SDCon: Integrated Control Platform for Software-Defined Clouds

Jungmin Son, *Student Member, IEEE* and Rajkumar Buyya, *Fellow, IEEE*

Abstract—Cloud computing has been empowered with the introduction of Software-Defined Networking (SDN) which enabled dynamic controllability of cloud network infrastructure. Despite the increasing popularity of studies for joint resource optimization in the cloud environment with SDN technology, the realization is still limited for developing integrated management platform providing a simultaneous controllability of computing and networking infrastructures. In this paper, we propose *SDCon*, a practical platform developed on OpenStack and OpenDaylight to provide integrated manageability for both resources in cloud infrastructure. The platform can perform VM placement and migration, network flow scheduling and bandwidth allocation, real-time monitoring of computing and networking resources, and measuring power usage of the infrastructure with a single platform. We also propose a network topology aware VM placement algorithm for heterogeneous resource configuration (TOPO-Het) that consolidates the connected VMs into closely connected compute nodes to reduce the overall network traffic. The proposed algorithm is evaluated on SDCon and compared with the results from the state-of-the-art baselines. Results of the empirical evaluation with Wikipedia application show that the proposed algorithm can effectively improve the response time while reducing the total network traffic. It also shows the effectiveness of SDCon to manage both resources simultaneously.

Index Terms—Cloud computing, software-defined networking, resource management, software-defined clouds, IaaS, OpenStack, OpenDaylight

1 Introduction

The emergence of software-defined networking (SDN) has brought many opportunities and challenges to both networking and computing community. Decoupling of the network control logic from data forwarding plane is a key innovation which contributes to the introduction of network programmability. With the global view of the entire network, the centralized controller can manage the network through the customized control logic programmed for an individual use case. SDN can slice the network bandwidth into multiple layers and provide different quality of service (QoS) on each slice for different applications. Network function virtualization (NFV) becomes more feasible with the introduction of SDN, where the virtualized network function can move around the network dynamically with SDN's network reconfiguration functionality.

SDN also triggers new innovations in cloud computing. The massive scale of a cloud data center where tens of thousands of compute nodes connected through thousands of network switches raises network manageability and performance issues. Cloud data centers also need to provision computing and network resources dynamically to adapt to the fluctuating requests from customers. Adoption of SDN in a cloud data center can address the networking issues raised from the aforementioned characteristic of clouds. Recently, researchers have introduced a new terminology, software-defined clouds (SDC), that integrates SDN and other SDx concepts in cloud computing where the programmable controller provides dynamic and autonomic configuration and management of the underlying physical layer, similar to the network controller managing the underlying forwarding plane in SDN [1], [2].

Since both cloud computing and SDN have been widely accepted in the industry and open-source community, many platforms are introduced and developed in each area. For cloud management, various full-stack softwares have been developed by companies and open-source communities, such as OpenStack, VMWare, and Xen Cloud Platform. For instance, OpenStack is an open source cloud management platform that has been employed by many institutions for years to run their private clouds. It provides manageability and provisioning for computing, networking, and storage resources based on the users and projects. OpenStack supports network virtualization on compute nodes for virtual machines (VM) so that cloud tenants can create their own virtual network to communicate between leased VMs.

SDN controllers are also actively developed by open source communities including OpenDaylight, Floodlight, NOX, Open Network Operating System, and Ryu. OpenDaylight is an open source SDN controller based on a modular structure which has been actively developed under The Linux Foundation. OpenDaylight provides comprehensive functionalities through its modular plug-ins, such as OpenFlow switch management, OpenVSwitch database management,

• *The authors are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, VIC 3010, Australia.*
E-mail: jungmins@student.unimelb.edu.au, rbuyya@unimelb.edu.au.

Manuscript received 12 Dec. 2017; revised 6 July 2018; accepted 8 July 2018. Date of publication 11 July 2018; date of current version 12 Dec. 2018.
(Corresponding author: Jungmin Son.)
Recommended for acceptance by R. Prodan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2855119

network virtualization, group-based policy, and so on. Users can select features to install for their purpose. Other controllers provide similar functions with different architecture and programming languages.

Although the platforms are matured in SDN and cloud computing individually, it is difficult to find a software platform in practice for integration of SDN and clouds to jointly control both networking and computing devices for clouds. For example, OpenStack adopts OpenVSwitch,[1] a software virtual switch compatible with SDN, in its networking module to provide network virtualization of VMs. However, in OpenStack, OpenVSwitch is created and managed only for compute nodes to establish virtual tunnels with other compute nodes. OpenStack does not provide any feature to control the network fabric of the cloud; network switches in a data center cannot be controlled by OpenStack's internal network controller (Neutron). Similarly, OpenDaylight can manage the virtual network of OpenStack clouds using its NetVirt feature, but it controls virtual switches on compute nodes separately from network switches that connect compute nodes. OpenDaylight and other SDN controllers do not support integrated controllability of both compute nodes and network switches. Cisco developed Unified Computing System,[2] on the other hand, integrates computing and networking resource management into a single platform to manage Cisco's servers and networking switches. While it provides integrated controllability for cloud infrastructure, only Cisco's products can be managed by the platform which is in lack of the openness and innovative features of SDN.

In this paper, we propose an integrated control platform, named *SDCon* (Software-Defined Clouds Controller),[3] which can jointly manage both computing and networking resources in the real world in the perspective of a cloud provider. SDCon is implemented on top of the popular cloud and SDN controller software: OpenStack and OpenDaylight. It is designed to support various controllers with the implementation of separate driver modules, but for simplicity of development, we adopted OpenStack and OpenDaylight as underlying software. We integrate cloud's computing fabric and SDN's network fabric controllability into a combined SDC controller. In addition, we propose TOPO-Het, a topology-aware resource allocation algorithm in heterogeneous configuration, in order to demonstrate the effectiveness and capabilities of SDCon platform in empirical systems.

The key *contributions* of the paper are:

- the design of a practical SDC controller that integrates the controllability of network switches with the management of computing resources;
- the implementation of the integrated SDC control system on a small-scale testbed upon existing software solutions with 8 compute nodes connected through 2-pod modified fat-tree network topology;
- a topology aware joint resource provisioning algorithm for heterogeneous resource configuration (TOPO-Het);
- an evaluation of the joint resource provisioning algorithm on the testbed with the proposed SDC control platform.

The rest of the paper is organized as follows: Section 2 provides the relevant literature studied and developed for SDN integration platform for cloud computing. In Section 3, we depict the design concept of the proposed platform and the control flows between modules and external components, followed by the detailed explanation of the implementation method and the functionalities of each component of SDCon in Section 4. The heterogeneity and network topology aware VM placement algorithm is proposed in Section 5 in addition to the explanation of baseline algorithms. Section 6 provides the validation results of SDCon with bandwidth measurements, and Section 7 shows the evaluation of SDCon and the proposed algorithm with Wikipedia application by comparing with baselines. Finally, Section 8 concludes the paper along with directions for future work.

2 Related Work

Increasing number of studies have investigated the joint provisioning of networking and computing resources in clouds [3], [4], [5], [6], [7], [8], in which experiments have been conducted either in simulation environment [3], [4], or on their in-house customized empirical system [5], [6], [7], [8]. In this section, we review the recent studies in the integration of SDN and clouds and the joint resource provisioning algorithms.

2.1 Integrated Platform of SDN and Clouds

We previously proposed SDN-enabled cloud computing simulator, CloudSimSDN, to enable large-scale experiment of SDN functionality in cloud computing [9]. CloudSimSDN can simulate various use-case scenarios in cloud data centers with the support of SDN approaches, such as dynamic bandwidth allocation, dynamic path routing, and central view and control of the network. Although simulation tools are useful to evaluate the impact of new approaches in a large-scale data center, there are still significant gaps between the simulation and real implementation.

Mininet [10] has gained a great popularity for SDN emulation to experiment practical OpenFlow controllers. Any SDN controllers supporting OpenFlow protocol can be tested with Mininet, where a customized network topology with multiple hosts can be created and used for several evaluations, such as measuring bandwidth utilization with the *iperf*[4] tool. Although Mininet opened up great opportunities in SDN studies, a lack of supporting multi-interface emulation limited its usage in cloud computing studies. Because researchers need to experiment with virtual machines in hosts, it is impractical to use Mininet to test common cloud scenarios such as VM migration or consolidation in a data center.

To overcome the shortcoming of Mininet, OpenStackEmu is proposed to integrate OpenStack with a large-scale network emulator named CORE (Common Open Research Emulator) [11]. The proposed system combines OpenStack platform with a network emulator to perform evaluations considering both cloud and networking. All network switches are

1. http://openvswitch.org
2. http://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/unified-computing/at_a_glance_c45-523181.pdf
3. Source code available at: http://github.com/cloudslab/sdcon
4. http://iperf.fr

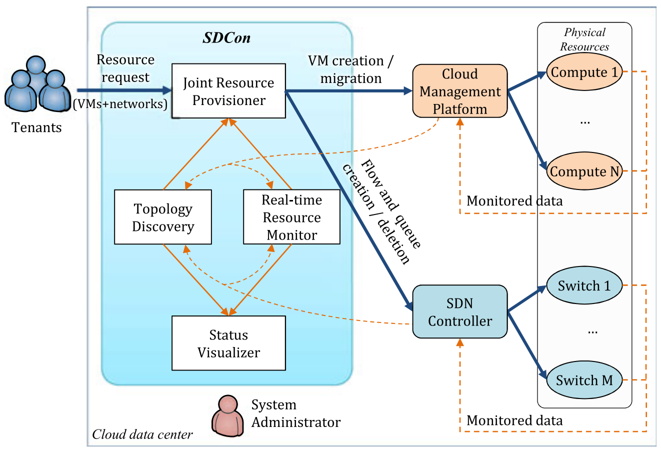


Fig. 1. Design principle and control flows.

emulated in one physical machine with the capability of Linux software bridges and OpenVSwitch. A real SDN controller manages the emulated switches on which transfer network frames from physical compute nodes. OpenStackEmu is a useful approach to build an SDN-integrated cloud testbed infrastructure with limited budget and resources, but the system does not provide an integrated control platform for both OpenStack and SDN. In our approach, we propose a joint control platform which can even run on OpenStackEmu infrastructure to control its networking and computing resources.

