# Integrated Provisioning of Compute and Network Resources in Software-Defined Cloud Data Centers

Jungmin Son

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

January 2018

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

# Integrated Provisioning of Compute and Network Resources in Software-Defined Cloud Data Centers

Jungmin Son
*Principal Supervisor: Professor Rajkumar Buyya*

---

**Abstract**

Software-Defined Networking (SDN) has opened up new opportunities in networking technology with its decoupled concept of the control plane from the packet forwarding hardware, which enabled the network to be programmable and configurable dynamically through the centralized controller. Cloud computing has been empowered with the adoption of SDN for infrastructure management in a data center where dynamic controllability is indispensable in order to provide elastic services. The integrated provisioning of compute and network resources enabled by SDN is essential in clouds to enforce reasonable Service Level Agreements (SLAs) stating the Quality of Service (QoS) while saving energy consumption and resource wastage.

This thesis presents the joint compute and network resource provisioning in SDN-enabled cloud data center for QoS fulfillment and energy efficiency. It focuses on the techniques for allocating virtual machines and networks on physical hosts and switches considering SLA, QoS, and energy efficiency aspects. The thesis advances the state-of-the-art with the following key contributions:

1. A taxonomy and survey of the current research on SDN-enabled cloud computing, including the state-of-the-art joint resource provisioning methods and system architectures.
2. A modeling and simulation environment for SDN-enabled cloud data centers abstracting functionalities and behaviors of virtual and physical resources.
3. A novel dynamic overbooking algorithm for energy efficiency and SLA enforcement with the migration of virtual machines and network flows.
4. A QoS-aware computing and networking resource allocation algorithm based on the application priority to fulfill different QoS requirements.
5. A prototype system of the integrated control platform for joint management of cloud and network resources simultaneously based on OpenStack and OpenDaylight.

# Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Jungmin Son, January 2018

# Preface

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of this thesis are discussed in Chapters 2 - 6 which are based on the following publications:

- **Jungmin Son** and Rajkumar Buyya, "A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing," *ACM Computing Surveys*, vol.51, no.3, article 59, 2018.
- **Jungmin Son**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, Xiaohui Ji, Young Yoon, and Rajkumar Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015)*, Shenzhen, China, May 4-7, 2015.
- **Jungmin Son**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "SLA-aware and Energy-Efficient Dynamic Overbooking in SDN-based Cloud Data Centers," *IEEE Transactions on Sustainable Computing (T-SUSC)*, vol.2, no.2, pp.76-89, April-June 1 2017.
- **Jungmin Son** and Rajkumar Buyya, "Priority-aware VM Allocation and Network Bandwidth Provisioning in SDN-Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, 2018 (accepted, in press).
- **Jungmin Son** and Rajkumar Buyya, "SDCon: Integrated Control Platform for Software-Defined Clouds," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (accepted, in press).

# Acknowledgements

PhD is a tough, long, but rewarding journey that a person can experience once in a lifetime. I am truly happy that I have overcome all the adversities and finally approached near to the end of this journey. It would not have happened without endless help from people around me. First and foremost, I would like to thank my supervisor, Professor Rajkumar Buyya, who has offered me the opportunity to undertake a PhD, and provided with insightful guidance, continuous support, and invaluable advice throughout my PhD journey.

I would like to appreciate the members of my PhD advisory committee, Prof. Rui Zhang, Prof. Ramamohanarao Kotagiri, Prof. Umesh Bellur, and Prof. Young Yoon, for their constructive comments on my work. My deepest gratitude goes to Dr. Rodrigo Neves Calheiros, Dr. Amir Vahid Dastjerdi, and Dr. Adel Nadjaran Toosi for the endless assistance on developing my research skills, the valuable comments on my papers, and the collaboration on the research projects in the beginning of my PhD journey.

I would also like to thank all the past and current members of the CLOUDS Laboratory, at the University of Melbourne. In particular, I thank Dr. Sukhpal Singh Gill, Dr. Marcos Assunção, Dr. Maria Rodriguez, Dr. Chenhao Qu, Dr. Deepak Poola, Dr. Atefeh Khosravi, Dr. Nikolay Grozev, Dr. Sareh Fotuhi, Dr. Yaser Mansouri, Safiollah Heidari, Xunyun Liu, Caesar Wu, Minxian Xu, Sara Kardani Moghaddam, Muhammad Hilman, Redowan Mahmud, Muhammed Tawfiqul Islam, TianZhang He, Artur Pilimon, Arash Shaghaghi, Diana Barreto, and Bowen Zhou, for their friendship and support during my PhD.

I acknowledge Australian Federal Government, the University of Melbourne, and Australian Research Council (ARC) for granting scholarships to pursue my PhD study.

I also express my sincerest appreciation to Fr. William Uren, Sean Burke, Dr. Guglielmo Gottoli, and the community of Newman College for giving me an opportunity to stay in such supportive, intellectual, and spiritual environment.

I would like to give heartfelt thanks to my friends in Australia and back in Korea: Taewoong Moon, Donghwan Lee, Sori Kang, Sunghwan Yoon, Miji Choi, Johnny Jiang, Andrew Wang, Charis Kho, Herianto Lim, Younghoon Kim, Namhun Song, Junyoub An, and Jack Fang to name a few, who made my PhD life filled with joy and happiness.

Finally, I am heartily thankful to my mother, brothers, sisters in law, and nephew and nieces for their support and encouragement at all times.

*Jungmin Son*
*Melbourne, Australia*
*January 2018*

# Contents


**1 Introduction** **1**
- 1.1 Background . . . 4
  - 1.1.1 Cloud Computing . . . 4
  - 1.1.2 Data Center Network (DCN) . . . 5
  - 1.1.3 Software-Defined Networking (SDN) . . . 6
  - 1.1.4 SDN-enabled Cloud Computing . . . 7
- 1.2 Research Problems and Objectives . . . 8
- 1.3 Evaluation Methodology . . . 9
- 1.4 Thesis Contributions . . . 10
- 1.5 Thesis Organization . . . 12

**2 Taxonomy and Literature Review** **15**
- 2.1 Introduction . . . 15
- 2.2 Terms and Definitions . . . 16
- 2.3 SDN-clouds Architectures . . . 18
- 2.4 Taxonomy of SDN usage in Cloud Computing . . . 20
  - 2.4.1 Objective . . . 20
  - 2.4.2 Method Scope . . . 23
  - 2.4.3 Target Architecture . . . 23
  - 2.4.4 Application Model . . . 24
  - 2.4.5 Resource Configuration . . . 25
  - 2.4.6 Evaluation Method . . . 26
- 2.5 Current Research on SDN usage in Cloud computing . . . 27
  - 2.5.1 Energy Efficiency . . . 28
  - 2.5.2 Performance . . . 33
  - 2.5.3 Virtualization . . . 41
  - 2.5.4 Security . . . 44
  - 2.5.5 Summary and Comparison . . . 45
- 2.6 Evaluation Methods and Technologies . . . 47
  - 2.6.1 Simulation Platforms and Emulators . . . 47
  - 2.6.2 Empirical Platforms . . . 49
- 2.7 Summary . . . 49

**3 Modeling and Simulation Environment for Software-Defined Clouds** **51**
- 3.1 Introduction . . . 51
- 3.2 Related Work . . . 52

3.3 Software-Defined Cloud Data Center Simulation: Goals and Requirements 55
3.4 Framework Design . . . 56
3.4.1 CloudSim core logic . . . 56
3.4.2 Abstracting physical and virtual topology . . . 57
3.4.3 Network modules . . . 58
3.4.4 Calculating packet transmission time . . . 59
3.4.5 Abstracting user requests . . . 60
3.5 Validation . . . 61
3.5.1 Mininet setup . . . 61
3.5.2 Testbed configuration . . . 62
3.5.3 Validation results . . . 64
3.6 Use Case Evaluation . . . 64
3.6.1 Joint host-network energy efficient resource allocation . . . 64
3.6.2 Traffic prioritization . . . 67
3.7 Summary . . . 70

