairwise relationships and routes to be defined based on different policies (e.g., shortest path). It also enables an SDN controller to manage the entire network routes according to the given criteria. BigDataSDNSim implements the Dijkstra's algorithm[14], which is capable of finding the shortest paths from a single source (e.g., VM, mapper, reducer) to all other destinations based on a single objective, which is a minimum number of traversed nodes. We also enhanced the Dijkstra's algorithm to meet multiple objectives by proposing SDN load balancing algorithm. The objective of our proposed algorithm is to constantly load balance network traffic by finding different paths based on maximum bandwidth.

- *Traffic*: As different types of applications share the resources of a given network, there must be some mechanisms to ensure the network traffic quality for every application. The traffic requirements and flows of every MapReduce application can be different; therefore, the modeling of network traffic policy within BigDataSDNSim is essential to shape the data flows of MapReduce applications according to some QoS criteria. To illustrate the advantage of modeling the traffic policy, we implemented two traffic policies: fair-share and prioritization. New MapReduce SDN-enabled traffic policies can be implemented by extending this component.

6 | VALIDATION OF BigDataSDNSim

This section reports the simulation accuracy and correctness of BigDataSDNSim by comparing the behavior and results of a real environment that combines MapReduce and SDN with an identical simulated environment that is provided by BigDataSDNSim. The experiment is conducted on two host machines. Each one has Intel Core i77500U 2.70 GHz, 16 GB of RAM memory, and 1000 Mbps of NIC. Each host is equipped with a single guest host (VM) running Linux 4.4.0-31-generic. Each VM has 4 virtual CPUs and 4 GB of memory. Two switches designed by Shenzhen Helor Cloud Computer [16] are used. Each switch has Intel Celeron 1037U (2 Cores, 1.80 GHz), 4GB of memory, and 6 Ethernet ports. The throughput of each port is 1000 Mbps. On each switch, Linux 4.15.0-29-generic and Open vSwitch (OvS) [17] are installed. The OvS is a virtual switch used to allow an SDN controller to instruct and control the switches' data plane via an OpenFlow protocol. An SDN-based framework called Ryu is used as an SDN controller[47]. The type of cables that connect machines and switches is Ethernet Cat5e. Each cable attains 1000 Mbps of speed.

For the real experiment, we executed a word count MapReduce application using the Hadoop-YARN framework[6]. We executed the experiment six times and reported the average processing and network transmission times. Moreover, two simulated experiments were carried out: BigDataSDNSim with α overhead (simulated exp1) and BigDataSDNSim without α overhead (simulated exp2). The obtained results of these experiments are compared with the real experiment. The simulated exp1 is intended to slow the simulated VMs to represent performance overheads (e.g., I/O operations, etc.) introduced in the VMs of the real experiment. *The configurations of MIPS, VMs, and network that we obtained from the real testbed was replicated in the simulated experiments.* Also, the same setup scenario of the real experiment (shown in Figure 7) is replicated in the simulated experiments. The configuration validation of the real and simulated experiments is illustrated in Table 2.

The validation scenario is designed to have one VM that contains HDFS while the other VM contains two mappers and a single reducer. The HDFS contains a text file of 272.7 MB that needs to be transferred to the mappers. Once the whole file is transferred, the mappers start processing. In the real experiment, the Hadoop framework tries to place mappers and reducers in the same VM; therefore, there is no need for network transmission between the mappers and reducers. We configured the real experiment to have a single replication of data blocks; thus, the destination VM would only receive unreplicated data blocks. Moreover, the reducer would not transfer its output and data blocks to other VMs since the replication factor is set to one. Data is only transferred once from HDFS to mappers. Table 3 shows the MapReduce configuration parameters used in the validation, which is obtained using the Equations 18-23.

In the real environment, we executed a Linux command on VMs to obtain the number of MIPS *nvm*, which only indicates the MIPS for a single core. Since the VMs have more than one core, we used Equation 18 to estimate the total number of MIPS *tnm* for every VM where *nvm* is multiplied by the total number of VM cores *vmc*. We estimated the real network bandwidth *bw* of the real experiment by dividing the file size fs by a network transmission time *ntt*, as shown in Equation 19.

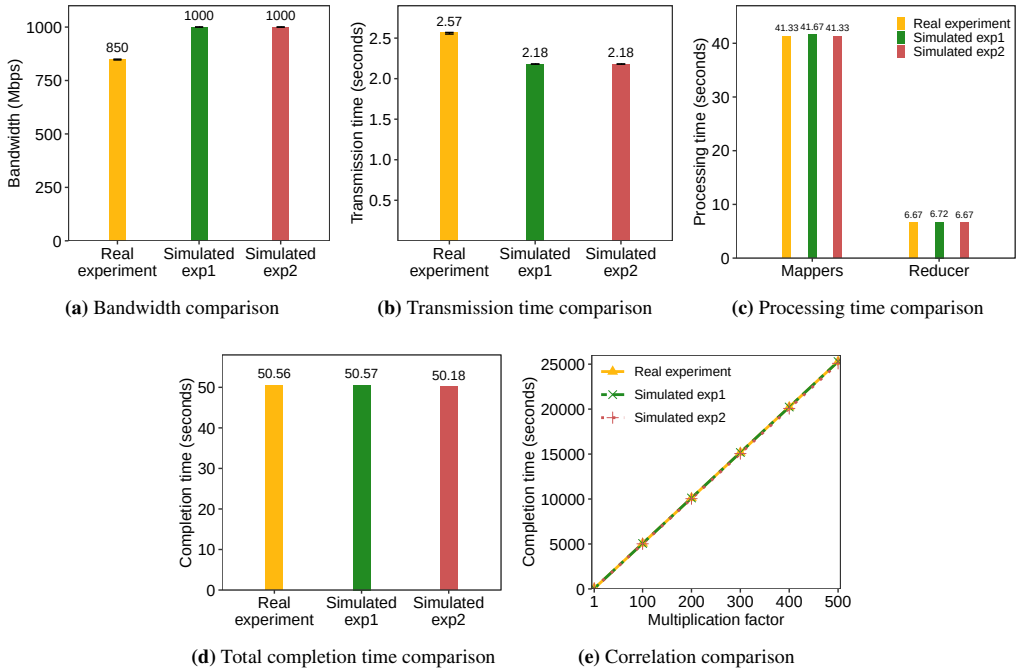
$$tnm = nvm * vmc \tag{18}$$

TABLE 2 Validation configuration

Environment	Configuration for every VM				Network Bandwidth
	MIPS	Number of cores	Total number of MIPS	Memory size	
real experiment	3592	4	14368	4GB	850 Mbps
BigDataSDNSim with α overheads	3563	4	14252	4GB	1000 Mbps
BigDataSDNSim	3592	4	14368	4GB	1000 Mbps