In their recent paper, Cziva et al. proposed S-SCORE, a VM management and orchestration system for live VM migration to reduce the network cost in a data center [7]. This platform is extended from Ryu SDN controller and includes VM management function by controlling hypervisors (Libvirt) in compute nodes. The authors implemented the system on the canonical tree topology testbed with eight hosts and showed the effectiveness of their migration algorithm to reduce the overall VM-to-VM network cost by moving VMs into closer hosts. The proposed system is suitable for studies focusing on networking resources, but is lack of optimizing computing resources such as CPU utilization of compute nodes.

Adami et al. also proposed an SDN orchestrator for cloud data centers based on POX as an SDN controller and Xen Cloud Platform as a VM management platform [12], [13], [14], [15]. The system provides a web portal for end users to submit VM requests which are handled by resource selection and composition engine in the core of the system. The engine utilizes virtual machine and OpenFlow rule handlers to configure VMs in hypervisors and flow tables in switches respectively. Similar to S-SCORE, this system focuses more on networking aspects of clouds without a comprehensive consideration of computing resources such as VM and hosts utilization and virtual networks for VMs.

2.2 Joint Resource Provisioning Algorithms

Many studies considering resource heterogeneity in clouds have focused on the level of brokers for different providers or geographically distributed data centers. For example, VM placement algorithm across different providers was studied to reduce the leasing cost of virtual machines while providing the same level of throughput from the hosted application [16]. Recently, a renewable-aware load balancing algorithm across geologically distributed data centers was proposed to increase the sustainability of data centers [17]. This algorithm selects a data center operated by more portion of renewable power sources (e.g., data center sourced by a solar power plant in a clear day) and places more VMs on the data center to reduce the carbon footprint. Also, researchers studied the resource optimization within a data center considering heterogeneity to reduce the energy consumption of the data center [18]. The on-line deterministic algorithm consolidates VMs dynamically into the smaller number of compute nodes and switches off the idle nodes to save electricity.

VM management method considering network topology was proposed by Cziva et al. [7] that migrates a VM with a high level of network usage onto the destination host to reduce the network cost for VMs and the traffic over the data center. For a VM, the algorithm calculates a current communication cost for each VM-to-VM flow and estimates a new communication cost if the VM is migrated onto the other end of the flow. When it finds a new location for the VM which can reduce the total communication cost, the VM is migrated to the other host. However, the approach does not consider the capacity of compute nodes, thus it may not migrate a large VM if the available computing resource of the selected host is not enough for the VM. Also, this algorithm limits migration target candidates by considering only the compute node that hosts the other VM of the flow. If no connected VM is placed in a host, it is not considered as a migration target, which in practice can be a proper destination if all the candidates in the first place cannot host the VM because of the resource limitation. Our algorithm deals with this limitation by considering the group of hosts, instead of an individual host, as a candidate of VM placement.

Although the heterogeneity in the cloud environment has been addressed in these studies, they are either considering only high-level entities (e.g., different data centers or providers) or only computing resources within a data center without considering joint optimization for both computing and networking resources in heterogeneous infrastructure in a data center.

3 SDCon: Software-Defined Clouds Controller

SDCon is designed to provide integrated controllability for both clouds and networks and implemented with popular software widely used in practice. The conceptual design and the control flows between components are shown in Fig. 1. As described in the previous section, cloud management platforms and SDN controller software have been developed and widely adopted for years. Therefore, in this study, we designed our platform on top of those mature software platforms. Cloud Management Platform, e.g., OpenStack, manages computing resources including CPU, memory, and storage which are provided by multiple underlying compute nodes. Monitored data, such as CPU utilization of VMs and the compute node, is sent back to Cloud Management Platform from each compute node, so that the information can be used for resource provisioning and optimization. Similarly, networking resources are controlled by SDN Controller, e.g., OpenDaylight. Switches are connected and managed by the SDN Controller, and the network utilization monitored in

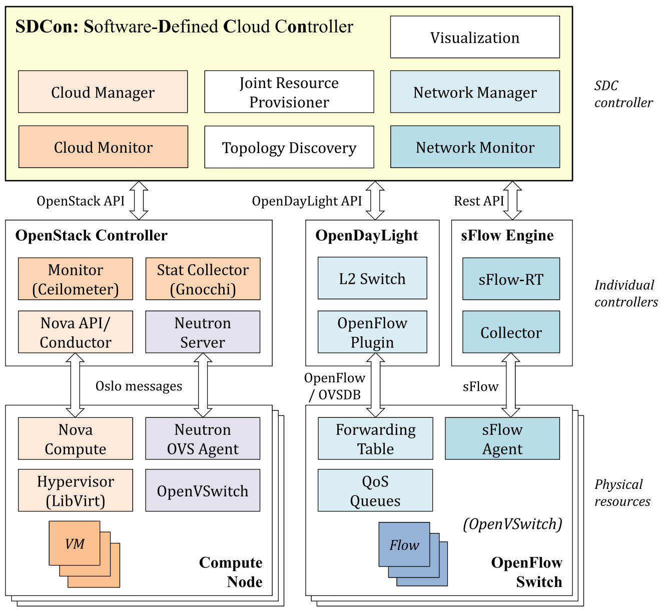


Fig. 2. System architecture of SDCon and underlying software components.

each switch is gathered by the controller. Both Cloud Management Platform and SDN Controller are deployed and properly configured with existing software solutions.

Based on the separate platform for cloud management and SDN control, SDCon is designed to manage and monitor both of them jointly. When tenants request resources from a cloud data center, Joint Resource Provisioner in SDCon accepts the request and controls both Cloud Management Platform and SDN Controller simultaneously to provide the required amount of resources. In the traditional cloud platform, tenants can specify only computing resources in detail, such as the number of CPU cores, the amount of memory and storage. With SDCon, we introduce additional configuration in the resource request regarding networking requirements, such as requested bandwidth between VMs, which can be specified along with the type of VMs. Joint Resource Provisioner is in charge of not only allocating the requested resources for tenants, but also optimizing both computing and networking resources based on the optimization policy provided by the system administrator of the cloud provider. For example, it can migrate low-utilized VMs and flows into the smaller number of hardware and power off the unused resources to increase power efficiency of the cloud data center. System administrators in the data center can also set a customized default path policy for the network, especially effective for multi-path network topology.

Resource provisioning and optimization at Joint Resource Provisioner is performed based on the network topology and real-time monitored data acquired from Cloud Management Platform and SDN Controller. Topology Discovery receives the network topology and its capacity from SDN Controller and the hardware specification of compute nodes from Cloud Management Platform. Real-time Resource Monitor also gathers the monitored data from both cloud and SDN controller. Contrary to Topology Discovery, Resource Monitor keeps pulling measurements from compute nodes and switches periodically to update real-time resource utilization, including CPU utilization of VMs and compute nodes, and bandwidth utilization of network links. Please note that the pulling interval should be properly configured by the administrator considering the size of monitored data and the number of monitoring compute nodes and switches to prevent the system overload.

In addition to the resource provisioning, SDCon supports a visualization of computing and networking resources through Status Visualizer components. Based on the topology information and monitored measurements, the visualizer displays current utilization of network links and compute nodes in real-time. System administrators can check the status of all resources on a graphical interface. In the following subsection, we explain the implementation of the proposed system in detail.

4 System Implementation

We implemented SDCon by utilizing OpenStack cloud management platform and OpenDaylight SDN controller to control computing and networking resources respectively. Fig. 2 shows the implemented architecture of SDCon platform. The system is implemented with Python, on top of OpenStack Python SDK,[5] Gnocchi Python API,[6] OpenDaylight OpenFlow plugin[7] and OVSDB plugin,[8] and sFlow-RT REST API.[9] SDCon utilizes these existing open-source software solutions to provide integrated controllability and optimization to manage both computing and networking resources. Please note that another software can be adopted in SDCon by modifying the relevant API.

OpenStack platform is exploited to manage compute nodes. Compute nodes are running OpenStack Nova-compute component to provide a bridge between the hypervisor and the OpenStack controller. VM creation, migration, or deletion is managed by Nova-API and Nova-Conductor components in the controller through messages passed to Nova-compute component in each compute node. Aside from Nova components, OpenStack Neutron is in charge of managing virtual networks for VMs. Neutron-OVS-Agent running on each compute node is creating and deleting virtual tunnels between compute nodes by updating OpenVSwitch rules, instructed by Neutron-Server in the controller. OpenStack Telemetry services are running on the controller to monitor compute nodes and VMs. OpenStack Ceilometer gathers computing resource measurements, such as CPU utilization and RAM usage, of both VMs and compute nodes. The monitored measurements are stored with Gnocchi component, which can be retrieved by its Rest API.

For networking resources, OpenDaylight controls OpenFlow switches that connect compute nodes. Each switch has OpenFlow forwarding tables managed by OpenDaylight L2-Switch and OpenFlow-Plugin modules which can install or modify forwarding rules for specific flows. OpenDaylight also manages QoS queues in a switch to enable dynamic

5. http://docs.openstack.org/openstacksdk/latest/
6. http://gnocchi.xyz/gnocchiclient/api.html
7. http://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Flows
8. http://docs.opendaylight.org/en/stable-carbon/user-guide/ovsdb-user-guide.html
9. http://www.sflow-rt.com/reference.php

bandwidth allocation. Due to the OpenFlow's limited support for QoS queues, queue management is provided through OpenDaylight's OpenVSwitch plug-in which can create and delete Hierarchical Token Bucket (HTB)[10] queues directly in switches. For network monitoring, our platform employed the sFlow protocol which can provide real-time measurements of the network by capturing sampled packets. sFlow agents are set up in switches and send sampled data to the sFlow engine running on the controller which gathers the data and provide the measurements through the Rest API.

On top of these services, SDCon is implemented to jointly manage both computing and networking resources. SDCon platform aims at integrating an SDN controller with a cloud manager to perform:

- allocating resources for VMs and flows with the information of applications and resources;
- monitoring both compute nodes and network switches for their utilization, availability, and capacity with the knowledge of network topology;
- dynamic provisioning of both computing and networking resources based on the real-time monitored matrices;
- providing management toolkit for system administrators to monitor power consumption and visualize the operational status of a data center.