**4 SLA-aware and Energy-Efficient Dynamic Resource Overbooking 71**
4.1 Introduction . . . 71
4.2 Background . . . 73
4.3 Related Work . . . 74
4.4 Problem Formulation . . . 76
4.4.1 Power Models . . . 76
4.4.2 Problem Formulation . . . 78
4.5 Resource Allocation Framework . . . 79
4.6 Resource Overbooking Algorithm . . . 81
4.6.1 Connectivity-aware Initial VM Placement Algorithms . . . 82
4.6.2 VM Migration Algorithms with Dynamic Overbooking . . . 85
4.6.3 Baseline Algorithms . . . 90
4.7 Performance Evaluation . . . 91
4.7.1 Testbed configuration . . . 92
4.7.2 Workload . . . 93
4.7.3 Initial placement . . . 94
4.7.4 Migration policy . . . 99
4.7.5 Analysis of energy consumption . . . 103
4.7.6 Dynamic overbooking ratio . . . 105
4.8 Summary . . . 106

**5 Priority-aware Joint VM and Network Resource Provisioning 107**
5.1 Introduction . . . 107
5.2 Related Work . . . 109
5.3 System Architecture . . . 112
5.4 Priority-aware Resource Provisioning Methods . . . 114
5.4.1 Priority-Aware VM Allocation (PAVA) . . . 114
5.4.2 Bandwidth Allocation for Priority Applications (BWA) . . . 115
5.4.3 Baseline algorithms . . . 117
5.5 Performance Evaluation . . . 119
5.5.1 Experiment configuration and scenarios . . . 120

5.5.2 Analysis of Response Time . . . 122
5.5.3 Analysis of QoS violation rate . . . 125
5.5.4 Analysis of energy consumption . . . 127
5.5.5 Analysis of algorithm complexity . . . 128
5.6 Summary . . . 129

**6 Prototype System of Integrated Control Platform 131**
6.1 Introduction . . . 131
6.2 Related Work . . . 133
6.3 SDCon: Software-Defined Clouds Controller . . . 135
6.4 System Implementation . . . 137
6.4.1 Cloud Manager . . . 139
6.4.2 Cloud Monitor . . . 139
6.4.3 Network Manager . . . 140
6.4.4 Network Monitor . . . 141
6.4.5 Topology Discovery . . . 141
6.4.6 Joint Resource Provisioner . . . 141
6.4.7 Power Consumption Estimator . . . 142
6.4.8 Status Visualizer . . . 143
6.4.9 Data Flow and Sequence Diagram . . . 144
6.5 Joint Resource Provisioning in Heterogeneous Clouds . . . 144
6.5.1 Topology-aware VM Placement Algorithm for Heterogeneous Cloud Infrastructure (TOPO-Het) . . . 147
6.5.2 Baseline algorithms . . . 150
6.6 System Validation . . . 150
6.6.1 Testbed Configuration . . . 151
6.6.2 Bandwidth Allocation with QoS Settings . . . 152
6.7 Performance Evaluation . . . 154
6.7.1 Application Settings and Workloads . . . 155
6.7.2 Analysis of VM Allocation Decision . . . 156
6.7.3 Analysis of Response Time . . . 157
6.7.4 Analysis of Power Consumption . . . 161
6.8 Summary . . . 161

**7 Conclusions and Future Directions 163**
7.1 Conclusions and Discussion . . . 163
7.2 Future Directions . . . 166
7.2.1 Energy-efficient Cloud Computing . . . 166
7.2.2 Supporting Big Data Applications . . . 167
7.2.3 Scalability of SDN . . . 168
7.2.4 Network Virtualization and NFV . . . 168
7.2.5 Enhancing Security and Reliability in Clouds and SDN . . . 169
7.2.6 Realization of Software-Defined Clouds and Edge Computing . . . 170
7.3 Software Availability . . . 171

# List of Figures


1.1 A conceptual architecture and resource provisioning process in a cloud data center. . . . 2
1.2 Data center network topologies. . . . 5
1.3 Comparison of SDN and traditional networking. . . . 6
1.4 The thesis organization . . . . 12

2.1 SDN-enabled cloud computing architecture. . . . 18
2.2 Taxonomy of SDN usage in cloud computing. . . . 21
2.3 Sub-categories used for our literature review. . . . 27

3.1 CloudSimSDN architecture. . . . 54
3.2 CloudSimSDN Class Diagram. . . . 57
3.3 User request modeling. . . . 60
3.4 Physical topology configuration for the validation test. . . . 63
3.5 Comparison of CloudSimSDN with Mininet for average transmission time for each scenario. . . . 65
3.6 Physical topology for traffic prioritization use case evaluation. . . . 67
3.7 Effect of traffic prioritization. . . . 69

4.1 Resource Allocation Architecture. . . . 79
4.2 Example of consolidation with overbooking . . . . 81
4.3 Network topology used in simulation . . . . 92
4.4 Wikipedia workload for different projects collected for 1st of Sep 2014. . . . 94
4.5 CPU utilization of VMs with Wikipedia workload . . . . 95
4.6 Energy consumption and SLA violation results of initial placement without consideration of connectivity. . . . 96
4.7 Energy consumption and SLA violation results of different initial placement algorithms. . . . 98
4.8 Energy consumption and SLA violation results of different migration strategies implemented on ConnCons (connected VMs in the same host) initial placement algorithm. . . . 100
4.9 Energy consumption and SLA violation results of dynamic and static overbooking algorithms . . . . 101
4.10 Energy saving origination . . . . 103
4.11 Energy consumption observation over time . . . . 104
4.12 CPU utilization and Resource Allocation Ratio of a sample VM. . . . 106

5.1 Architectural design of priority-aware VM and flow management system. 112

5.2 8-pod fat-tree topology setup for experiments. . . . . 121
5.3 Performance matrices of the critical application in Scenario 1 (synthetic workload). . . . . 123
5.4 Performance matrices of the critical application in Scenario 2 (Wikipedia workload). . . . . 123
5.5 Average response time of normal (lower-priority) applications. . . . . 125
5.6 QoS violation rate of critical application workloads. . . . . 126
5.7 Detailed power consumption of hosts and switches in a data center. . . . . 127

6.1 Design principle and control flows. . . . . 135
6.2 System architecture of SDCon and underlying software components. . . . 137
6.3 Data flow diagram of SDCon components. . . . . 143
6.4 Sequence diagram to deploy VMs and flows with SDCon. . . . . 145
6.5 Testbed configuration. . . . . 151
6.6 Bandwidth measurement of multiple flows sharing the same network resource. . . . . 153
6.7 Used and available CPU cores of compute nodes after VM placement in different algorithm. . . . . 156
6.8 Network traffic after Wikipedia application deployment visualized with Status Visualizer. . . . . 158
6.9 Average and CDF of response time measured with Wikipedia workload from App 1. . . . . 159
6.10 Average and CDF of response time measured with Wikipedia workload from App 2. . . . . 160
6.11 Overall power usage of the test bed estimated from CPU and network utilization. . . . . 162

7.1 Future directions. . . . . 166

# List of Tables


2.1 Summary of current research on SDN usage for energy efficiency in cloud computing. . . . 29
2.2 Summary of current research for performance improvement in cloud computing with SDN. . . . 35
2.3 Summary of current research for virtualization in cloud computing with the usage of SDN. . . . 41
2.4 Summary of current research for security in cloud computing with the usage of SDN. . . . 44
2.5 Characteristics of SDN usage in cloud computing. . . . 46