TABLE 3 Configuration for validating MapReduce application

Environment	Total executed MIPS per mapper (mp_{tmi})	Total executed MIPS per reducer (rd_{tmi})	Number of mappers	Number of reducers	File size (HDFS to mappers)	Replication factor
Real experiment	296939	100576	2	1	272.7 MB	1
Simulated experiments	296939	100576	2	1	272.7 MB	1



(a) Bandwidth comparison **(b)** Transmission time comparison **(c)** Processing time comparison

(d) Total completion time comparison **(e)** Correlation comparison

FIGURE 8 Comparison of the real experiment, simulated exp1, and simulated exp2

$$bw = \frac{fs}{ntt} \tag{19}$$

By using Equation 20, the number of MIPS for mappers mp_{mi} is estimated where tnm is divided by the number of mappers mp_n running in the VM. We use Equation 21 to obtain the number of MIPS for reducers rd_{mi} where tnm is divided by the number of reducers rd_n running in the VM. Note that reducers only run after all mappers are finished; therefore, the number of MIPS for mappers and reducers are different. Equation 22 is used to calculate the total number of MIPS mp_{tmi} executed by mappers where mp_{mi} is multiplied by the maximum execution time mp_{ex} taken by the mappers obtained in the real experiment. Equation 23 is used to estimate the total number of MIPS rd_{tmi} executed by reducers where rd_{mi} is multiplied by the maximum execution time rd_{ex} taken by the reducers.

$$mp_{mi} = \frac{tnm}{mp_n} \tag{20}$$

$$rd_{mi} = \frac{tnm}{rd_n} \tag{21}$$

$$mp_{tmi} = mp_{mi} * \max(mp_{ex}) \tag{22}$$

$$rd_{tmi} = rd_{mi} * \max(rd_{ex}) \tag{23}$$

Figure 8 shows the comparison results of real and simulated experiments. In Figure 8a, it can be seen that the bandwidth of the real experiment is 850 Mbps, while the simulated experiments are 1000 Mbps. The small discrepancy is acceptable because there are many factors that hinder the real VMs to achieve the maximum network capacity, such as the speed of CPU, the speed of hard drive (I/O), and the size of memory assigned to the MapReduce application. Figure 8b illustrates the transmission times taken to transfer the text file from the source VM (HDFS) to the destination VM that contains the mappers. It can be seen that the real experiment shows a slightly higher transmission time compared with the simulated experiments. Such difference of time can be reduced by changing the value of α parameter in the simulated exp1.

In Figure 8c, it can be observed that processing time of mappers and reducer of the real experiment and simulated exp2 are the same. Such similarities are expected since the VMs, mappers, and reducer of both experiments have similar configurations of MIPS. In the same figure, the simulated exp1 has a higher processing time as compared to the real experiment because of the

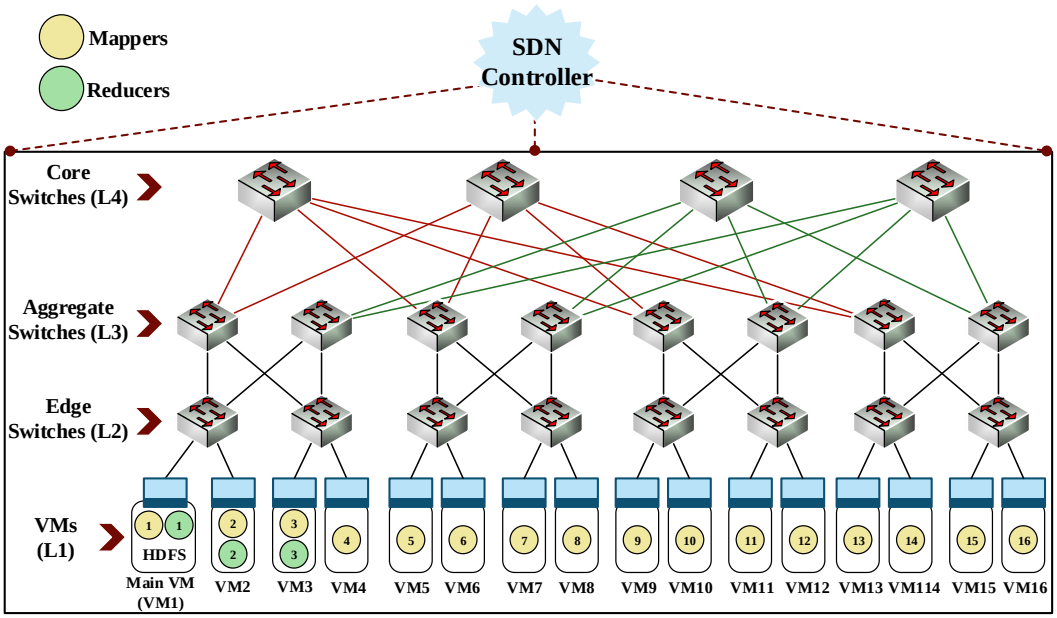


FIGURE 9 Fat-tree topology used in the simulated use-case experiments

value of α parameter, which makes up the time difference of the network transmission in the real experiment. Figure 8d shows the completion time taken in every experiment. By tuning the value of α parameter of the simulated exp1, the simulated exp1 is capable of obtaining approximately a similar completion time as compared to the real experiment. Also, the completion time of the simulated exp2 is closely comparable to the real experiment. Figure 8e illustrates the correlation of the completion time among the experiments. We derive the figure based on the completion time of Figure 8d. It can be seen that the completion time has a strong positive correlation, which reveals that the accuracy and correctness of BigDataSDNSim are closely comparable to the real SDN-enabled MapReduce environment.

7 | USE CASES OF BigDataSDNSim

For the purpose of demonstrating the practicality and advantages of using BigDataSDNSim, we present two use cases: *one replica (1R) and three replicas (3R)*. They shed light on MapReduce performance in terms of illustrating the impact of using HDFS replication mechanisms and the advantages of using SDN. We developed and implemented several MapReduce policies and routing algorithms, which are used in the simulated use-case experiments. Following is the list of policies we modeled and implemented in BigDataSDNSim:

- *MapReduce application selection policy* is configured on a first-come-first-served basis.
- *HDFS replica placement policy* is configured based on round-robin, where each replicated block is forwarded to an elected VM in a sequence-based manner.
- *VM-CPU scheduling* is configured based on time-shared-multiple-CPUs, where every map and reduce task can use multiple CPUs for processing.
- *Traffic policies* are configured as fair-share, where all the network flows obtain an equal amount of bandwidth.
- *Routing algorithms* are configured to use two different routing policies: SDN load balancing and traditional network shortest path. Each routing policy is used in both 1R and 3R experiments and final results are presented and compared.

Both routing algorithms play a vital role in illustrating some advantages of using our simulator. They clearly show the difference between traditional networks and SDN-enabled networks in terms of optimizing performance of MapReduce applications.