Below, we explain the detail of each component in SDCon, data flows between components, and sequence diagram of VM allocation with SDCon.

4.1 Cloud Manager

Cloud Manager is in charge of controlling OpenStack Controller through OpenStack SDK. It can create or migrate a VM on a specific compute node by specifying the target node. Current allocation status of compute nodes is also retrieved through OpenStack so that SDCon can identify the number of used and available cores and memory in each node. This information is passed to Topology Discovery in order to keep the up-to-date resource allocation information in the topology database. It also provides other functions related to VMs and compute nodes, such as getting the IP address of a VM, VM types (Flavor), VM disk images, and virtual networks for VMs. Note that the virtual network configured in a compute node is managed by OpenStack Neutron through OpenVSwitch installed on compute nodes, whereas switches are controlled by OpenDaylight. Thus, the IP address of a VM is retrieved by Cloud Manager, not by Network Manager.

4.2 Cloud Monitor

Cloud Monitor is to retrieve real-time measurements of computing resources monitored at compute nodes and VMs. We use OpenStack Ceilometer and Gnocchi components to measure and collect the monitored data in the implementation. Ceilometer installed on each compute node measures CPU and memory utilization of all VMs and the compute node itself, and sends to Gnocchi that collects and aggregates the data from every compute node. By default, Gnocchi polls and stores the resource utilization measurement every 60 seconds from all VMs and compute nodes, which can be customized by changing Gnocchi configuration settings. Our Cloud Monitor uses Gnocchi Python API to retrieve the collected data, such as CPU utilization of both VMs and compute nodes. It is also utilized by Power Consumption Estimator to calculate the estimated power consumption of compute nodes, in which the detail is explained later.

4.3 Network Manager

In SDCon, OpenDaylight SDN Controller is managed by Network Manager module to push OpenFlow forwarding tables, default path setting for multi-path load balancing, and QoS configurations. In SDN, we can add a specific flow rule based on source and destination IP/MAC addresses, TCP/UDP port numbers, protocol types, and other criteria supported by OpenFlow standard. SDCon uses this feature to install or remove a flow rule onto switches through OpenDaylight OpenFlow plugin. This feature can be used for dynamic flow control or flow consolidation to change the path of a flow dynamically for performance improvement or power efficiency. We also implement a default path routing module enabling the load-balancing routing in fat-tree topology as suggested in the original paper [19]. Because the default L2 switch module in OpenDaylight only supports STP for multipath topology, we cannot use the benefit of multiple paths with the default module. Thus, we needed to implement our own routing module to set the default path between compute nodes based on the source address of the node. Our default path module installs flow rules on each switch to forward the packet to different ports by looking at source address if there are multiple paths. With the default path module, packets are forwarded to the pre-configured path based on the source node without flooding them to the entire network or dumping all traffic to the SDN controller.

SDCon's Network Manager is also capable of configuring QoS queues on each switch to provide bandwidth allocation for a specific flow if a resource request includes specific bandwidth requirements. For a priority flow which needs a higher priority over other traffics, a tenant can request a certain amount of bandwidth, then SDCon can allocate it by configuring minimum and maximum bandwidth in QoS queues on an output port in a switch. For example, OpenVSwitch supports traffic shaping functionality, similar to the *tc* tool in Linux, which can control the network bandwidth for different QoS level using HTB. We implement this QoS management feature in Network Manager. For VM-to-VM bandwidth allocation, Network Manager installs QoS queues on each port in every switch along the path passing the traffic. Network Manager at first aggregates all the requests for different source and destination VMs, because the flows have to be mapped to output ports simultaneously if the link is shared by multiple flows. Once the number of flows to be configured at each port in each switch is calculated, Network Manager sends QoS and queue creation request to OpenDaylight OVSDB plugin which controls OpenVSwitch entries on each switch. Then, we push OpenFlow rules for the specific flow in need of the bandwidth allocation, so that packets for the specific flow can be enqueued at the created QoS queue. We show the effectiveness of our bandwidth allocation method in Section 7.

10. http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.qdisc.classful.html

4.4 Network Monitor

Similar to Cloud Monitor, Network Monitor pulls a real-time network status from switches and compute nodes. We leverage sFlow protocol that collects sampled packets from sFlow agents in switches and sends them to the collector in the controller. By default, the sampling rate of sFlow is 1 in 500 for 100 Mbps link and 1 in 1000 for 1 Gbps link,[11] which sends a packet for every 500 packets passing the 100 Mbps link. For data collection and aggregation, sFlow-RT is installed in the controller node aside from the SDN controller. SDCon Network Monitor utilizes REST APIs provided by sFlow-RT to retrieve aggregated measurements of bandwidth usage and traffic flows on each link. Although OpenDaylight provides statistics of every port on OpenFlow switches, we decide to use sFlow because it provides more detailed information such as the source and destination address of the flow and encapsulated packets for tunneling protocols. Please note that the collected monitoring data from both Cloud and Network Monitor is stored in the monitor modules which can be accessed by SDCon at any time. Thus, the up-to-date information can be simultaneously provided from both monitors and utilized by SDCon.

4.5 Topology Discovery

Network topology information is retrieved from OpenDaylight through Network Manager. With the support of host tracker in OpenDaylight, SDCon can acknowledge compute nodes and switch devices on the network along with ports and links. Topology Discovery module internally generates a graph structure to store the topology information. In addition, it also retrieves the hardware configuration of compute nodes through Cloud Manager. With the retrieved information about computing resources, e.g., the number of CPU cores, size of RAM, etc., Topology Discovery can provide the detailed information to other modules. Unlike Cloud and Network Monitor modules which needs a periodic update for real-time data, Topology Discovery is created when the platform starts and updated only when there is an update on the topology.

4.6 Joint Resource Provisioner

The main control logic of SDCon resides in Joint Resource Provisioner module. When VM and flow requests are submitted from cloud tenants, Joint Resource Provisioner module decides which compute nodes to place the VM and which network path to transfer the VM flows. With the retrieved information of the topology information through Topology Discovery and real-time utilization via Cloud and Network Monitor, the decision is made with the provisioning algorithm provided by the system administrator. A customized VM allocation and migration policy can be implemented for VM provisioning, as well as a network flow scheduling algorithm for network provisioning.

Once the decision is made at the control logic, Joint Resource Provisioner exploits Cloud Manager and Network Manager, to create a VM on the selected node and push new flow entries respectively. For bandwidth allocation, QoS rules and queues are created in switches by updating OpenVSwitch configuration through Network Manager. Joint Resource Provisioner can utilize the real-time monitored data at any time whenever necessary. For example, a VM migration policy can be implemented upon Joint Resource Provisioner which constantly monitors the current status of VMs, compute nodes, and network bandwidth usage, then dynamically migrates a busy VM to less utilized resources in real time. If the migration policy is implemented, it can schedule the migration process by sending migration command to OpenStack through Cloud Manager.

4.7 Power Consumption Estimator

For supporting energy-related experiments, we additionally implement Power Consumption Estimator that calculates the power consumption of the infrastructure based on the utilization of compute nodes and switch links using an energy model. Although the best practice to measure the power consumption is to use a commercialized monitoring device, we add this feature to provide a simple alternative approximation method at no additional cost. Accumulated utilization data from both monitors is input to the energy model, and the estimated power consumption is calculated and output based on the energy model provided to SDCon.

For compute nodes, we use the linear power model derived from Pelly et al. [20] for its simplicity and ease of implementation. Please note that the power model can be replaced with other linear or quadratic models for more accurate estimates by changing the source code. In our implementation, the power consumption of the host h_i with the utilization u_i is calculated by:

$$P(h_i) = P_{idle}(h_i) + (P_{peak}(h_i) - P_{idle}(h_i)) \cdot u_i,$$

where P_{idle} refers to the power consumption of the host in idle state (0 percent utilization), and P_{peak} refers to the peak power consumption at 100 percent utilization. The power model of switches is derived from Wang et al. [21] which can be calculated by:

$$P(s_i) = P_{static}(s_i) + P_{port}(s_i) \cdot q_i,$$

where q_i refers to the number of active ports of the switch, P_{static} being the static power consumption for no network traffic, and P_{port} being per port power consumption of the switch.

4.8 Status Visualizer

Status Visualizer is implemented to intuitively visualize the current state of the data center. This module retrieves the topology information to draw the physical infrastructure with network links, and then draws the real-time monitored flows with different colors. We use D3 JavaScript library[12] to visualize the data onto web interface. By refreshing the page periodically, the Visualizer continuously updates the monitored measurements in real time.

4.9 Data Flow and Sequence Diagram

Fig. 3 depicts the overall data flow between components in SDCon. Once application deployment request is submitted to Joint Resource Provisioner, it calls Cloud Manager and Network Manager to request VM creation and flow placement for

11. http://blog.sflow.com/2009/06/sampling-rates.html

12. http://d3js.org

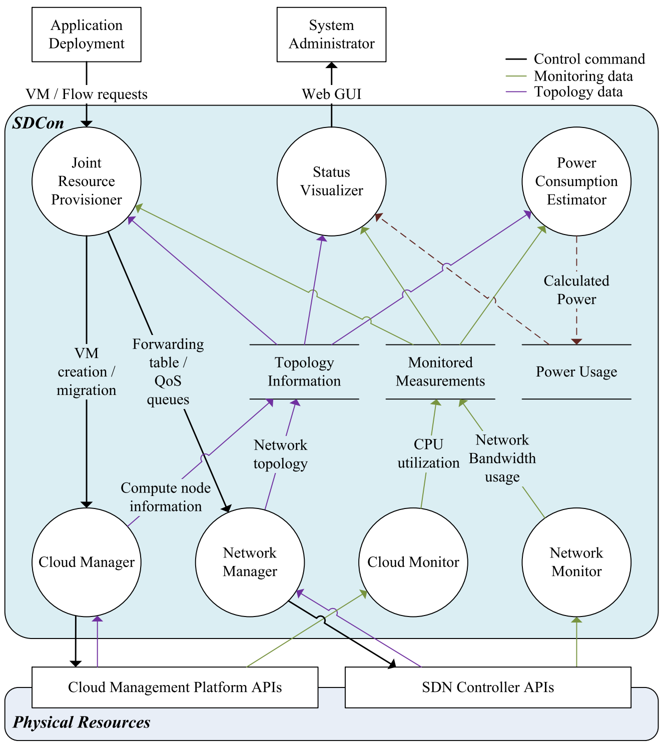


Fig. 3. Data flow diagram of SDCon components.

computing and networking resources respectively. In order to decide where and how to provision the resources, it retrieves the topology information prepared by Topology Discovery, as well as the monitoring data gathered from Cloud Monitor and Network Monitor. In addition, Power Consumption Estimator calculates the power usage based on the monitored measurements and topology information. Measured information is visualized with Status Visualizer through a web interface.