3.1 Link configuration for the validation experiments. . . . 62
3.2 Various scenarios for validation. . . . 63
3.3 VM configurations used for joint host-network energy efficient resource allocation use case. . . . 65
3.4 Energy consumption and the maximum number of simultaneously utilized hosts for different VM placement policies. . . . 66
3.5 VM configurations for traffic prioritization use case. . . . 67
3.6 Characteristics of requests used for traffic prioritization evaluation. Requests are based on the model proposed by Ersoz et al. [33]. . . . 68

5.1 Summary of related works. . . . 109

6.1 Hardware specification of controller and compute nodes. . . . 152
6.2 VM types for Wikipedia application deployment. . . . 156
6.3 Network flow settings for VM traffic. . . . 157

# Chapter 1
# Introduction

THE emergence of cloud computing has led to evolutionary changes in the way of provisioning computing resources on demand by offering vitualized resources on a pay-as-you-go basis. Previously, application providers built their own data center with a vast amount of investment to run and deliver their application services. This needed a huge up-front cost for hardware procurement and infrastructure installation which were not scalable or flexible to increasing demand of the application. In the era of cloud computing, with the expansion of utility computing model [17], application providers only need to select a cloud service which fits their applications, or even automated brokering systems can dynamically select a suitable resource for applications on behalf of the application provider. They can lease computing resources from cloud providers by clicking a few mouse buttons within minutes and deploy their applications without any upfront payment. Cloud computing has brought a scalable and elastic computing along with the subscription-oriented service model.

In networking, the introduction of Software-Defined Networking (SDN) has brought many opportunities to both networking and computing community. Decoupling of the network control logic from data forwarding plane is a key innovation in SDN which contributes to the introduction of network programmability. With the global view of the entire network, the centralized controller can manage the entire 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) became more feasible with the introduction of SDN, where the virtualized network function can move around dynamically around the network with SDN's dynamic network reconfiguration

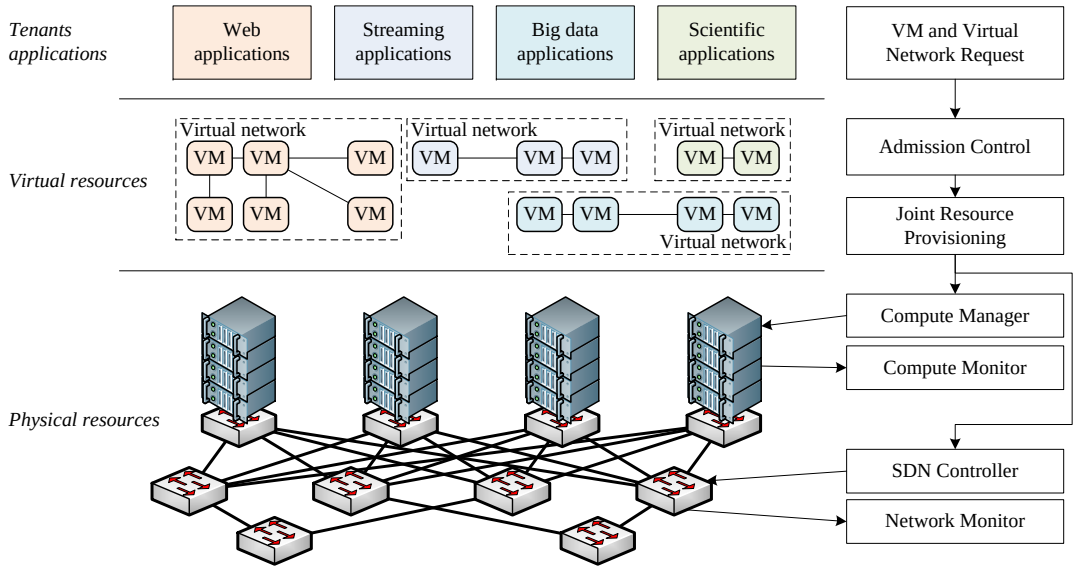



Figure 1.1: A conceptual architecture and resource provisioning process in a cloud data center.

ability.

To provide elastic service to their customers (i.e., tenants), cloud providers exploit virtualization technologies for computing, storage, and networking resources in a data center. The virtualized resources are shared and leased to cloud tenants, which are provisioned from the physical resources in the data center. As the scale of a cloud grows, it often runs tens of thousands of compute servers connected via thousands of switches. With its high complexity of resource management, a cloud provider requires more efficient and sophisticated resource provisioning schemes for both compute and network resources. However, in traditional networks, each network switch has its own control logic which individually decides its behavior based on the information obtained from its neighbors. The traditional network approach raises network manageability and performance issues in a cloud data center due to its massive scale on which computing and network resources should be dynamically provisioned to adapt to the fluctuating requests from various tenants. In this regard, SDN triggered new innovations in a cloud data center. Adoption of SDN in cloud computing can address the networking problems raised from the aforementioned characteristic of clouds, which will be detailed later in Section 1.1.

A conceptual architecture of a cloud data center is depicted in Figure 1.1. Cloud ten-

ants can rent virtualized resources, i.e., virtual machines (VMs) connected via virtual networks, from providers where the tenants can deploy various types of applications including web, streaming, big data, or scientific applications. The virtual resources are provisioned over the physical resources consisting of compute hosts (physical machines) and network switches. Once VM and virtual network requests are submitted to the data center, admission is determined by the provider to fulfill the application request. Physical compute and network resources are jointly provisioned to provide virtual machines and networks through Compute Manager and SDN Controller, which are in charge of managing to compute nodes and networking switches respectively. Additionally, resource monitors update the current utilization and status of compute and network resources periodically which provide essential information for resource provisioning.

Efficient resource provisioning of both compute and network resources are necessary to guarantee the Service Level Agreement (SLA) for customers and to ensure a different level of reliability and QoS requirements. For instance, QoS-critical applications such as medical software for cyber surgery, IoT applications for real-time disaster management, or deadline constrained scientific applications, need more strict policies in cloud resource management, while non-critical applications may have a loose requirement. With the mixed applications sharing the data center with different QoS requirements, cloud providers should provision the resources efficiently to satisfy various QoS requirements of different applications, while minimizing the operating cost, such as electricity usage.

Providing such QoS in clouds while minimizing operational cost is challenging, because resources in a data center are shared by various tenants and applications which are often over-booked to save the resource usage. As an example, resources are oversubscribed in many data center designs to reduce the cost of the data center [4]. With oversubscribed resources, however, performance degradation may occur possibly followed by SLA violation which results in occurring penalty and reducing profit for cloud providers. Thus, efficient resource provisioning must be employed in cloud data centers to provide QoS fulfillment and SLA guarantee, while reducing energy consumption and operational cost.

This thesis tackles *joint resource provisioning problem* for both computing and networking resources in cloud data centers using SDN technology. SDN is exploited for dynamic

network configuration, bandwidth allocation, and network monitoring in conjunction with computing resource management techniques. We propose a taxonomy and comprehensive survey in the area and a modeling and simulation of SDN-enabled cloud data centers. We also present joint computing and networking resource management algorithms for energy efficiency and QoS satisfaction, and a working prototype of integrated control system to manage both resources in a data center.

The rest of this chapter details the background of SDN-enabled cloud computing and discusses research problems, evaluation methodology, contributions, and organization of the thesis.

## 1.1 Background

This section describes the fundamentals of cloud computing, Data Center Network (DCN), Software-Defined Networking (SDN), and SDN-enabled cloud computing.

### 1.1.1 Cloud Computing

Cloud computing has been studied and implemented for many years and can be considered as three services: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [81]. SaaS is to provide complete software to cloud customers, which includes cloud-based email services, social network services, scheduler services, and basically any programs running in the cloud. PaaS resides on the underlying layer where application developers can use the platform services provided by clouds. The software in SaaS may or may not use the platforms provided by PaaS provider. IaaS is the most fundamental which provides virtual machine servers and the related infrastructure to cloud customers. The infrastructure can be used for any purpose by the customers, e.g., for deploying their own platforms or developing a software running on the virtual machines. SaaS and PaaS can be implemented upon the usage of IaaS.