TABLE 4 Infrastructure configuration for the use-case

VM	**CPUs**	**RAM (GB)**	**MIPS/CPU**	**Total MIPS**
	4	4	1250	5000
Network	**Link**	**Bandwidth**	-	-
	HDFS to edge switch	3 Gbps	-	-
	VMs to edge switches	1 Gbps	-	-
	edge switches to aggregate switches	1 Gbps	-	-
	aggregate switches to core switches	1 Gbps	-	-

TABLE 5 MapReduce configuration for the use-case experiments

Use-case	Replication factor	Total executed MIPS per mapper	Total executed MIPS per reducer	Number of mappers	Number of reducers	Data size - HDFS to VMs (mappers)	Data size – mappers to reducers	Final data size - reducers to the main VM	Replicated data size - reducers to VMs	HDFS Block Size
Single replica (R1)	1	500000	150000	16	3	1 * 15 = 15 GB	12 GB	3 GB	1 * 3 = 3 GB	950 MB
Three replicas (R3)	3	500000	150000	16	3	3 * 15 = 45 GB	12 GB	3 GB	3 * 3 = 9 GB	950 MB

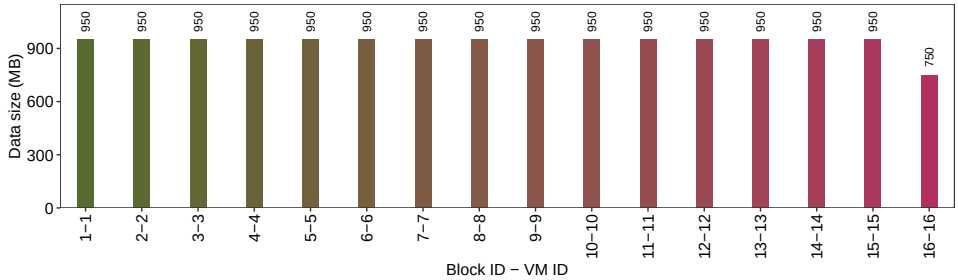
For the traditional networks, we implemented the well-known Dijkstra's algorithm[14] to find the shortest paths to destinations based on a minimum number of traversing network nodes. One limitation of traditional networks is that it lacks the selection of different paths from a specific source to a specific destination in a dynamic manner, despite having many elected paths. It finds all elected paths and randomly selects one path where flows of respective source and destination permanently travel via the selected path. On the other hand, SDN-enabled networks can program the network on the fly; therefore, we modeled and implemented an SDN load balancing (SDN-LB) algorithm by extending the Dijkstra's algorithm with two objectives of finding routes that obtain a minimum number of traversing node and then finding a route that has the maximum bandwidth among the elected routes. Every time a new flow enters the network, the SDN controller attempts to balance the usage of links by finding an appropriate route for the flow based on our proposed algorithm. For example, two flows from the same VM can have two different routes to the same destination VM, which would theoretically and practically reduce network transmission time for MapReduce applications.

To set up the simulated environments, we created a single cloud data center with the most used fat-tree topology in existing data centers[48]. Figure 9 depicts the physical topology that includes three layers of switches and one leaf layer of hosts containing VMs. There are four core switches (L4), eight aggregation switches (L3), eight edge switches (L2), and 16 VMs (L1) at the leaf of the tree. The network and VMs are configured according to Table 4. The structure and link arrangement of the fat-tree topology enables all switches to have additional links to one another so that transmission of data can follow different paths according to a given routing and QoS policies. Every link is configured with a bandwidth of 1 Gbps. The main VM that contains the YARN engine, including the HDFS file system, is connected to the edge switch via a link's bandwidth of 3 Gbps. As the HDFS node is connected to a single link, it would not take advantage of SDN load balancing if its link cannot accommodate a high volume of data. Thus, it is important to have such high bandwidth to demonstrate the impact of SDN load balancing.

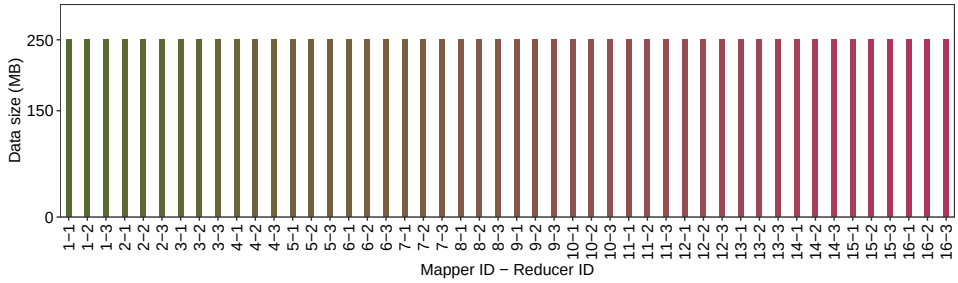
The MapReduce configurations of R1 and R3 experiments are shown in Table 5. For R1, the replication factor is set to one, which means that every block of HDFS and reducer is only transferred to a single elected VM. On the other hand, R3 is configured to replicate every HDFS data block to three different VMs along with requesting every reducer to split its final output into blocks and replicate each block according to the replication factor. Such configuration demonstrates the impact of replication mechanisms on the overall MapReduce performance. As the HDFS block size can be altered, we set it up to 950 MB so that the number of mappers is equal to 16. We try to place HDFS, mappers, and reducers in separate VMs as much as possible in order to heavily stress the network within more MapReduce traffic.

7.1 | Results of use cases

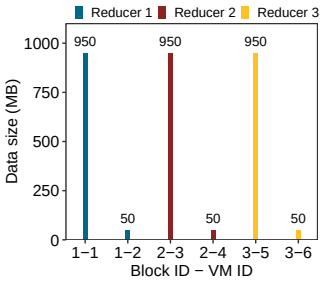
BigDataSDNSim derived the proper data sizes to be transferred from one element to another (e.g. from HDFS to elected VMs) based on Table 5 and the MapReduce-HDFS modeling in Figure 4. Figure 10a demonstrates the size of blocks to be transferred from HDFS, residing in the main VM, to other elected VMs, which are nominated for running mappers. It can be seen that the last block(s) contain fewer data compared to others due to the use of division by repeated subtraction. Figure 10b shows the output size to be transferred from every mapper to every reducer. We assume that the output of every mapper is equally



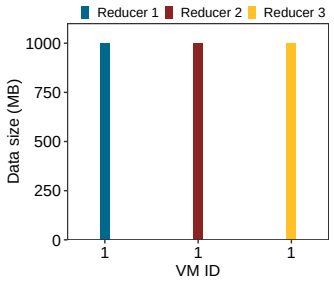
(a) Block size transferred from HDFS to elected VMs (R1)



(b) The output size of every mapper transferred to every reducer (R1)



(c) The output block size of every reducer transferred to elected VMs (R1)