To provide an in-depth understanding of the working process of SDCon, we also depict sequence diagram of VMs and flows deployment with SDcon (see Fig. 4). The diagram shows the process to get VM and host information from OpenStack API and network topology information from OpenDaylight API. For VM deployment, it first runs the VM placement algorithm to select the compute node and create a VM in the selected compute node through CloudManager and OpenStack. For network deployment, we depict the procedure of updating a flow path based on the real-time bandwidth usage of each link as an example. IP address of source and destination VMs are retrieved from Cloud Manager and passed to Network Manager to start the dynamic flow scheduling for the specified pair of IP addresses. Network Manager periodically selects a series of switches along the path with the lowest bandwidth utilization by obtaining monitored data. Then, new forwarding rules with the updated path are pushed into the switches of the selected path.

5 Joint Resource Provisioning in Heterogeneous Clouds

In this section, we propose a joint computing and networking optimization algorithm for heterogeneous resource configuration which can be used with SDCon in empirical environment. Many approaches have been proposed for joint optimization in clouds, but most of them consider a homogeneous configuration where the hardware of physical hosts and switches are identical across the data center [4], [7], [8]. This is an effective assumption in large-scale public cloud considering the

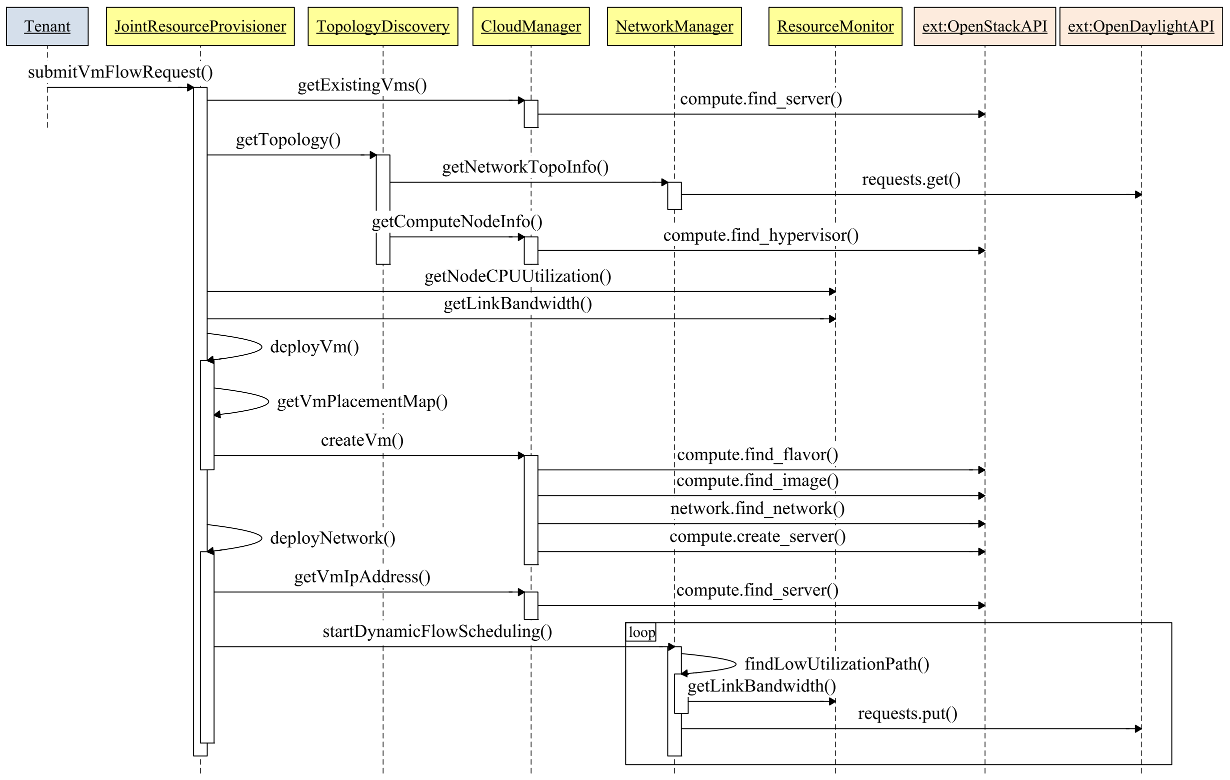


Fig. 4. Sequence diagram to deploy VMs and flows with SDCon.

identical hardware within the same rack which is procured at the same time. However, as the data center grows in need of upgrading or purchasing new hardware, the configuration of new machines can become different from old machines. Also, it is often impractical to purchase many machines at the same time in a small scale private cloud, which can lead to having different hardware configuration even in the same rack.

Resource optimization in heterogeneous configuration needs a different approach from the one for homogeneous resources because of the varied per-unit performance and power consumption. For example, power optimization of a homogeneous cloud by consolidation considers only the number of hosts, whereas in heterogeneous it needs to consider per-unit power consumption level of each compute node. When VMs can be consolidated into the smaller number of physical hosts, unused hosts can be powered off to save energy of the data center. In a homogeneous model, reducing the number of turned-on hosts obviously leads to the reduction of energy consumption of the entire data center, whereas in heterogeneous configuration, we have to consider the different capacity and power consumption level of each physical machine. If VMs are consolidated into a less power-efficient host consuming more energy than two or more hosts, consolidation can actually increase the total power consumption of a data center. Thus, the power consumption level of different host types must be taken into an account for power optimization in heterogeneous clouds.

5.1 Topology-aware VM Placement Algorithm for Heterogeneous Cloud Infrastructure (TOPO-Het)

VM placement in heterogeneous resources can be considered as a variable-sized bin packing problem, which is an NP-hard problem to find the optimal solution [22]. As it is impractical to find an optimal solution in real-time, we present a heuristic algorithm in order to reduce the problem complexity suitable for on-line VM placement. Our algorithm aims at network-aware VM allocation for heterogeneous cloud infrastructure for improved network performance of VMs.

For compute nodes with different computing capacity (e.g., number of cores and size of memory), there is higher possibility that the compute node with larger capacity runs more numbers of VMs than the one with smaller capacity. Assuming that all compute nodes have the same network capacity, those VMs in the larger compute node will utilize less bandwidth when the VMs use the network simultaneously, because the same amount of network bandwidth is shared by more numbers of VMs in the larger node. On the other hand, VMs placed in the same node can communicate to each other via in-memory transfer rather than through network interface, which can provide far more inter-VM bandwidth within a node. Thus, we need to consider the connectivity of VMs for placing VMs into the smaller number of compute nodes. If the connected VMs are placed into the same node, it not only provides more bandwidth for VMs but also reduces the amount of traffic in network infrastructure. However, if unrelated VMs are placed in the same node, it consumes more networking resources and reduces per-VM bandwidth because the limited bandwidth of a network interface should be shared by more number of VMs.

Algorithm 1 is proposed to address the contradiction of VM placement in the same or closer compute nodes. The algorithm considers the connectivity between VMs and finds the nearest compute node if the other VM is already placed, or the group of closely-connected compute nodes which can collectively provide the requested resources for VMs. In Line 5, VMs are grouped based on the network connectivity information gathered from the additional networking requests of SDCon input data in JSON format. If there is a connection request between two VMs, they are put into the same VM group. In Line 7, VMs in the same group are sorted in the order of required resources from high to low to make sure a large VM is placed before smaller VMs. Please note that *sort* function in the Algorithm 1 is a general sort function, such as quick sort, which has the complexity of $O(n \cdot \log n)$. *HG* on Line 8 refers a list of the set of compute nodes (hosts) which collectively provide sufficient resources to the group of VMs. The algorithm overall constructs *HG* with the available host groups in the order of closely connected host groups in the network topology.

Algorithm 1. TOPO-Het: Topology-Aware Collective VM Placement Algorithm in Heterogeneous Cloud Data Center

```
1:  Data: $VM$: List of VMs to be placed.
2:  Data: $F$: List of network flows between VMs.
3:  Data: $H$: List of all hosts in data center.
4:  Data: $topo$: Network topology of data center.
5:  $VMG \leftarrow$ Group $VM$ based on the connection in $F$.
6:  for each VM group $vmg$ in $VMG$ do
7:      sort($vmg$, key=$vm$.size, order=desc)
8:      $HG \leftarrow$ empty list for available host groups;
9:      $VM_{conn} \leftarrow topo$.getConnectedVMs($vmg$);
10:     if $VM_{conn}$ is not empty then
11:         $H_{host} \leftarrow topo$.findHostRunningVMs($VM_{conn}$);
12:         $H_{edge} \leftarrow topo$.getHostGroupSameEdge($H_{host}$);
13:         $H_{pod} \leftarrow topo$.getHostGroupSamePod($H_{host}$);
14:         $HG$.append($H_{host}$, $H_{edge}$, $H_{pod}$);
15:     end if
16:     $HG_{host} \leftarrow topo$.findAvailableHost($vmg$);
17:     sort($HG_{host}$, key=$hg$.capacity, order=desc);
18:     $HG_{edge} \leftarrow topo$.findAvailableHostGroupEdge($vmg$);
19:     sort($HG_{edge}$, key=$hg$.capacity, order=desc);
20:     $HG_{pod} \leftarrow topo$.findAvailableHostGroupPod($vmg$);
21:     sort($HG_{pod}$, key=$hg$.capacity, order=desc);
22:     $HG$.appendAll($HG_{host}$, $HG_{edge}$, $HG_{pod}$);
23:     for each $vm$ in $vmg$ do
24:         for each host group $hg$ in $HG$ do
25:             sort($hg$, key=$h$.freeResource, order=asc);
26:             isPlaced $\leftarrow$ place($vm$, $hg$);
27:             if isPlaced then
28:                 break;
29:             end if
30:         end for
31:         if not isPlaced then
32:           place($vm$, $H$);
33:         end if
34:     end for
35: end for
```

Line 9 finds if any VMs are already deployed in the infrastructure. If there are VMs already placed, it tries to place new VMs to nearby compute nodes close to the placed VMs, e.g., the same node (Line 11), the connected nodes under the same edge switch (Line 12), or the nodes within the same pod (Line 13). In Lines 16-22, the algorithm tries to find the rest of host groups (a host, a host group under the same edge network, or a host group in a same pod) regardless of the placed VMs, if they can provide the requested resources to construct a complete HG that assure the VMs are placed in any hosts regardless of the VM connectivity. These host groups are sorted by the available bandwidth calculated from the number of VMs running on the entire host group with the information of network topology. If there are more VMs running on the host group, it is more likely congested even if the total computing capacity of the group is higher. Note that the heterogeneity of resources is considered at this step when the VMs are assigned to the less congested host group regarding the difference in the capacity of each host group.