In this thesis, we tackle problems and issues in managing cloud infrastructure on IaaS level, considering both computing and networking resource provisioning. In order to provide cloud services, the provider should build and maintain one or more large-scale cloud data centers where tens of thousands of physical hosts are connected through

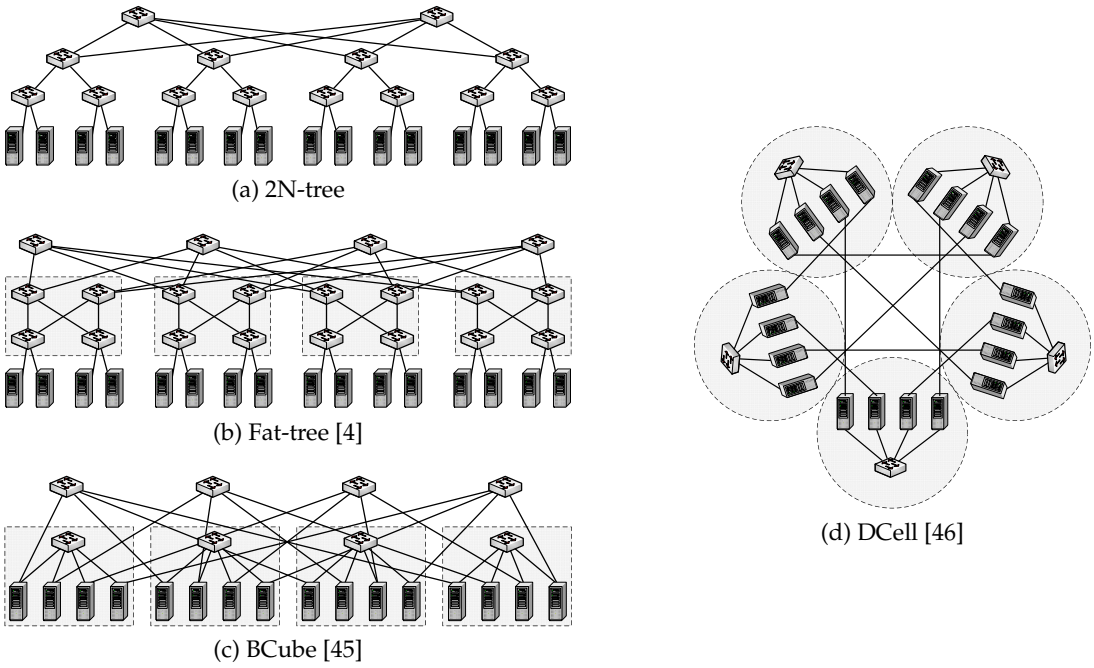



Figure 1.2: Data center network topologies.

thousands of network switches. With the high complexity of network connections in large scale, the provider has to consider the data center network with a different perspective from the traditional network.

### 1.1.2 Data Center Network (DCN)

Host servers in a cloud data center are interconnected through network links and switches, composing of DCN. Typical DCN architecture is made up of three-tier topology, which consists of edge, aggregate and core switches on each layer. Host servers within a rack are connected to one or more edge switches, i.e., Top-of-Rack (ToR) switches, and edge switches are connected to upper tier aggregate switches. Each of the aggregate switches connects to multiple core switches of the top tier.

Since DCN has a complex structure with the massive number of hosts and networking equipment, performance and scalability issues arise. The high scale brought inflexible and insufficient network bandwidth which makes researchers find an alternative topology to the canonical 2N-tree. Al-Fares et al. [4] suggested fat-tree topology that shares a similar approach to Clos network [24] to provide redundant network connection and en-

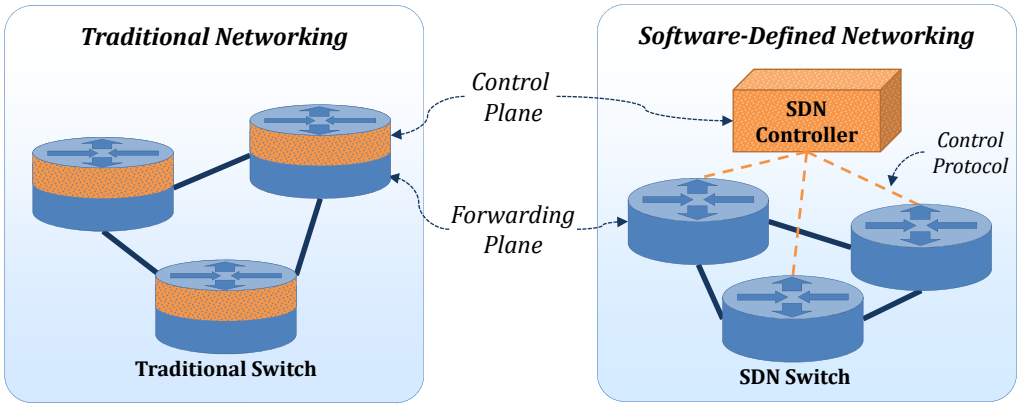



Figure 1.3: Comparison of SDN and traditional networking.

sure higher availability. In fat-tree, there are multiple links between switches that make a redundant and high-available network. Guo et al. presented DCell [46] to address scalability and performance issues of the traditional three-tier topology. BCube [45] was also proposed by the same authors where hosts were serving as both end hosts and relay nodes that transfer network data to another host. BCube was designed for shipping-container based modular data centers. The aforementioned topologies are shown in Figure 1.2. More recently, a hypercube based DCN topology called ExCCC was introduced in [131]. ExCCC substituted each node in Extended Hypercube network with a cycle and employed link removal for scalability and energy efficiency.

### 1.1.3 Software-Defined Networking (SDN)

SDN is a new paradigm in networking that manages the entire network at the conceptually centralized programmable software controller. With the central view, the controller can manage network flows more efficiently and dynamically. SDN provides fine-grained network configuration dynamically adaptable to the network condition. Figure 1.3 shows the conceptual difference between a traditional network and SDN. In the traditional network, the control plane is placed on top of the forwarding plane in each network device. The discrete decision is made by each switch based on the information gathered from the neighbor devices in a distributed matter. Although the network information is shared among devices, the decision is made solely by each device which can increase the complexity and behavioral unpredictability of the entire networks.

On the other hand, SDN centralizes the control plane into a software controller which can observe the entire network. With centrally gathered information, the controller can make more straightforward decisions and deploy it onto forwarding devices efficiently and dynamically. The control logic can be easily programmable in the controller and send control packets to the switches with SDN protocols. Thus, networking behavior becomes highly customizable depending on different purposes and goals.

As the adhesion between control logic and forwarding hardware has been broken up in SDN, it opened up more opportunities to universities and research institutes in developing new networking protocols, traffic management, virtualization, and software technologies. It led to the introduction of OpenFlow which is collective results of a number of universities and becomes a de-facto standard protocol for SDN controllers [78]. Many controllers have been introduced and actively developed based on OpenFlow, including NOX [44], Floodlight [100], RYU [87], and OpenDaylight [92].

SDN has also opened up innovative opportunities on the way of developing the control logic of networks. In traditional networks, the network control logic is solely developed by large switch vendors, such as Cisco or Juniper, as it is tightly coupled with the networking hardware. Although individual developers and users could customize configurations and settings of the hardware, it was still limited to the control logic equipped with the hardware subordinate to the vendor. With the introduction of SDN and its decoupled concept of the controlling software from the forwarding hardware, anyone can easily develop and test a new control logic by creating an SDN controller software. As the forwarding hardware is separated and fully controlled by the controller software in SDN, any control logic can be designed and implemented as long as it follows SDN standard, such as OpenFlow. Thus, SDN has fostered innovations in the development of various network control logic that would have been much limited in traditional networks.