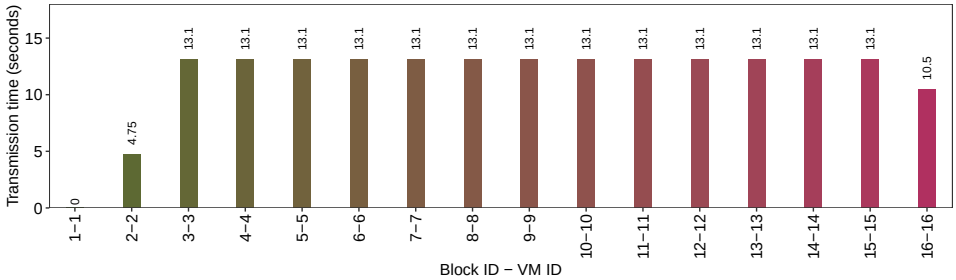


(d) The final output size of every reducer transferred to the main VM (R1)

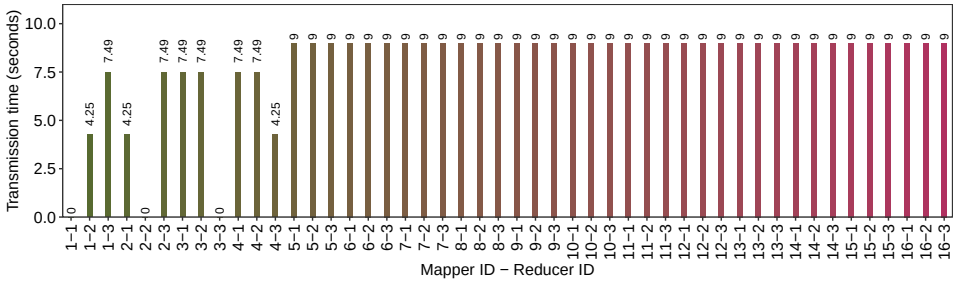
FIGURE 10 Data size of a single replication (R1)

divided by the total number of reducers. Figure 10c illustrates the block size of every reducer to be transferred to elected VMs. As mentioned earlier, the HDFS requires every reducer to split its output into blocks, replicate every block according to the replication factor, and send every replicated block to nominated VMs. Finally, Figure 10d shows the total output size of every reducer to be transferred to the main VM to be combined as the final data.

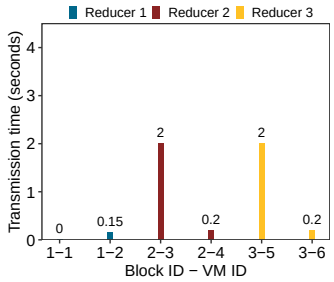
Figure 11 and Figure 12 illustrate the transmission time of R1 in the traditional network and SDN load balancing, respectively. Some of the transmissions are equal to zero because both the source and destination elements reside in the same VM. Moreover, some of the transmissions are shorter than others. This is because the number of transmissions traveling in the same path is smaller than the number of transmissions traveling via different paths, which results in shorter transmission time. Figure 13 shows the performance comparison of the R1 experiment between the traditional network and SDN load balancing. In Figure 13a, it is apparent that the SDN load balancing decreases the transmission time by approximately 45% as compared to the traditional network. Figure 13b shows the execution time of the mappers and reducers. It is expected similar execution times for the SDN and traditional network as the policy for the CPU scheduling, the distribution of mappers and reducers, and the number of MIPS is the same. The results in Figure 13c show the total completion time of R1 in the traditional network and SDN load balancing. The total completion time is determined from the time the HDFS starts transferring data blocks and ends when the reducers send their final output to the main VM. It can be seen that the SDN load balancing decreases the total completion time by approximately 8% in comparison to the traditional network.



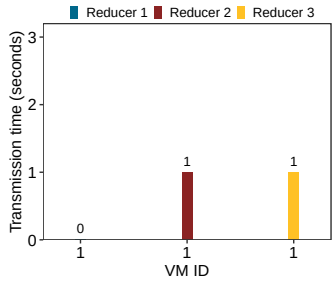
(a) Transmission time of blocks from HDFS to the VMs of mappers (R1)



(b) Transmission time of intermediate data from every mapper to all reducers (R1)



(c) Transmission time of blocks from reducers to elected VMs (R1)



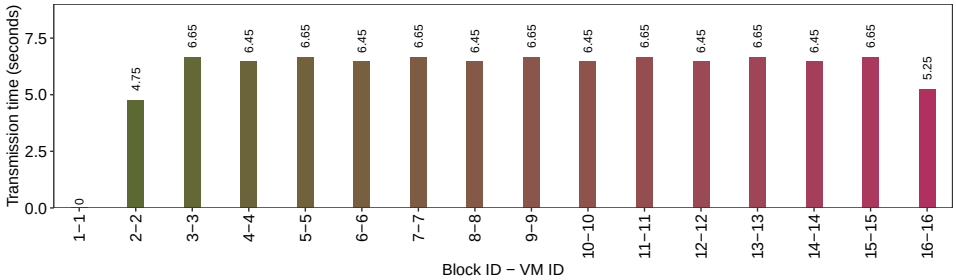
(d) Transmission time of final outputs from reducers to the main VM (R1)

FIGURE 11 Transmission time of MapReduce in a traditional network (R1)

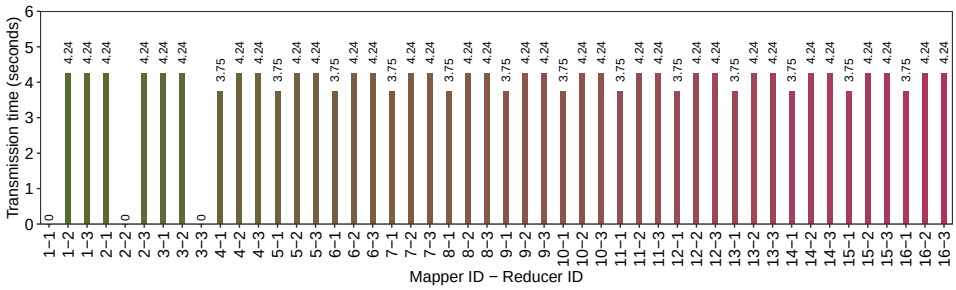
Figure 14 illustrates the data size to be transferred from one element to another in R3. The only difference between R3 and R1 is that the former replicates every block of HDFS and reducer three times, which increases the overall data sizes. Figures 15 and 16 illustrate the transmission time of R3 in the traditional network and SDN load balancing, respectively. Figure 17 depicts a performance comparison of R3 between the traditional network and SDN load balancing. It can be noticed that the former decreases the network transmission time by approximately 48% as compared to the latter. The SDN load balancing also decreases the total completion time by approximately 14% in comparison to the traditional network.

The comparison of total data size and total completion time between R1 and R3 is shown in Figure 18. Figure 18a shows that R1 has a smaller data size because its replication factor is set to one. In Figure 18b, it can be observed that R1 has a shorter total completion time in both SDN load balancing and traditional network as compared to R3. Moreover, the replication factor is directly proportional to the total completion time.