Once the algorithm found all the candidate host groups for the VM group, each VM in the group is tried to place in a host within the higher priority host group on Lines 23-34. Note that the host list within a group is sorted by the number of free CPU cores before placing VM Line 25, in order to consolidate VMs into a smaller number of hosts. If the host is not fit for the VM, the next candidate host is tried until the available one is found. If all the candidate host groups are failed, the algorithm places the VM onto any available host in the data center regardless of the network topology.

The time complexity of the proposed algorithm depends on the number of hosts $|H|$, VMs to be placed $|VM|$, and the VM groups $|VMG|$ which can be at most $|VM|$ if each group has only one VM. Sorting the VMs on Line 7 takes at most $O(|VM| \cdot \log|VM|$, and for each VM group sorting the host group takes at most $O(|H|\log|H|)$ in the case of $|HG| = |H|$. As $topo$ uses a tree data structure, all finding functions using $topo$ structure takes $\log|H|$ time. Once the candidate host groups are constructed, placing each VM in one of the candidate hosts takes $O(|H| \cdot \log|H|)$ as it needs to sort the candidate hosts, which needs to be done for all VMs. Therefore, the total time complexity of TOPO-Het is:

$$\begin{aligned} &O(|VM|\log|VM|) + O(|VMG| \cdot |H|\log|H|) \\ &\qquad + O(|VM| \cdot |H|\log|H|) \\ &= O(|VM| \cdot |H|\log|H| + |VM|\log|VM|), \end{aligned}$$

which is feasible time for online VM placement.

5.2 Baseline Algorithms

Our heterogeneity-aware resource allocation algorithm is compared with First-Fit Decreasing (FFD) algorithm in conjunction with Bandwidth Allocation (BWA) and Dynamic Flow Scheduling (DF) network management schemes. FFD is a well-known heuristic algorithm for a bin-packing problem which allocates the largest VM to the most full compute node capable of the VM adopted in many studies [23], [24]. VMs are sorted in descending order of the size of the required resource, and compute nodes are sorted in ascending order of available resources. As it chooses the most full node first, the algorithm consolidates VMs into a smaller number of compute nodes which can benefit to the energy efficiency and network traffic consolidation.

Further, we implement two network management methods, Bandwidth Allocation and Dynamic Flow Scheduling (DF), in combination with FFD to empower FFD baseline algorithm and show the effectiveness of SDCon. In BWA, the requested network bandwidth is allocated for traffic flows between VMs using the aforementioned method in Section 4.3. SDCon renders flow settings provided with VM request and creates QoS queues along switches forwarding the VM traffic. DF (Dynamic Flow Scheduling), on the other hand, updates the network path of the VM-to-VM flow periodically by monitoring real-time traffic, which can be seen in many approaches [25], [26], [27]. It first finds the shortest path between VMs and retrieves monitored traffic of the candidate path. After collecting bandwidth usage statistics of every link on each path, it selects the less congested path and updates forwarding tables on switches along the selected path through SDN controller. In our implementation, DF checks and updates each network path every 60 seconds using the monitored data, which can be adjusted in SDCon for delay sensitive applications.

We combine the network management methods with FFD VM allocation algorithm. In the following sections, we refer BWA and DF as a combination of FFD VM allocation method with the referred network management scheme. The key difference between the baselines and TOPO-Het is that TOPO-Het considers network topology and distance between compute nodes in a data center for allocating VMs and network flows. TOPO-Het also considers the heterogeneous configuration of compute nodes in the decision process. It is also worth to note that both TOPO-Het and FFD utilize the topology information obtained from Topology Discovery module in SDCon, and DF utilizes the real-time monitored bandwidth utilization data from Network Monitor module. BWA installs flow rules in each OpenFlow switch through SDCon's Network Manager.

6 System Validation

In this section, we describe a testbed setup and the experiment for validating the system by testing bandwidth allocation functionality of SDCon. Additional validations, such as VM placement, dynamic flow scheduling, and power usage estimation, are also undertaken in conjunction with the performance evaluation of the topology-aware resource provisioning algorithm and network management schemes in Section 7.

6.1 Testbed Configuration

In order to evaluate the proposed system and algorithm on the empirical environment, we deployed SDCon on our testbed equipped with 8 compute nodes, a controller node, and 10 OpenFlow switches. Fig. 5 shows the architectural configuration of our testbed. The network topology is built as a modified 2-pod fat-tree architecture which has less number of pods from the original proposal [19]. With the cost and physical limitation, we built two pods connected with two core switches instead of four. This modified topology still provides multiple paths between two hosts with the full support of 1:1 over-subscription ratio within the same pod as proposed in the original fat-tree topology.

Core, aggregate, and edge switches are implemented with 10 Raspberry-Pi hardware with external 100 Mbps USB 2.0 Ethernet connectors on which cables are connected to the other Raspberry-Pi or compute nodes. A similar approach was proposed by researchers from Glasgow University [28]

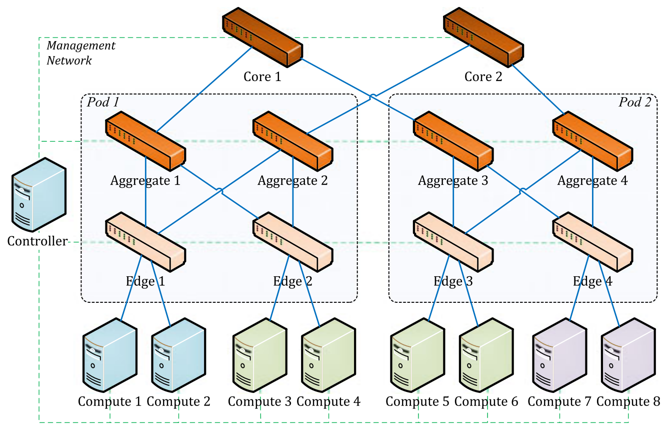


Fig. 5. Testbed configuration.

that Raspberry-Pi was used for compute nodes in clouds. In our testbed, we use Raspberry-Pi to build SDN switches, not compute nodes, as we have enough servers for compute nodes whereas we cannot procure OpenFlow switches.

In each Raspberry-Pi switch, OpenVSwitch is running as a forwarding plane on a Linux-based operating system. A virtual OpenVSwitch bridge is created in each Raspberry Pi to include all Ethernet interfaces as ports and connected to OpenDaylight SDN controller running on the controller node. All links between Raspberry-Pi switches and compute nodes are set to 100 Mbps due to the limited bandwidth of the external USB Ethernet adapters of Raspberry-Pi. Please note that packet processing in OpenVSwitch uses run to completion model without buffering.[13] We configured a separate management network for controlling and API communications between switches, compute nodes, and the controller.

Each compute nodes are running OpenStack Nova-Compute component to provide computing resources to VMs as well as OpenStack Neutron-OVS Agent for virtual network configuration for the hosted VMs. OpenStack components in compute nodes are connected to the OpenStack server running on the controller node through a separate management network. Due to our limited device availability, three different types of computers are used for the controller and compute nodes. Table 1 shows the different hardware configuration of each computer.

In OpenStack configuration, we create a flat network for VM communications instead of using tunneling methods, due to the limitation of flow matching rules in OpenFlow. Tunneled addresses are invisible at OpenFlow switches unless decapsulating packets in each switch which can result in severe overheads at switches. Instead, IP addresses of VMs are assigned with the same range of the compute nodes in the flat network, thus switches can differentiate flows based on the pair of source and destination VM addresses without decapsulation and re-encapsulation at every switch.

6.2 Bandwidth Allocation with QoS Settings

As described in Section 4, we implement network bandwidth allocation by applying OpenVSwitch's QoS and Queue configuration. This experiment is to see the effectiveness of the implemented bandwidth management method in SDCon. We use *iperf* tool to measure the bandwidth of a flow between VMs sharing the same network path. In order to make the network bandwidth to be shared by different flows, we run iperf on two or three different VMs simultaneously. Also, various combinations of TCP and UDP protocols are used in experiments to compare the impact of our bandwidth allocation mechanism on different protocols. Note that the maximum bandwidth of our testbed is measured at approximately 95 Mbps (bits/s), due to the Ethernet port limitation of our Raspberry-Pi switches. We set up QoS configuration with HTB policy and specify the minimum bandwidth of 70 Mbps for QoS flows and 10 Mbps for other flows, and maximum of 95 Mbps for all flows.