### 1.1.4 SDN-enabled Cloud Computing

SDN features can bring enormous benefits to cloud computing, such as adapting to dynamically changing workload, programmable control plane, network security enhancement, network virtualization, global view of DCN, integration of hypervisor and cloud computing manager.

The nature of cloud computing that supports multi-tenants in complex networking has formed SDN with the desire of scalability and cost-efficiency [36]. As SDN can manage network forwarding plane with centralized control plane, network elements become programmable and controllable dynamically. With the ability of SDN controllers monitoring the entire network, various research projects have been conducted [8,20,60,132] including bandwidth allocation per flow, traffic consolidation, and dynamic configuration, in order to improve QoS and energy efficiency. The dynamics of cloud computing require quick response to manage traffic demands in real-time. In industry, Google started deploying SDN in its cloud data centers and their private wide-area network (WAN) between data centers as described in their papers [53,54,118].

## 1.2 Research Problems and Objectives

This thesis aims at addressing research questions on provisioning cloud resources exploiting SDN technology in a multi-tenant cloud data center. More specifically, it can be stipulated as:

*How to efficiently provision computing and networking resources simultaneously in a data center with SDN technology in terms of:*

- **QoS satisfaction** – resources should be provisioned fulfilling various QoS requirements from multiple tenants and applications in a cloud data center.
- **SLA enforcement** – cloud providers should allocate enough resources for tenants as per agreement in SLA.
- **Energy efficiency** – energy consumption should be maintained as low as possible in resource provisioning.
- **Request dynamicity** – resources should be provisioned dynamically to accept more requests from tenants.

To tackle the research questions formulated above, the following objectives have been identified:

- Review the state-of-the-art in the area of SDN-enabled cloud computing to gain systematic understanding and identify the gap in the area.
- Model SDN-enabled cloud data center components and their interactions for resource provisioning method implementation and performance evaluation.
- Propose a novel joint computing and networking resource provisioning method aware of QoS, SLA, operational cost, workload dynamicity, and energy efficiency.
- Propose a joint resource allocation algorithm for heterogeneous resource configuration in a data center.
- Implement and evaluate the proposed algorithms in realistic environment to compare with the result from the state-of-the-art.
- Design and implement an integrated control platform to jointly manage both computing and networking resources.

## 1.3 Evaluation Methodology

The proposed approaches in this thesis have been evaluated with two methodologies: *discrete-event simulation* and *empirical system*. Simulation is a common method for evaluating a large-scale and complex system, such as cloud data centers, that is impractical to test in a real system due to the cost and management issues. In this thesis, we developed an extended version of CloudSim [18] to model SDN features, including dynamic network reconfiguration, bandwidth allocation, and network monitoring, in a cloud data center. The simulation framework supports reproducible experiments for large-scale cloud infrastructure with different preset configurations in short time. For a realistic test in simulation, we exploited real traces such as Wikipedia [117], and 3-tier web application models [33].

We also implemented the proposed method in a small-scale empirical system for validation and evaluation. It is important to show the feasibility and effectiveness of a proposed method in the real-word scenario. The system prototype was designed and implemented using OpenStack cloud management platform [93] and OpenDaylight SDN

controller [92] which is detailed in Chapter 6. Performance evaluation in the empirical system was conducted with Iperf [51] and WikiBench [117] tools.

## 1.4 Thesis Contributions

The main contributions of this thesis can be divided into 4 categories: literature review and classification of this area, modeling SDN-enabled cloud data centers in a simulation framework, novel algorithms for joint resource provisioning, and implementation of an integrated control framework for managing both computing and networking resources. The **key contributions** of the thesis are:

1. A taxonomy and literature review of SDN usage in the context of cloud computing infrastructure management.

2. Formalized model and simulation framework for SDN-enabled cloud computing:

    - A model of SDN architecture to separate a control plane from a forwarding plane, supporting dynamic forwarding rule update, per-flow bandwidth allocation, user-defined network control logic, and network performance monitoring.
    - A performance model of SDN-enabled network switches capable of measuring bandwidth utilization and energy consumption.
    - A formal definition of requirements and objectives of SDN-enabled cloud computing architecture.
    - An event-driven interaction model between computing hosts and networking switches in an SDN-enabled data center.
    - An abstracted workload consisting of a series of computing and networking user requests.
    - A CloudSim based discrete-event simulation framework supporting aforementioned features.
    - Experimental validation and evaluation of the proposed simulation framework by comparing the simulation results with the observed performances

from practical systems.

3. Novel joint computing and networking resource allocation algorithms for SDN-enabled cloud data centers:

   - Formulated joint resource allocation problem aware of energy efficiency and SLA fulfillment.
   - Energy and SLA aware resource overbooking algorithms that jointly migrate and consolidate VMs and network traffics.
   - A priority-aware VM allocation algorithm (PAVA) that provision cloud resources differentiated by an application priority.
   - A network bandwidth allocation method (BWA) to allocate the requested bandwidth for a priority application.
   - Power model implementation in a simulation framework to estimate the energy consumption in a data center.
   - Topology-aware collective VM placement algorithm in a heterogeneous cloud data center (TOPO-Het).
   - Performance evaluation on empirical testbed and simulation framework in comparison with the state-of-the-art baselines with real-world traces.

4. Software implementation of an integrated control framework for joint resource management of cloud and SDN:

   - An architecture and system design of a software framework to simultaneously manage both cloud and network resources in a data center.
   - An implementation of the integrated control platform in Python based on OpenStack cloud control platform and OpenDaylight SDN controller.
   - An implementation of various algorithms for joint computing and networking resource allocation upon the integrated control platform.
   - A deployment of the control platform on a testbed with a customized fat-tree topology consisting of 8 OpenStack compute nodes connected through 10 OpenVSwitch switches.

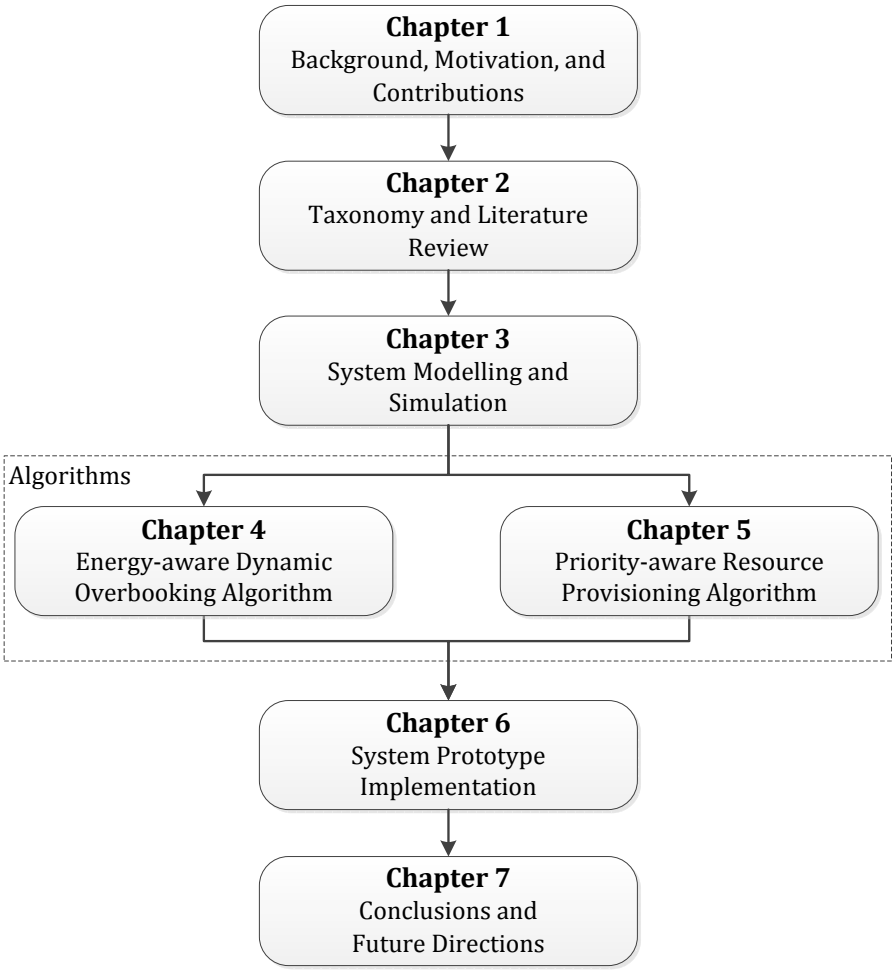



Figure 1.4: The thesis organization

- Experimental validation and evaluation of the deployed platform with a realistic benchmark tool (WikiBench [117]) using Wikipedia traces.