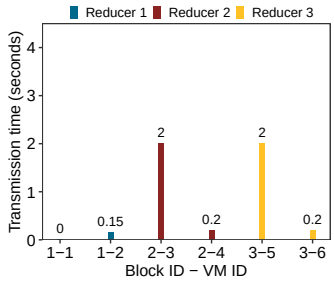
R1 and R3 illustrate an overview of BigDataSDNSim's modeling and features. They stress the significance of the simulator for testing the strengths and weaknesses of new solutions intended for optimizing the performance of MapReduce applications in SDN-enabled cloud environments. BigDataSDNSim shows a full picture of the possible states of hypotheses and new proposed solutions. By obtaining the results of every solution, individuals can easily identify and address hidden issues along with ensuring the optimal performance of their approaches and algorithms.



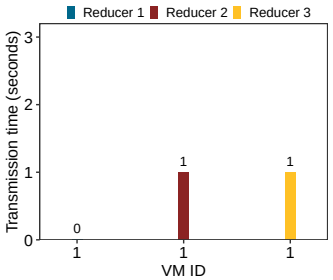
(a) Transmission time of blocks from HDFS to the VMs of mappers (R1)



(b) Transmission time of intermediate data from every mapper to all reducers (R1)

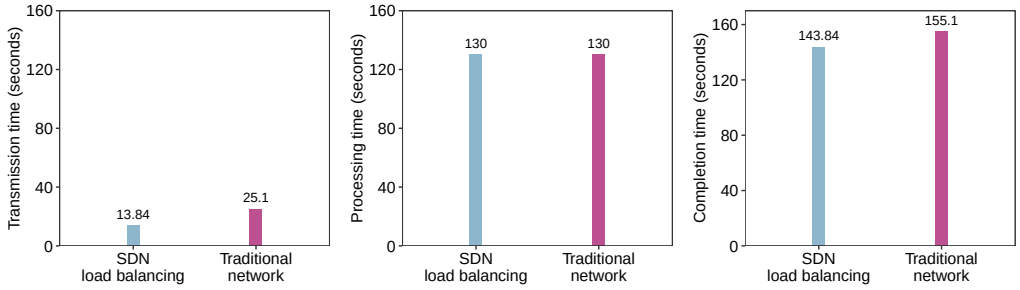


(c) Transmission time of blocks from reducers to elected VMs (R1)



(d) Transmission time of final outputs from reducers to the main VM (R1)

FIGURE 12 Transmission time of MapReduce in an SDN load balancing network (R1)

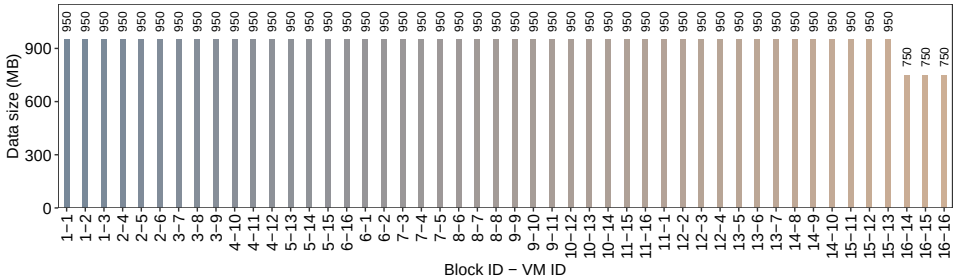


(a) Total transmission time **(b)** Total processing time **(c)** Total completion time

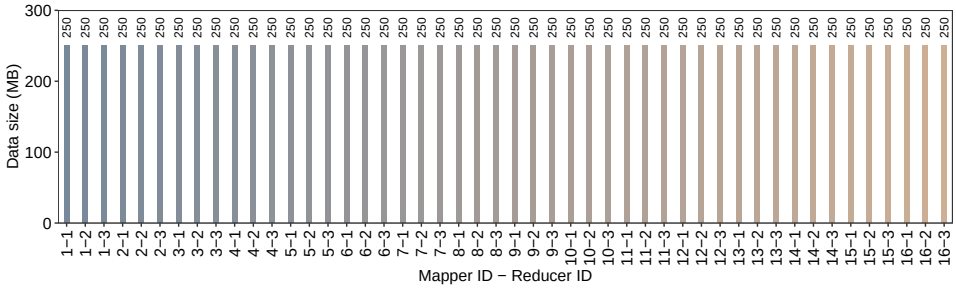
FIGURE 13 Performance comparison of MapReduce in a traditional network and SDN load balancing (R1)

8 | CONCLUSIONS AND FUTURE WORK

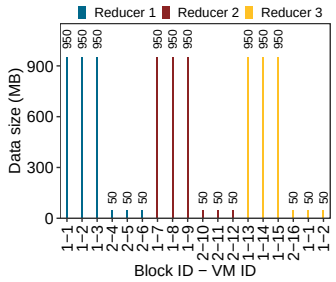
The advent of the MapReduce programming model, BDMS, and clouds has contributed to improving the existing practice of data analysis and synthesis. The architecture and performance of MapReduce, which is managed by a BDMS, depends on



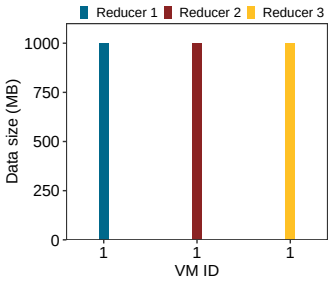
(a) Block size transferred from HDFS to elected VMs (R3)



(b) The output size of every mapper transferred to every reducer (R3)



(c) The output block size of every reducer transferred to elected VMs (R3)

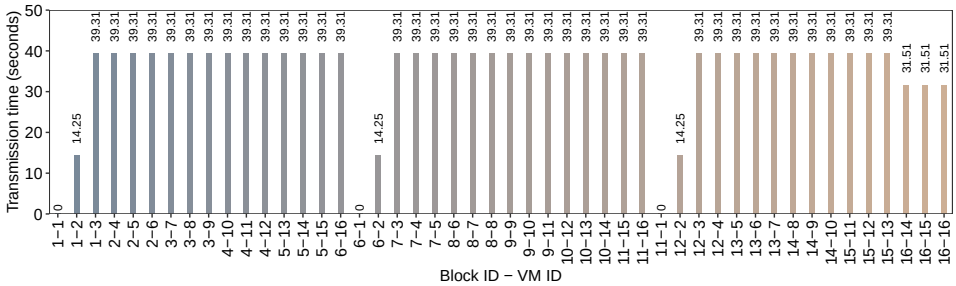


(d) The total output size of every reducer transferred to the main VM (R3)