Fig. 6 shows the measured bandwidth of different flows sharing the same network path. When iperf was used in TCP mode (Fig. 6a), the bandwidth is equally shared by two flows without QoS configuration. After applying QoS configuration for Flow 1, the bandwidth for Flow 1 is increased to 73.6 Mbps, whereas Flow 2 can acquire only 14.9 Mbps. Fig. 6b shows the measured bandwidth for two UDP flows. In UDP mode, we run iperf at a constant rate of 70 Mbps which causes a heavy congestion when two or more flows are shared the same path. Because of the heavy congestion, the shared bandwidth between two flows, in this case, decreased to about 30 Mbps for each flow without QoS setting. However, similar to the TCP result, Flow 1's bandwidth is increased to 56.1 Mbps after applying QoS, which is slightly less than the minimum bandwidth specified in the QoS setting. This is due to the characteristic of UDP which does not provide any flow and congestion control mechanism. For three UDP flows sharing the same path (Fig. 6c), Flow 1 with QoS configuration acquires 46.8 Mbps while the other two flows acquire less than 8.5 Mbps.

The last case is to measure the bandwidth with a mixed flow of TCP and UDP protocol (see Fig. 6d). When QoS was not configured, the UDP flow could obtain 61.1 Mbps while the TCP flow acquired 28.1 Mbps, because TCP's flow control mechanism adjusts the transmission rate to avoid network congestion and packet drop. However, after applying QoS to the TCP flow (Flow 1), its bandwidth is drastically increased to 70.6 Mbps, although Flow 2 is constantly sending UDP packets at 70 Mbps. This shows that the QoS queue could forward TCP packets at 70 Mbps speed as specified in the settings, whereas UDP packets in a non-QoS queue were mostly dropped resulting in 5.5 Mbps bandwidth measurement at the receiver. A similar result is observed when we applied QoS configuration for the UDP flow. The UDP flow (Flow 2) with QoS setting can obtain 65 Mbps while the bandwidth for Flow 1 is measured at 25.5 Mbps.

The results show that our QoS configuration mechanism for bandwidth allocation is effective for both TCP and UDP flows in the situation of multiple flows simultaneously sharing the same network path, although the acquired bandwidth is varied depending on the network status. Configuring QoS and queue settings on OpenVSwitch in addition to the flow rules added for a specific flow can make the priority flow exploit more bandwidth than non-priority flows in the congested network.

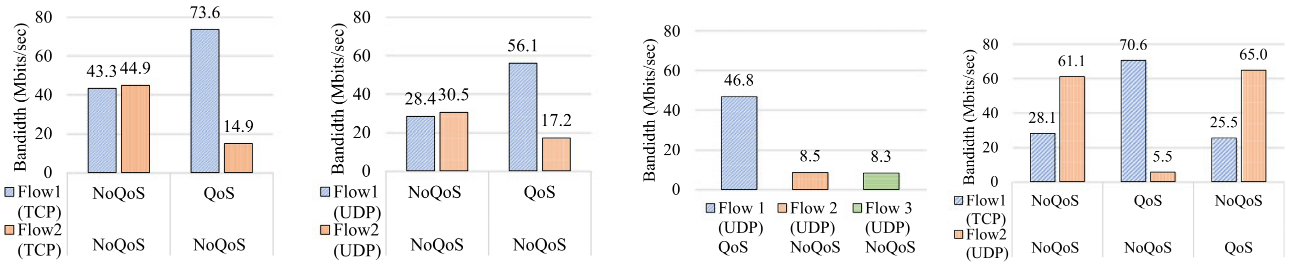
7 Performance Evaluation

We evaluate the proposed system and the algorithm with a real-world application and workload: Wikipedia application.

13. http://docs.openvswitch.org/en/latest/faq/design/

TABLE 1
Hardware Specification of Controller and Compute Nodes

Nodes	CPU model	Cores (VCPUs)	RAM	Quantity
Controller, Compute 1,2	Intel(R) E5-2620 @ 2.00 GHz	12 (24)	64 GB	3
Compute 3-6	Intel(R) X3460 @ 2.80 GHz	4 (8)	16 GB	4
Compute 7,8	Intel(R) i7-2600 @ 3.40 GHz	4 (8)	8 GB	2



(a) Measurerd bandwidth of two TCP flows.

(b) Measurerd bandwidth of two UDP flows.

(c) Measurerd bandwidth of three UDP flows.

(d) Measurerd bandwidth of TCP and UDP flows.

Fig. 6. Bandwidth measurement of multiple flows sharing the same network resource.

Wikipedia publishes its web application, named MediaWiki, and all database dump files online[14] so that we can replicate Wikipedia application on our own environment with various settings. Testing with Wikipedia application becomes more straightforward with the introduction of WikiBench [29], a software to deploy Wikipedia database dumps and measure web requests to MediaWiki servers based on the historical traffic trace. Thanks to the open-sourced MediaWiki application and WikiBench tool, Wikipedia application is widely adopted in cloud computing research for evaluation. Our experiments with Wikipedia are conducted on the same testbed explained in the previous section, comparing our proposed algorithm with baseline algorithms discussed in Section 5.

7.1 Application Settings and Workloads

For empirical evaluations, we deploy two Wikipedia applications consisting of multiple VMs across our testbed using SDCon to measure the response time. A general web application architecture is exploited to configure the Wikipedia with three types of VMs: client (presentation tier), web server (application tier), and database (data tier) VMs. Client VM is acting as a client which sends requests parsed from the trace to web servers, in which provides web response with the local file or the data retrieved from the database server. Client VMs run WikiBench program to read trace files and measure the response time of the request. Web server VMs are configured with Apache2 server to host MediaWiki application written in PHP language. Database VMs run MySQL which retrieves Wikipedia database rebuilt from the dump file.

We set two applications to represent two different tenants using the data center with a different number of VMs to check the impact of the TOPO-Het algorithm on the scale of the application. Application 1 (App 1) is deployed with four client VMs, four web server VMs, and one database VM, whereas Application 2 (App 2) consists of two clients, two web servers, and one database VM. Although they are separate applications without any network traffic in between, we deploy and run both applications at the same time to see the impact of multiple applications run by different tenants sharing the network resources of a data center. Each VM is configured with a predefined VM Type shown in Table 2. Before running the experiment, we created several VMs in some compute nodes (12 cores in Compute node 2, 4 cores in Compute node 4, and 1 core in Compute node 5 and 9) to represent the other tenants occupying resources. We also manually placed the database VM of App 2 onto Compute 5 node in order to evaluate the case when some VMs of the application are pre-deployed. VM images are prepared with proper software settings to boost the speed of application deployment.

In addition to VM configurations, network flow settings are defined for VM traffics (Table 3). We set 30 Mbps for traffics from web server VM to client VM, and 70 Mbps from the database to web server VM. Since the maximum bandwidth available for each physical link in our testbed is 100 Mbps, a web server still can utilize the entire bandwidth when it needs to send data to both database and client if a

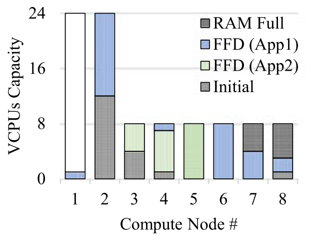
14. http://dumps.wikimedia.org

TABLE 2
VM Types for Wikipedia Application Deployment

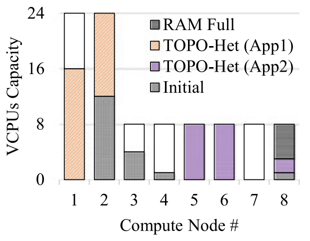
					# of VMs	
VM Type	CPU cores	RAM	Disk	Software	App 1	App 2
Client	1	2 GB	20 GB	WikiBench	4	2
Web server	4	7 GB	80 GB	MediaWiki	4	2
Database	8	15.5 GB	160 GB	MySQL	1	1

TABLE 3
Network Flow Settings for VM Traffic

Source VM	Destination VM	Requested Bandwidth
Database	Web server	70 Mbits/s
Web server	Client	30 Mbits/s

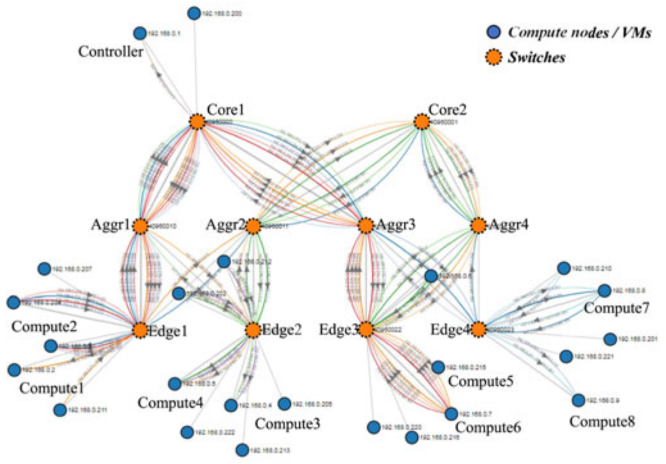


(a) Number of VCPUs after placing VMs with FFD algorithm.

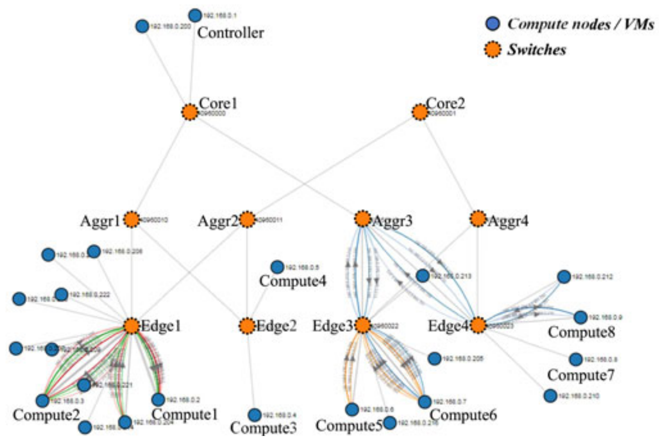


(b) Number of VCPUs after placing VMs with TOPO-Het algorithm.

Fig. 7. Used and available CPU cores of compute nodes after VM placement in different algorithm.



(a) Visualized network traffic of VMs placed with FFD algorithm.



(b) Visualized network traffic of VMs placed with TOPO-Het.