## 1.5 Thesis Organization

The structure of the thesis chapters is shown in Figure 1.4, which are derived from several publications published during the PhD candidature. The remainder of the thesis is organized as follows:

- Chapter 2 presents a taxonomy and literature review of SDN usage in cloud computing. This chapter is derived from:

  - **Jungmin Son** and Rajkumar Buyya, "A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing," *ACM Computing Surveys*, vol.51, no.3, article 59, 2018.

- Chapter 3 presents a modeling and simulation environment of SDN-enabled cloud data centers. This chapter is derived from:

  - **Jungmin Son**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, Xiaohui Ji, Young Yoon, and Rajkumar Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015)*, Shenzhen, China, May 4-7, 2015.

- Chapter 4 proposes a novel dynamic overbooking algorithm for energy efficiency and SLA fulfillment based on correlation analysis. This chapter is derived from:

  - **Jungmin Son**, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "SLA-aware and Energy-Efficient Dynamic Overbooking in SDN-based Cloud Data Centers," *IEEE Transactions on Sustainable Computing (T-SUSC)*, vol.2, no.2, pp.76-89, April-June 1 2017.

- Chapter 5 proposes a priority-aware computing and networking resource provisioning algorithm in SDN-clouds. This chapter is derived from:

  - **Jungmin Son** and Rajkumar Buyya, "Priority-aware VM Allocation and Network Bandwidth Provisioning in SDN-Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, 2018 (accepted, in press).

- Chapter 6 describes a prototype system of the integrated control platform for joint cloud and SDN management based on OpenStack and OpenDaylight. This chapter is derived from:

  - **Jungmin Son** and Rajkumar Buyya, "SDCon: Integrated Control Platform for Software-Defined Clouds," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (accepted, in press).

- Chapter 7 summarizes the thesis with a discussion on future directions. It is partially derived from:

  - **Jungmin Son** and Rajkumar Buyya, "A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing," *ACM Computing Surveys*, vol.51, no.3, article 59, 2018.

# Chapter 2
# Taxonomy and Literature Review

*This chapter proposes a taxonomy to depict different aspects of SDN-enabled cloud computing and explain each element in details. The detailed survey of studies utilizing SDN for cloud computing is presented with focus on data center power optimization, traffic engineering, network virtualization, and security. We also present various simulation and empirical evaluation methods that have been developed for SDN-enabled clouds. Finally, we analyze the gap in current research and propose future directions.*

## 2.1 Introduction

TO overcome the shortcomings of traditional networks, cloud data centers started adopting software-defined networking (SDN) concept in their DCN. SDN provides a centralized control logic with a global view of the entire network at the central controller and dynamically change the behavior of the network. It can also adjust the network flow dynamically by the controller which is well fitted for the dynamic nature of the cloud service. Giant cloud providers such as Google already adopted SDN concept in their data center to increase the scalability and manageability [118].

Although many surveys and taxonomies have been presented in cloud computing and SDN contexts, each of them has been addressed a specific problem in the area. For example, Toosi et al. [113] presented a survey focusing on inter-connected cloud computing. The paper includes inter-operability scenarios with multiple data centers and detailed explanation of various approaches to operate and use inter-connected cloud data centers. The article described networking challenges for inter-clouds in a sub-section, but

This chapter is derived from: Jungmin Son and Rajkumar Buyya, "A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing," *ACM Computing Surveys*, vol.51, no.3, article 59, 2018.

the primary focus was on the wider issues for integrating multiple cloud data centers as a cloud broker's perspective. Mastelic et al. [74] also presented a survey in energy efficiency for cloud computing. A systematic category of the energy consumption in cloud computing has been suggested in the context of hardware and software infrastructure in clouds. The authors also included a networking aspect emphasizing on DCN, inter-data center network, and end-user network. A comprehensive survey has been presented in various aspects of energy efficiency including networks, however the paper is in lack of SDN context. Jararweh et al. [56] provided details of Software-defined clouds focusing on systems, security, storage, and networking, but with more focus on the system architecture rather than the individual research works in SDN-clouds.

In this chapter, both SDN and cloud computing are considered as the survey topic. Among the enormous studies conducted in both distributed computing and networking disciplines for cloud computing and SDN respectively, we select the state-of-the-art considering both aspects simultaneously. We emphasize on SDN utilization and challenges in the context of cloud computing.

This chapter is organized as follows: in Section 2.2 we clarify terms and definitions to be used in this chapter and throughout the thesis. Section 2.3 provides different architectures of SDN-enabled cloud computing proposed in the literature, followed by Section 2.4 that describes a taxonomy of the usage of SDN in cloud computing in various aspects. In Section 2.5, comprehensive surveys have been undertaken to find the achievement and the challenges in SDN usage in clouds in the context of energy efficiency, performance, virtualization, and security enhancement. The following section presents a survey of simulation and empirical methods developed for evaluation of SDN-enabled cloud computing (Section 2.6), and summarizes the chapter in Section 2.7

## 2.2 Terms and Definitions

By emergence of adopting SDN in cloud computing context, several terms have been presented in order to capture the architectural characteristic of the new system. In this section, we propose the organized terms for different purposes based on the collective survey, presenting in the order of scope from narrow to wide terms.

*SDN-enabled Cloud Data Center (SDN-DC)* is to provide SDN features within a cloud data center. On the basis of the traditional DCN architecture, SDN-DC replaces traditional networking equipment with SDN-enabled devices. In SDN-DC, every networking component in a data center is capable of performing SDN functions which can bring all the aforementioned benefits to a cloud data center. This architecture focuses within a data center and excludes the inter-networking part outside of a data center.

*SDN-enabled Cloud Computing (SDN-clouds or SDN-enabled clouds)* refers to not only SDN-DC, but also inter-cloud networking that expends the SDN usage across multiple data centers and wide-area network (WAN). Thus, SDN benefits can be applied to inter-networking domains, such as a data center to data center network or an end-user to data center transport network. In this chapter, we focus on SDN-clouds to build our taxonomy and analyze the state-of-the-art.

A broader term *Software-Defined Cloud Computing (SDC or Software-Defined Clouds)* has been proposed by Buyya et al. [16] and Jararweh et al. [56] where not only networking but also all infrastructure components in cloud computing are considered to be software-defined. The approach is to build fully automated cloud data center optimizing configurations autonomously and dynamically. Buyya et al. [16] extended the concept of virtualization from a virtualized server (VM) to all other elements in cloud data centers. The core technologies to enable SDC include server virtualization, SDN, network virtualization, and virtual middleboxes. With the recently evolved technologies, the reconfiguration and adaptation of physical resources in SDC has become more feasible and simple to be implemented in practice. The proposed architecture was implemented and evaluated in the simulation environment. Jararweh et al. [56] also studied a system focusing on various aspects of SDC including networking, storage, virtualization, data centers, and security. The authors also built an experimental setup for SDC with the inspected elements in the survey to show the effectiveness of the proposed software defined cloud architecture.