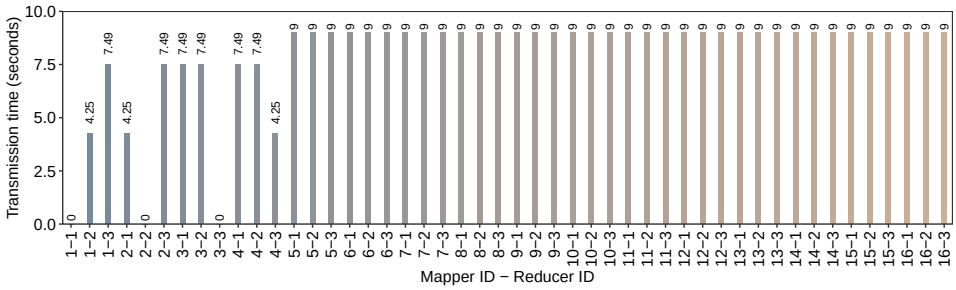
FIGURE 14 Data size of three replications (R3)

two major factors: processing and network transmission. Most of the existing solutions inform design decisions in terms of MapReduce processing performance and neglects the transmission performance due to the lack of real-time, dynamic network configurations. The emergence of SDN has enabled numerous applications, including big data applications, to jointly optimize the performance of processing and transmission. Simulation-based approaches can be significant tools to test and trace the strengths of proposed MapReduce SDN-powered solutions and techniques. As the use of simulation-based approaches has been widely applied in numerous fields for analyzing new hypotheses and solutions along with raising awareness of hidden dilemmas, this paper presents BigDataSDNSim: a novel simulation-based tool that is capable of simulating and evaluating the performance of big data applications in SDN-enabled cloud data centers.

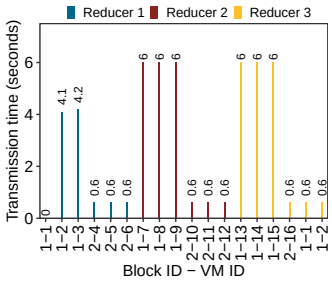
The objective of our simulator is to offer holistic modeling and integration of MapReduce BDMS-based models that are compatible with SDN network functions in cloud infrastructures. BigDataSDNSim provides an infrastructure for researchers to quantify the performance impacts of MapReduce applications in terms of a joint-design of host and network. It contains a variety of application-network policies for diverse purposes (e.g., scheduling and routing), which can be seamlessly extended without a deep understanding of the complex interactions among BigDataSDNSim's components. In order to demonstrate the correctness and accuracy of our simulator, the performance of BigDataSDNSim is validated with a real MapReduce SDN-enabled environment. The validation measures the performance similarities of BigDataSDNSim with the real environment. It reports the comparison results of bandwidth, transmission time, processing time, total completion time, and correlation. Validation results



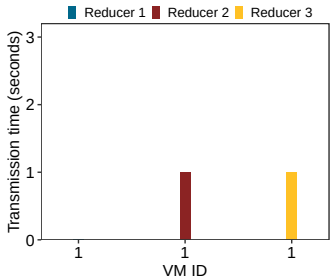
(a) Transmission time of blocks from HDFS to the VMs of mappers (R3)



(b) Transmission time of intermediate data from every mapper to all reducers (R3)



(c) Transmission time of blocks from reducers to elected VMs (R3)



(d) Transmission time of final outputs from reducers to the main VMs (R3)

FIGURE 15 Transmission time of MapReduce in a traditional network (R3)

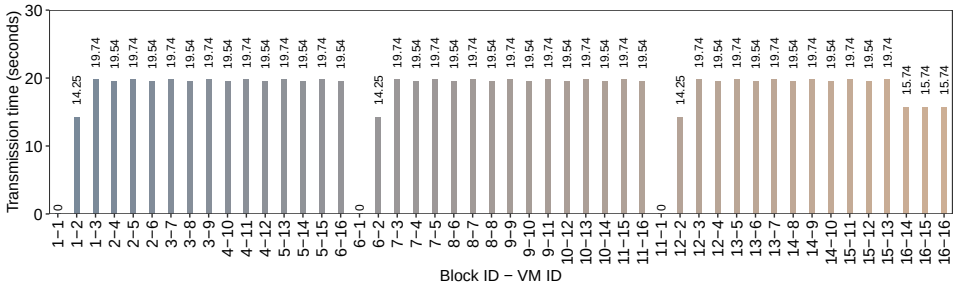
reveal that BigDataSDNSim simulation results are closely comparable to results obtained from real MapReduce SDN-enabled environments.

The practicality and advantages of using BigDataSDNSim are demonstrated by presenting two use cases. The use cases focus on MapReduce performance in terms of the impact of using HDFS replication mechanisms and the advantages of using SDN. The performance impacts of SDN load balancing versus traditional networks on MapReduce applications in cloud data centers are illustrated. The results of the simulated experiments confirm the SDN load balancing decreases the total completion time of MapReduce applications as compared to the traditional networks. As our future work, we will focus on the modeling and simulating of stream paradigms in the context of big data SDN-powered cloud environments.

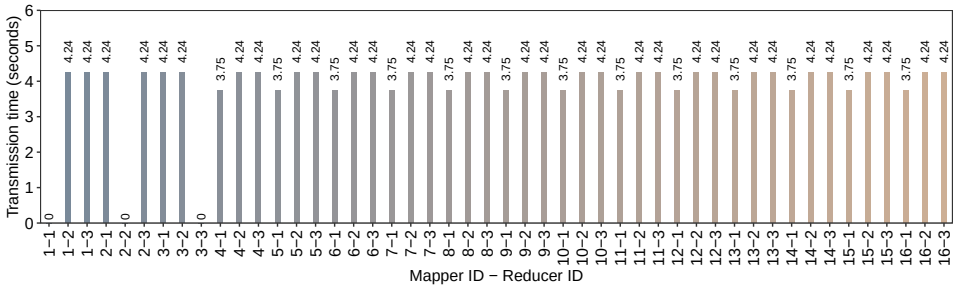
Software availability: The BigDataSDNSim software with the source code can be downloaded from https://github.com/kalwasel/BigDataSDNSim. A number of examples and tutorials illustrating the use of BigDataSDNSim are given on the web site.

ACKNOWLEDGMENTS

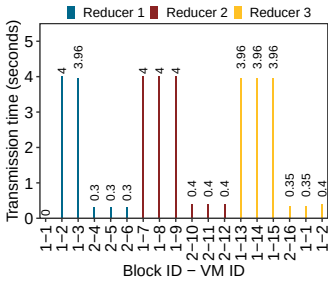
The work in this paper is supported by Saudi Electronic University (SEU) through the Saudi Arabian Culture Bureau (SACB) in the United Kingdom. This research is also supported by three UK projects LANDSLIP: NE/P000681/1, Flood-Prep:NE/P017134/1, and PACE: EP/R033293/1.