Fig. 8. Screenshots of Status Visualizer showing network traffic between compute nodes and switches after Wikipedia application deployment with different algorithms.

whole compute node is excursively occupied by a web server VM. Please note that all the VMs in the same application are grouped as a same VM group in TOPO-Het algorithm, because they have connections to each other. Thus, there are two VM groups (one group per application) in the scenario.

7.2 Analysis of VM Allocation Decision

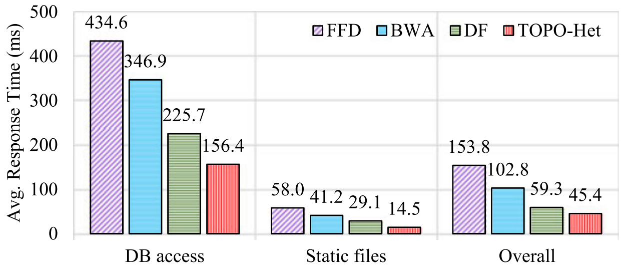
We submit the VM configurations and flow definitions for two applications to SDCon and run the VM allocation algorithms. Fig. 7 shows CPU capacity of compute nodes in the testbed before and after allocating VMs with different algorithms. Before deploying two Wikipedia applications, Compute node 1, 6, and 7 were empty, while the other nodes had running VMs representing other tenants, which is shown as Initial in the figure. FFD algorithm allocates VMs for App 2 to Compute 3 and 4 nodes, and VMs for App 1 across the data center filling up empty slots. Because FFD does not consider network topology, VMs of both App 1 and App 2 are scattered across the data center. VMs for App 1 are placed almost every compute node, whereas VMs for App 2 are consolidated in Compute 3, 4, and 5 which are still scattered on different network pods. Note that Compute 7 and 8 could not host more VMs because of RAM limitation, even though they have free CPUs.

In comparison, TOPO-Het considers network topology, and the VMs for the same application are consolidated within the same pod or edge network (Fig. 7b). For App 2, new VMs are placed in Compute 5 and 8, which are under the same network pod with the already placed VM (in Compute 5). Similarly, all VMs of App 1 are placed either in Compute 1 or 2, which are connected to the same edge switch. Also, Compute 7 remains empty after VM allocation so that the data center can save more power consumption if Compute 7 is turned into idle mode or powered off. In short, the proposed algorithm allocates VMs to nearby compute nodes aware of network topology while still provides VM consolidation to the minimum number of compute nodes.

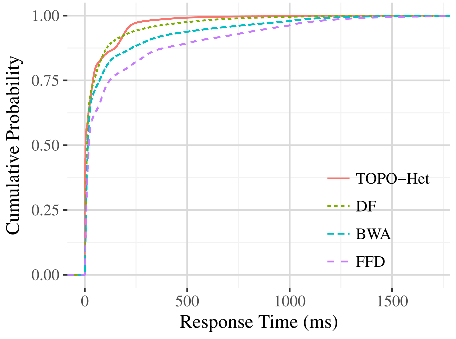
The network traffic visualized by SDCon Status Visualizer shows the effectiveness of TOPO-Het in network resources. Fig. 8 depicts the screenshot of Status Visualizer web UI captured two minutes after starting each experiment. The network topology follows the testbed architecture (Fig. 5), with orange circles with dash line refer to network switches (Core1-2, Aggr1-4, Edge1-4), and blue circles refer to compute nodes (Compute1-8) and VMs. Please note that both VMs and compute nodes are represented as blue dots because we use a flat network for VMs instead of VLAN or other tunneling methods. The arc lines between dots refer to the network traffic between those nodes, separated by the end-to-end VM flows. Thus, more numbers of lines with different colors indicate that the link had more network traffics for different source-destination pairs of VMs. When VMs are placed with FFD that is lack of the information of network topology, network traffics for both applications flood across all the network links in the testbed. In contrast, VMs placed with TOPO-Het are consolidated under the same pod or edge switches which results in significant reduction of network traffics.

7.3 Analysis of Response Time

The average response time of Wikipedia applications measured with WikiBench program from App 1 and App2 are shown in Fig. 9 and 10 respectively. Web requests are sent for 30 minutes duration with 20 percent for App 1 (approx. 140,000 requests per client) and 10 percent (approx. 75,000 requests per client) for App 2 from the original trace from Wikipedia. Each client injects the workload individually from a local file to avoid the network latency and traffic between client VMs. For clear comparison, we further separate the average response time based on the type of request. DB access is the request that needs a database access to retrieve the



(a) Average response time per request of App 1.



(b) CDF graph of response time of App 1.

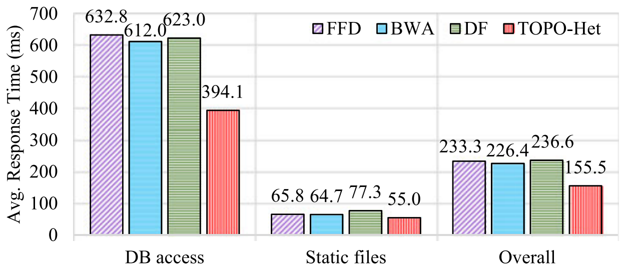
Fig. 9. Average and CDF of response time measured with Wikipedia workload from App 1.

requested contents, whereas static file is to transfer files from the web server without access to the database.

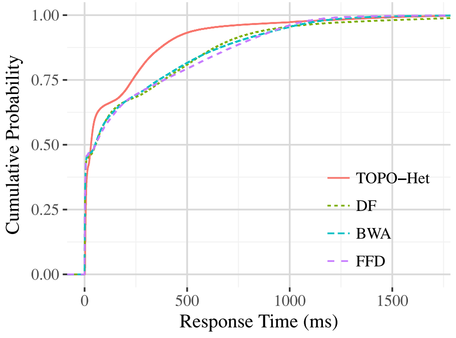
Fig. 9 shows the average and CDF of response times measured from App 1. The overall response time is measured at 45.4 ms with the TOPO-Het algorithm which is about one-third of the response time from FFD algorithm (153.8 ms). Because TOPO-Het places the connected VMs in a short distance to each other, the network latency is significantly lowered transferring within an edge or pod network. On the other hand, FFD distributes VMs across the data center which leads to increasing the response time. The result also shows that applying network management mechanism in addition to the FFD allocation can improve the performance. The average response time is reduced to 102.8 ms by applying BWA method, and further to 59.3 ms with DF algorithm. Even for the VMs spread out across the data center by FFD algorithm, network traffic engineering schemes supported by SDCon can improve the performance of the application. We can see the response times for DB access requests are longer than the static files because they obviously need to access database server involving more network transmission between the web server and database VMs and file operation in the database. Nonetheless, the response time for DB access and static file requests follow the same tendency as the overall result: TOPO-Het resulting in the best performance whereas FFD without a network management scheme being the worst. CDF graph of response time from all requests (Fig. 9b) also shows that TOPO-Het outperforms compared to baselines.

For App 2 which has been running simultaneously with App 1, a similar result can be observed in Fig. 10. Compared to App 1, the overall average response time is increased to between 155.5 ms and 236.6 ms mainly because of the poor performance of database VM. In our heterogeneous testbed, the disk I/O performance of Compute 5 node hosting the database VM for App 2 is lower than Compute 1 and 2 nodes, because of the hardware specification. Due to the poor performance, longer response time is observed especially for DB access requests in App 2. Another reason is that VMs for App 2 are more scattered across the data center into smaller compute nodes as observed in Fig. 7 leading to increasing the network latency.

Nonetheless, we can still see that TOPO-Het outperforms other baselines although the average response time is increased compared to App 1. However, network management schemes are less effective for App 2 not improving the response time significantly. In fact, DF method degrades the overall performance slightly compared to the original FFD, increasing the response time from 233.3 ms to 236.6 ms. The reason is that the smaller scale with less number of VMs and workloads did not generate as much network traffic as App 1, which results in the network bandwidth not a significant bottleneck for the overall performance. Rather than bandwidth which could be improved by SDCon's network management schemes, the poor performance of App 2 is mainly due to the network latency caused by switches. As shown in Fig. 10a, the time difference for static file requests are not as high as the difference for DB access requests between FFD and TOPO-Het. This implies the network traffic between the database and web server is improved by TOPO-Het which allocates web server VMs into the compute node under the same edge as the database VM.



(a) Average response time per request of App 2.



(b) CDF graph of response time of App 2.

Fig. 10. Average and CDF of response time measured with Wikipedia workload from App 2.

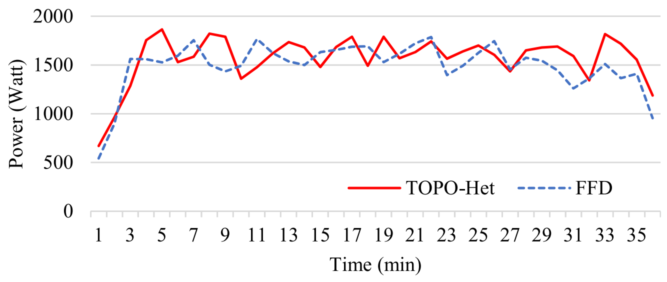


Fig. 11. Overall power usage of the testbed estimated from CPU and network utilization.

In summary, results show that SDCon can perform VM placement with the intended algorithm, either FFD or TOPO-Het, as well as configure network settings dynamically through SDN controller based on the selected traffic engineering scheme. TOPO-Het can improve the application performance for both application scenarios by allocating VMs to nearby compute nodes aware of network topology, although the level of improvement is different based on the application configuration, the workload and the current status of the infrastructure.

7.4 Analysis of Power Consumption

We also measured the power consumption of the testbed using SDCon. Fig. 11 depicts the aggregated power usage of all infrastructure per time unit measured at every minute. Note that the measurement is estimated by calculating from energy models with the CPU utilization level for compute nodes and the number of online ports for switches as explained in Section 4.7. We set P_{idle} at 147W, 111W, 102W, and P_{idle} at 160W, 80W, 80W, for Compute 1-2, 3-6, and 7-8 respectively. For switches, we set P_{static} at 66.7W and P_{port} at 1W. Although TOPO-Het could leave one compute node empty, the estimated power consumption is not significantly different between two algorithms. Because we did not implement the method to power off idle compute nodes and the aggregated CPU utilization level remains similar in both algorithms, the power consumption was not significantly differentiated between algorithms. In summary, the power consumption of data center with TOPO-Het remains at least as the same level as the baseline.