SDC is more conceptual as it is proposed for research purposes and has not yet been explored extensively. Therefore, in the survey, we focus on SDN-clouds to depict the state-of-the-art and SDN usage in cloud computing.

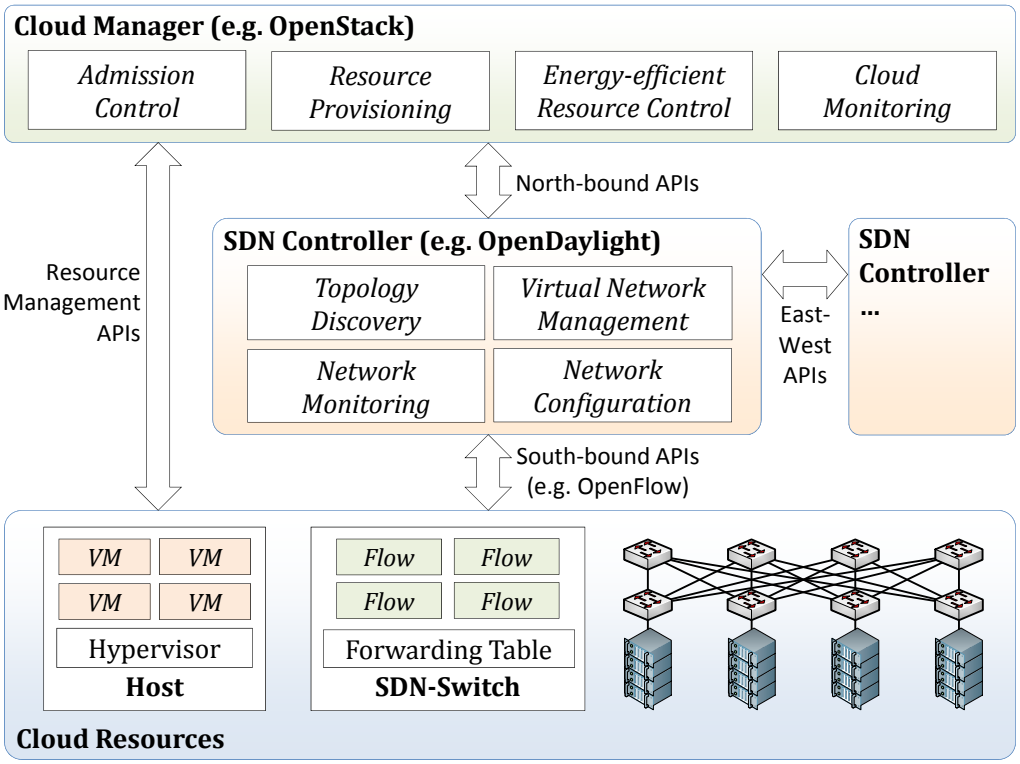



Figure 2.1: SDN-enabled cloud computing architecture.

## 2.3 SDN-clouds Architectures

Figure 2.1 shows the abstract architecture of common SDN-enabled cloud computing systems derived from literature [9, 17, 129].

*Cloud Manager* is managing the tenants and resources of the entire clouds. It controls the incoming requests from tenants such as a VM creation and provisions cloud resources to provide the cloud service. Energy-efficient resource management and monitoring of resources are also performed by Cloud Manager. OpenStack [93] is an open-source example of a Cloud Manager widely used to build private clouds.

Network-related functions are controlled by *SDN Controller* which is connected to the Cloud Manager through north-bound APIs. SDN features, such as network topology discovery, network monitoring, virtual network management, and dynamic network configurations, are enabled through SDN Controller. Multiple SDN Controllers can be functional to increase scalability by inter-communicating via East-West-bound APIs.

*Cloud Resources* consist of compute and networking resources. Compute resources

(hosts) are provisioned by the Cloud Manager to run VMs on a hypervisor, whereas networking resources (switches) are managed by SDN Controller through updating forwarding tables in switches via South-bound APIs (e.g., OpenFlow). Note that the SDN Controller also manages networking functions in hosts, for example virtual switches, for network virtualization and VM traffic management.

Meridian is a SDN platform for cloud network services proposed by IBM [9]. In typical cloud network services, users are required to set up switches, subnets, and ACLs to be used by cloud application. Meridian adopts service-level network model with connectivity and policy abstractions and integrates the high-level application provisioning closely to the cloud network through SDN's programmable network concept.

PDSDN is a policy exchange scheme between SDN controller and the cloud manager to simplify the interaction of cloud tenants and the controller [31]. The proposed scheme sends network policies to the controller and processes them with the information of tenants' requirements and priorities. The prototype is implemented on Floodlight and validated on OpenStack.

More recently, Mayoral et al. [76] presented cloud architectures with a single SDN controller and multiple controllers. The system orchestrates networking services in a data center with the capability of SDN. In order to find the differences in one or multiple controllers in a data center, the authors evaluated the proposed architecture on OpenStack and OpenDaylight [92] with the integration of SDN.

The same authors also proposed an end-to-end orchestration architecture of cloud and networks resources which manages both resources in distributed cloud data centers [75]. Virtual infrastructures in clouds are dynamically allocated and managed through the manager. The authors modeled the VM allocation problem as a graph mapping with the parameter of switches, hosts, and network links. The paper also includes heuristic algorithm to find minimum number of hosts with enough capacity and selects the shortest path between hosts. The orchestration architecture is validated on empirical testbed with OpenStack [93] and OpenVSwitch, whereas the algorithm is evaluated in simulation.

Fichera et al. [38] presented an experimental test-bed setup for SDN orchestration across edge, cloud, and IoT domains in 5G service. The platform consists of separate orchestrator(s) for cloud, IoT, and SDN. SDN orchestrator manages not only IoT and

cloud networks, but also the transport network between them. On top of individual orchestrators, Service orchestrator oversees service provisioning for applications based on the users requirements. The proposed system is implemented in a test bed consisting of Mininet for edge-network, OpenStack cloud, and ONOS controller.

Control Orchestration Protocol (COP) was proposed by researchers from multiple institutions to allow managing heterogeneous SDN control planes and clouds [77]. While OpenFlow specifies control messages between SDN controller and switches at south-bound of the controller, COP is in charge of orchestrating among SDN controllers in different networks as well as cloud controllers working at north-bound of controllers. COP aims at unified transport API for orchestrating different transport network in inter-cloud environment. The protocol provides an abstraction for resource provisioning, topology discovery, and path computation. It is validated through two proof of concept experiments for network resource provisioning and recovery.

## 2.4 Taxonomy of SDN usage in Cloud Computing

Several methods are proposed for utilizing SDN in cloud computing in different research areas. We carried out a detailed study of research in SDN-enabled cloud computing and propose a taxonomy (shown in Figure 2.2) to capture various elements of SDN usage. The taxonomy is explored in the context of objective, method scope, target architecture, application model, resource configuration, and evaluation method.

### 2.4.1 Objective

Cloud providers deploy SDN in their data centers for different purposes, and researchers consider various objectives in their studies. We categorize the objectives into four types, which are used as main categories to classify the surveyed works in Section 2.5.