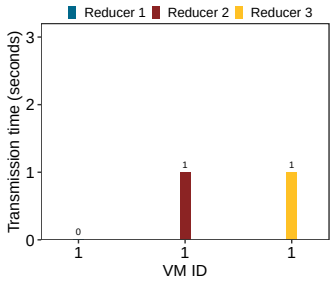
(a) Transmission time of blocks from HDFS to the VMs of mappers (R3)



(b) Transmission time of intermediate data from every mapper to all reducers (R3)



(c) Transmission time of blocks from reducers to elected VMs (R3)

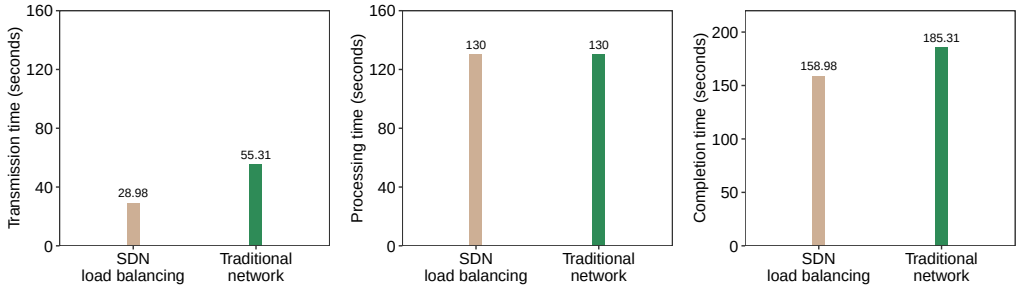


(d) Transmission time of final outputs from reducers to the main VMs (R3)

FIGURE 16 Transmission time of MapReduce in SDN load balancing network (R3)

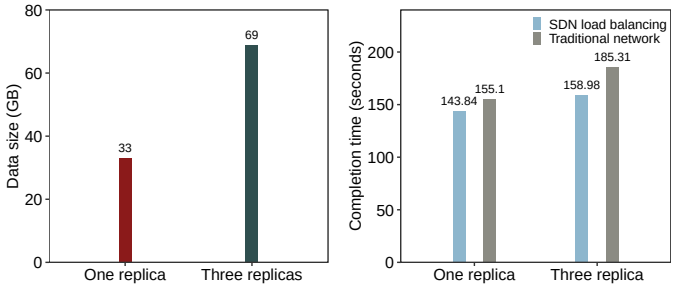
References

1. Chen Min, Mao Shiwen, Liu Yunhao. Big data: a survey. *Mobile networks and applications.* 2014;19(2);171–209.
2. Kambatla Karthik, Kollias Giorgos, Kumar Vipin, Grama Ananth. Trends in big data analytics. *Journal of Parallel and Distributed Computing.* 2014;74(7);2561–2573.
3. Dean Jeffrey, Ghemawat Sanjay. MapReduce: simplified data processing on large clusters. 2004;.
4. Iqbal Muhammad Hussain, Soomro Tariq Rahim. Big data analysis: Apache Storm perspective. *International journal of computer trends and technology.* 2015;19(1);9–14.
5. Ranjan Rajiv. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing.* 2014;1(1);78–83.
6. Vavilapalli Vinod Kumar, Murthy Arun C, Douglas Chris, et al. Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing; 2013; Santa Clara, California, USA.
7. Ahmad Faraz, Chakradhar Srimat T, Raghunathan Anand, Vijaykumar TN. Tarazu: optimizing MapReduce on heterogeneous clusters. *ACM SIGARCH Computer Architecture News.* 2012;40(1);61–74.



(a) Total transmission time (R3) **(b)** Total processing time (R3) **(c)** Total completion time (R3)

FIGURE 17 Performance comparison of MapReduce between a traditional network and SDN load balancing (R3)



(a) Total data size **(b)** Total completion time

FIGURE 18 Comparison between R1 and R3

8. Fischer Michael J, Su Xueyuan, Yin Yitong. Assigning tasks for efficiency in Hadoop. In: Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures;30–39; 2010.

9. Alwasel Khaled, Li Yinhao, Jayaraman Prem Prakash, Garg Saurabh, Calheiros Rodrigo N, Ranjan Rajiv. Programming SDN-native big data applications: research gap analysis. *IEEE Cloud Computing*. 2017;4(5);62–71.

10. Cui Laizhong, Yu F Richard, Yan Qiao. When big data meets software-defined networking: SDN for big data and big data for SDN. *IEEE network*. 2016;30(1);58–65.

11. Cheng Dazhao, Rao Jia, Jiang Changjun, Zhou Xiaobo. Resource and deadline-aware job scheduling in dynamic Hadoop clusters. In: 2015 IEEE International Parallel and Distributed Processing Symposium;956–965IEEE; 2015.

12. Hashem Ibrahim Abaker Targio, Anuar Nor Badrul, Marjani Mohsen, et al. MapReduce scheduling algorithms: a review. *The Journal of Supercomputing*. 2018;;1–31.

13. Stone Harold S.. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE transactions on Software Engineering*. 1977;(1);85–93.

14. Dijkstra Edsger W. A note on two problems in connexion with graphs. *Numerische mathematik*. 1959;1(1);269–271.

15. Holland John Henry, others . *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press; 1992.

16. Chen Quan, Zhang Daqiang, Guo Minyi, Deng Qianni, Guo Song. Samr: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: 2010 10th IEEE International Conference on Computer and Information Technology;2736–2743IEEE; 2010.

17. Jha Devki Nandan, Michalak Peter, Wen Zhenyu, Watson Paul, Ranjan Rajiv. Multi-objective deployment of data analysis operations in heterogeneous IoT infrastructure. *IEEE Transactions on Industrial Informatics.* 2019;.

18. Kreutz Diego, Ramos Fernando MV, Verissimo Paulo Esteves, Rothenberg Christian Esteve, Azodolmolky Siamak, Uhlig Steve. Software-defined networking: a comprehensive survey. *Proceedings of the IEEE.* 2015;103(1);14–76.

19. McKeown Nick, Anderson Tom, Balakrishnan Hari, et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review.* 2008;38(2);69–74.

20. Qin Peng, Dai Bin, Huang Benxiong, Xu Guan. Bandwidth-aware scheduling with SDN in Hadoop: A new trend for big data. *IEEE Systems Journal.* 2015;11(4);2337–2344.

21. Zhao Shuai, Sydney Ali, Medhi Deep. Building application-aware network environments using SDN for optimizing Hadoop applications. In: Proceedings of the 2016 ACM SIGCOMM Conference;583–584; 2016.

22. Anadiotis Angelos-Christos G, Morabito Giacomo, Palazzo Sergio. An SDN-assisted framework for optimal deployment of MapReduce functions in WSNs. *IEEE Transactions on Mobile Computing.* 2015;15(9);2165–2178.

23. Calheiros Rodrigo N, Ranjan Rajiv, Beloglazov Anton, De Rose César AF, Buyya Rajkumar. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience.* 2011;41(1);23–50.