Nevertheless, the result of power consumption measurement shows the potential application of SDCon for energy-related research such as energy-efficient joint resource provisioning. With the full integrity controlling both computing and networking resources, measuring power usage, and real-time resource monitoring, empirical studies on SDC are feasible with SDCon platform.

8 Conclusions and Future Work

The introduction of SDN brings up many opportunities and challenges when it is applied in cloud computing environment. The increasing popularity of applying SDN in cloud data centers demands a management platform which can jointly control both networking and cloud resources. In this paper, we introduced SDCon, an integrated control platform for SDN and clouds to enable orchestrated resource management for both resources. SDCon is capable of placing VM requests and configuring network forwarding rules and QoS queues with the knowledge of networking and computing information of the data center. It also implements real-time monitoring for computing and networking resources which can be further exploited for dynamic resource provisioning. The power monitoring module can calculate the estimated power consumption based on the resource utilization using a power model, and the visualizer displays an overall status of the data center.

We also proposed a VM placement algorithm for jointly considering both computing and networking resources in heterogeneous infrastructure. The algorithm consolidates VMs for the same application with network connectivity into closely connected compute nodes in order to decrease the network traffic as well as to improve the application performance. The algorithm is evaluated on our testbed controlled by SDCon platform and compared with baselines. The results show that the proposed algorithm outperforms the compared state-of-the-art baselines. The effectiveness of SDCon's controllability is also shown through the validation and evaluation experiments, including bandwidth allocation with QoS configuration and dynamic flow scheduling.

The current version of SDCon has a flaw on the scalability for large-scale infrastructure such as those involving multiple data centers. When we designed and implemented SDCon, it aims at working on a small-scale testbed configured within the lab, due to the limited infrastructure resources. Scalability can be achieved by creating a cluster with multiple instances of SDCon with the introduction of East-West-bound APIs adopted from SDN controller design [30]. In addition to SDCon, the underlying cloud and SDN controllers (OpenStack and OpenDaylight) have to be able to scale out in accordance with the size of cloud infrastructure, e.g., the monitoring modules need more careful attention in scalability to be able to collect and aggregate the real-time information in a timely manner.

For the same reason, the evaluation of TOPO-Het algorithm was performed only within the small-scale testbed in our laboratory with a Wikipedia application. More extensive evaluations can be performed to show the effectiveness of TOPO-Het in a large-scale infrastructure with various application models. To support large-scale evaluations, we have developed a simulator called CloudSimSDN [9] and evaluated several joint host-network provisioning algorithms [4], [31] in the simulation environment setting. In these earlier research work, we conducted simulation-based experiments in large-scale and demonstrated the scalability of similar joint VM and network resource provisioning approaches with hundreds of compute nodes and network switches. In this paper, we followed a similar approach for designing the integrated platform and the joint provisioning algorithm, so that our approaches are essentially scalable as shown in the previous work.

In the future, SDCon can be extended to support alternative cloud management platforms and SDN controller softwares other than OpenStack and OpenDaylight. The power monitoring module can be improved for accurate measurements by installing a physical monitoring equipment for compute nodes and switches instead of model-based theoretical calculation. Also, the platform can be improved to support a variety of network topologies whereas the initial version was developed in consideration of three-tier Fat-tree topology only. More functional features, such as managing virtual network functionalities, network middleboxes, and security modules, can be added to the platform for autonomic software-defined clouds. Such features will allow SDCon to be employed for developing innovative approaches dealing with energy-aware resource provisioning, NFV scheduling, and security threat protection. TOPO-Het algorithm can be improved by importing various migration policies for VMs and flows to adapt to the dynamic update of the infrastructure. More evaluations with complex application models including Map-Reduce and batch processing can be further performed with a large-scale testbed.

Acknowledgments

This work is partially supported by ARC Discovery Project. We thank Adel Nadjaran Toosi, Minxian Xu, Editor-in-Chief (Manish Parashar), Associate Editor, and reviewers for their valuable comments and suggestions that helped in improving the paper significantly.

References

[1] R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon, “Software-defined cloud computing: Architectural elements and open challenges,” in *Proc. 3rd Int. Conf. Adv. Comput. Commun. Informat.*, 2014, pp. 56–74.

[2] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, “Software defined cloud: Survey, system and evaluation,” *Future Generation Comput. Syst.*, vol. 58, pp. 56–74, 2016.

[3] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, “Joint host-network optimization for energy-efficient data center networking,” in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, May 2013, pp. 623–634.

[4] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, “SLA-aware and energy-efficient dynamic overbooking in SDN-based cloud data centers,” *IEEE Trans. Sustainable Comput.*, vol. 2, no. 2, pp. 76–89, Apr. 2017.

[5] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, “Joint vm placement and routing for data center traffic engineering,” in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2876–2880.

[6] K. Zheng, X. Wang, L. Li, and X. Wang, “Joint power optimization of data center network and servers with correlation analysis,” in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 2598–2606.

[7] R. Cziva, S. Jout, D. Stapleton, F. P. Tso, and D. P. Pezaros, “SDN-based virtual machine management for cloud data centers,” *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 2, pp. 212–225, Jun. 2016.

[8] K. Zheng, W. Zheng, L. Li, and X. Wang, “PowerNetS: Coordinating data center network with servers and cooling for power optimization,” *IEEE Trans. Netw. Serv. Manage.*, vol. 14, no. 3, pp. 661–675, Sep. 2017.

[9] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, “CloudSimSDN: Modeling and simulation of software-defined cloud data centers,” in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, May 2015, pp. 475–484.

[10] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proc. 9th ACM SIGCOMM Workshop Hot Top. Netw.*, 2010, Art. no. 19.

[11] C. H. Benet, R. Nasim, K. A. Noghani, and A. Kassler, “OpenStackEmu - a cloud testbed combining network emulation with OpenStack and SDN,” in *Proc. 14th IEEE Annu. Consum. Commun. Netw. Conf.*, Jan. 2017, pp. 566–568.

[12] B. Martini, D. Adami, A. Sgambelluri, M. Gharbaoui, L. Donatini, S. Giordano, and P. Castoldi, “An SDN orchestrator for resources chaining in cloud data centers,” in *Proc. Eur. Conf. Netw. Commun.*, Jun. 2014, pp. 1–5.

[13] B. Martini, D. Adami, M. Gharbaoui, P. Castoldi, L. Donatini, and S. Giordano, “Design and evaluation of SDN-based orchestration system for cloud data centers,” in *Proc. IEEE Int. Conf. Commun.*, May 2016, pp. 1–6.

[14] M. Gharbaoui, B. Martini, D. Adami, S. Giordano, and P. Castoldi, “Cloud and network orchestration in SDN data centers: Design principles and performance evaluation,” *Comput. Netw.*, vol. 108, no. Supplement C, pp. 279–295, 2016.

[15] D. Adami, B. Martini, A. Sgambelluri, L. Donatini, M. Gharbaoui, P. Castoldi, and S. Giordano, “An SDN orchestrator for cloud data center: System design and experimental evaluation,” *Trans. Emerging Telecommun. Technol.*, vol. 28, no. 11, 2017, Art. no. e3172.

[16] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, “Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers,” *Future Generation Comput. Syst.*, vol. 28, no. 2, pp. 358–367, 2012.

[17] A. N. Toosi, C. Qu, M. D. de Assuno, and R. Buyya, “Renewable-aware geographical load balancing of web applications for sustainable data centers,” *J. Netw. Comput. Appl.*, vol. 83, pp. 155–168, 2017.

[18] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” *Concurrency Comput.: Practice Exp.*, vol. 24, no. 13, pp. 1397–1420, 2012.

[19] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.

[20] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, “Understanding and abstracting total data center power,” in *Workshop Energy-Efficient Des.*, 2009.

[21] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, “CARPO: Correlation-aware power optimization in data center networks,” in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1125–1133.

[22] J. Kang and S. Park, “Algorithms for the variable sized bin packing problem,” *Eur. J. Oper. Res.*, vol. 147, no. 2, pp. 365–372, 2003.

[23] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing,” *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, May 2012.

[24] R. Wang, R. Esteves, L. Shi, J. A. Wickboldt, B. Jennings, and L. Z. Granville, “Network-aware placement of virtual machine ensembles using effective bandwidth estimation,” in *Proc. 10th Int. Conf. Netw. Serv. Manage. Workshop*, Nov. 2014, pp. 100–108.

[25] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, “OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks,” in *Proc. Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, Dec. 2012, pp. 1–8.

[26] R. Wang, S. Mangiante, A. Davy, L. Shi, and B. Jennings, “QoS-aware multipathing in datacenters using effective bandwidth estimation and SDN,” in *Proc. 12th Int. Conf. Netw. Serv. Manage.*, Oct. 2016, pp. 342–347.

[27] K. Zheng, X. Wang, and J. Liu, “DISCO: Distributed traffic flow consolidation for power efficient data center network,” in *Proc. 16th Int. IFIP TC6 Netw. Conf. Netw.*, 2017, pp. 1–9.

[28] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, “The Glasgow Raspberry Pi cloud: A scale model for cloud computing infrastructures,” in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst. Workshops*, Jul. 2013, pp. 108–112.

[29] G. Urdaneta, G. Pierre, and M. van Steen, “Wikipedia workload analysis for decentralized hosting,” *Elsevier Comput. Netw.*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

[30] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, “On scalability of software-defined networking,” *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[31] J. Son and R. Buyya, “Priority-aware VM allocation and network bandwidth provisioning in software-defined networking (SDN)-enabled clouds,” *IEEE Trans. Sustainable Comput.*, 2018. [Online]. Available: http://doi.org/10.1109/TSUSC.2018.2842074

Jungmin Son is a Research Associate within CLOUDS Laboratory, University of Melbourne, Australia.

Rajkumar Buyya is a Redmond Barry distinguished professor and the director with CLOUDS Laboratory, University of Melbourne, Australia.