24. Jha Devki Nandan, Alwasel Khaled, Alshoshan Areeb, et al. IoTSim-Edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. *Software: Practice and Experience.* ;.

25. Kecskemeti Gabor, Casale Giuliano, Jha Devki Nandan, Lyon Justin, Ranjan Rajiv. Modelling and simulation challenges in internet of things. *IEEE cloud computing.* 2017;4(1);62–69.

26. Garg Saurabh Kumar, Buyya Rajkumar. NetworkCloudSim: modelling parallel applications in cloud simulations. In: Utility and Cloud Computing (UCC), Fourth IEEE International Conference on; 2011; Victoria, NSW, Australia.

27. Chen Weiwei, Deelman Ewa. WorkflowSim: a toolkit for simulating scientific workflows in distributed environments. In: E-science (e-science), IEEE 8th International Conference on; 2012; Chicago, IL, USA.

28. Kliazovich Dzmitry, Bouvry Pascal, Khan Samee Ullah. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing.* 2012;62(3);1263–1283.

29. Núñez Alberto, Vázquez-Poletti Jose L, Caminero Agustin C, Castañé Gabriel G, Carretero Jesus, Llorente Ignacio M. iCanCloud: a flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing.* 2012;10(1);185–209.

30. Lantz Bob, Heller Brandon, McKeown Nick. A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks; 2010; Monterey, California, USA.

31. Wette Philip, Dräxler Martin, Schwabe Arne, Wallaschek Felix, Zahraee Mohammad Hassan, Karl Holger. Maxinet: distributed emulation of software-defined networks. In: 2014 IFIP Networking Conference;1–9IEEE; 2014.

32. Neves Marcelo Veiga, De Rose Cesar AF, Katrinis Kostas. MRemu: an Emulation-based Framework for Datacenter Network Experimentation using Realistic MapReduce Traffic. In: Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE 23rd International Symposium on; 2015; Atlanta, GA, USA.

33. Wang Shie-Yuan, Chou Chih-Liang, Yang Chun-Ming. EstiNet OpenFlow network simulator and emulator. *IEEE Communications Magazine.* 2013;51(9);110–117.

34. Son Jungmin, Dastjerdi Amir Vahid, Calheiros Rodrigo N, Ji Xiaohui, Yoon Young, Buyya Rajkumar. CloudSimSDN: modeling and simulation of software-defined cloud data centers. In: Cluster, Cloud and Grid Computing (CCGrid), 15th IEEE/ACM International Symposium on; 2015; Shenzhen, China.

35. Kathiravelu Pradeeban, Veiga Luís. Software-defined simulations for continuous development of cloud and data center networks. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems";3–23Springer; 2016.

36. Ghosh Saptarshi, Busari SA, Dagiuklas Tasos, et al. SDN-Sim: integrating a system-level simulator with a software defined network. *IEEE Communications Standards Magazine.* 2020;4(1);18–25.

37. Tüxen Michael, Rüngeler Irene, Rathgeb Erwin P. Interface connecting the INET simulation framework with the real world. In: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops;1–6; 2008.

38. Shalimov Alexander, Zuikov Dmitry, Zimarina Daria, Pashkov Vasily, Smeliansky Ruslan. Advanced study of SDN/OpenFlow controllers. In: Proceedings of the 9th central & eastern european software engineering conference in russia;1–6; 2013.

39. Zeng Xuezhi, Garg Saurabh Kumar, Strazdins Peter, Jayaraman Prem Prakash, Georgakopoulos Dimitrios, Ranjan Rajiv. IOTSim: a simulator for analysing IoT applications. *Journal of Systems Architecture.* 2017;72;93–107.

40. Jung Jongtack, Kim Hwangnam. MR-CloudSim: designing and implementing MapReduce computing model on CloudSim. In: ICT Convergence (ICTC), International Conference on; 2012; Jeju Island, South Korea.

41. Hammoud Suhel, Li Maozhen, Liu Yang, Alham Nasullah Khalid, Liu Zelong. MRSim: a discrete event based MapReduce simulator. In: Fuzzy Systems and Knowledge Discovery (FSKD), Seventh International Conference on; 2010; Yantai, China.

42. Wang Guanying, Butt Ali R, Pandey Prashant, Gupta Karan. Using realistic simulation for performance analysis of MapReduce setups. In: Proceedings of the 1st ACM workshop on Large-Scale system and application performance; 2009; Garching, Germany.

43. Almeida Leandro Batista, Almeida Eduardo Cunha, Murphy John, Robson E, Ventresque Anthony. BigDataNetSim: a simulator for data and process placement in large big data platforms. In: 2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT);1–10IEEE; 2018.

44. Calcaterra Claudio, Carmenini Alessio, Marotta Andrea, Bucci Ubaldo, Cassioli Dajana. MaxHadoop: an Efficient scalable emulation tool to test SDN protocols in emulated Hadoop environments. *Journal of Network and Systems Management.* 2020;;1–29.

45. Huang Shengsheng, Huang Jie, Dai Jinquan, Xie Tao, Huang Bo. The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010);41–51IEEE; 2010.

46. Pfaff Ben, Davie Bruce. The open vSwitch database management protocol. *Internet Requests for Comments, RFC Editor, RFC.* 2013;7047.

47. Tomonori FUJITA. Introduction to Ryu SDN framework. *Open Networking Summit.* 2013;;1–14.

48. Al-Fares Mohammad, Loukissas Alexander, Vahdat Amin. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review.* 2008;38(4);63–74.

49. Guo Chuanxiong, Lu Guohan, Li Dan, et al. BCube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review.* 2009;39(4);63–74.

50. Teixeira José, Antichi Gianni, Adami Davide, Del Chiaro Alessio, Giordano Stefano, Santos Alexandre. Datacenter in a box: test your SDN cloud-datacenter controller at home. In: Software Defined Networks (EWSDN), Second European Workshop on; 2013; Berlin, Germany.

51. Nardelli Matteo, Nastic Stefan, Dustdar Schahram, Villari Massimo, Ranjan Rajiv. Osmotic flow: Osmotic computing+ IoT workflow. *IEEE Cloud Computing.* 2017;4(2);68–75.

52. Gubbi Jayavardhana, Buyya Rajkumar, Marusic Slaven, Palaniswami Marimuthu. Internet of things (IoT): a vision, architectural elements, and future directions. *Future generation computer systems.* 2013;29(7);1645–1660.

53. DeCandia Giuseppe, Hastorun Deniz, Jampani Madan, et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review.* 2007;41(6);205–220.

How to cite this article: Alwasel K., Calheiros R.N., Garg S., Buyya R., Pathan M., and Ranjan R. BigDataSDNSim: A Simulator for Analyzing Big Data Applications in Software-Defined Cloud Data Centers, *Softw Pract Exper*, *2018;00:1–6*.