*Energy efficiency* is one of the most commonly studied objectives for SDN-enabled cloud computing. As cloud data centers consume enormous amount of electricity globally, energy saving methods for data centers have received more attention over the last decade. Energy consumption of a data center mainly relies on host servers, networking devices, and cooling system [106], and is proportional to utilization level [94]. If all

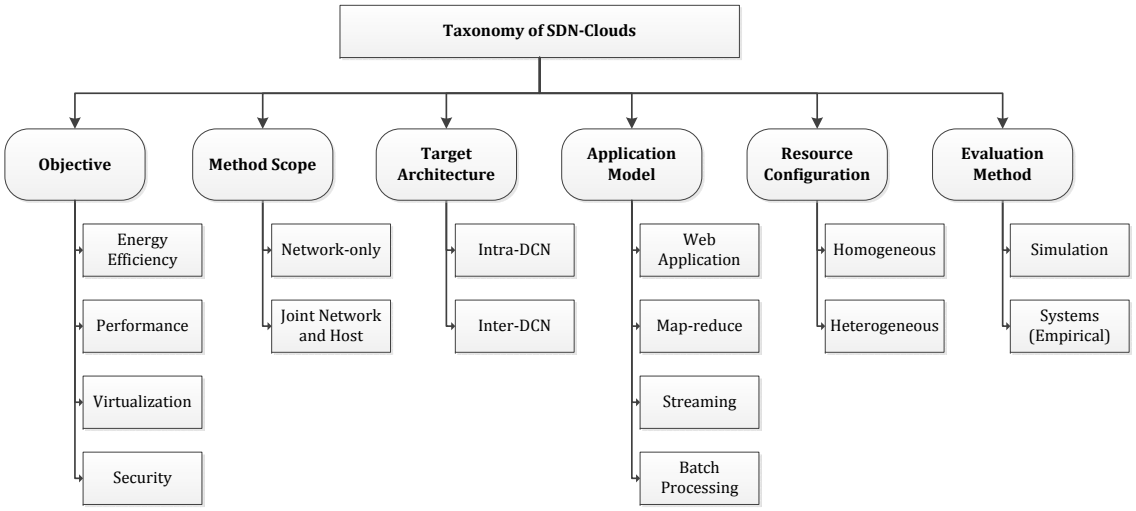



Figure 2.2: Taxonomy of SDN usage in cloud computing.

servers in one server room are not utilized, the servers and the cooling system of the room can be switched off to consume less power. With SDN, network elasticity is possible in addition to the computing elasticity. Network traffic can be consolidated into the smaller number of switches with SDN for the duration of low network utilization, which can provide possibilities to turn off unused switches. Not only the network itself but also the joint optimization of network and host is possible for further energy savings.

*Performance* can be improved by providing fine-grained control over the entire network in SDN. As SDN observe the entire network in the centralized controller, it can easily improve network performance in case of traffic congestion by altering the path of the flows. For instance, SDN can be used as one of the techniques to reduce the VM deployment time in a large-scale data center to boot up a large number of VMs. VM deployment time can be decreased by reducing the amount of data transferred, the bottlenecks at the VM image repository, the CPU load at the target physical machine, or the boot-up time of the VM [105]. The dynamic bandwidth allocation feature between the image repository and physical machines can reduce the image transfer time.

Performance includes not only network throughput and latency, but also availability and QoS. The data center network must be available throughout the whole operating hours, and SDN can help to increase availability. In case of a hardware malfunction on a network path, SDN controller can easily notice and modify the following flows to use an alternate path through the programmable switches with appropriate DCN topol-

ogy configuration. QoS includes providing guaranteed bandwidth to a specific user that was almost impossible in the traditional network as network medium must be shared by multiple users in cloud data centers. SDN, however, can manipulate the network path to provide a dedicated bandwidth to a certain type of users. To improve the QoS, a congested link can be detected, and its traffic can be distributed by changing the path of the network flows. Network performance can be quantified with the measurements such as throughput, latency, and error rate. Other works considered workload acceptance rate and network blocking probability to measure the performance of the resource management scheme [42]. Optimal joint orchestration of cloud and network resources can increase VM acceptance ratio which leads to increasing the number of VMs hosting in a cloud data center with the limited resource, and thus maximizing the benefit cloud provider.

*Virtualization* can be further realized for DCN by adopting SDN in clouds. Cloud computing is implemented based on a variety of virtualization technologies which can make it possible to run virtual machines (VMs) in clouds [16]. Hypervisors running on physical hosts can virtualize the host resources such as CPU, memory, and storage. SDN is capable of virtualizing network resources. Network resources such as switches and physical links can be virtualized and leased to multiple tenants of a cloud data center. Network embedding problem, for example, is to map the network flows by different tenants into physical resources by virtualizing the infrastructure [39]. Network Function Virtualization (NFV) is another example of SDN usage for virtualization. NFV conventionally provided by middleboxes equipped with dedicated hardware, such as firewall, load balancer, and intrusion detectors. The hardware middleboxes are expensive to purchase and difficult to manage and scale. Several researchers attempted to substitute the hardware middleboxes with a series of SDN switches with specific controller program [101, 108, 120].

*Security* is an important objective in networks which have not yet extensively studied in integrating SDN with cloud data centers. There are two research directions in the usage of SDN in clouds: 1) securing networks by utilization of SDN and 2) protecting core components of SDN from attacks. On one hand, SDN technology is widely adopted in cloud data centers to protect and prevent a Distributed Denial of Service (DDoS) attack where the overwhelming network traffic is generated to disable online services. By uti-

lizing SDN features such as traffic analysis, centralized control, and dynamic update of network rules, it is easier to detect and react to DDoS attacks [129]. On the other hand, SDN controller must be protected in high level of security as its central operation can manage the entire data center network. It is also necessary to secure the management packets communicating between the controller and its switches.

### 2.4.2 Method Scope

Here, we propose a taxonomy based on the scope of methodology. As we are discussing SDN-clouds, all methods in our taxonomy are fundamentally targeting the networking functionality in cloud computing. While some researchers consider only networking in their studies, others consider both hosts and networking at the same time.

The scope of *network-only* approaches solely lies within the networking function in cloud computing without consideration of hosts or VMs. These approaches try to collect the network data and alter the forwarding rules in clouds using SDN to solve the research problem. For example, most approaches targeting network performance improvement naturally consider only networking resources in clouds [5, 32, 52, 54].

*Joint* approaches take not only networking but also hosts and VMs into account simultaneously to orchestrate both resources. Optimizing hosts and networks resources to reduce power consumption of cloud data centers is an example of joint approaches. Before introducing SDN, traditional approaches dominantly consider only hosts for data centers' power optimization [11], whereas joint optimization has been enabled by SDN and recently appealed to many researchers [27, 135]. A joint resource allocation method proposed by Souza et al. [30] also tries to deliver better network performance for VMs hosted in SDN-clouds.

### 2.4.3 Target Architecture

As SDN can bring flexibility and controllability to the network with its programmable controller, it applies to various network architectures in cloud data centers.

In many research studies, SDN is applied to the connection between hosts within a data center (*intra-DCN*). SDN-enabled switches replace traditional switches connecting

from one host to another. Networking in a data center can be optimized based on the current utilization of network traffic, with the bandwidth usage of incoming and outgoing packets.

We can also find studies focusing on *inter-DCN* architecture which can optimize network connections between geographically distributed data centers [54, 79, 96]. Cloud providers operating multiple data centers can benefit from SDN by developing their own SDN controller to manage the WAN between the data centers.

Others proposed orchestration methods considering both inter and intra-DCN architectures [15, 38, 70] to interconnect and manage multiple cloud and/or edge data center networks. The upper layer controller oversees individual SDN controllers within geographically distributed data centers as well as in transport network interconnecting data centers. With the orchestration of inter and intra-DCN management, SDN can provide end-to-end QoS for applications and a virtual network environment for VMs hosting in different data centers.

### 2.4.4 Application Model

Researchers can consider a specific application model to benefit from SDN on their studies, or a common generic approach which does not consider any specific application model. For instance, improving network performance between cloud data centers with SDN is rarely related to the specific application running on the clouds [54]. In contrast, consolidating VMs and their network traffics using correlation analysis of the utilization can be highly relevant to the application running on the VMs [132, 135]. Thus, it is worth to consider various application models before exploring the research works.

*Web application* model consists of multiple tiers such as load balancers, application servers, and databases which communicate with the upper and lower tier servers. As the end-users of the web application expect a spontaneous response from the internet, response time and latency are crucial in this model. With the popularity of web applications hosted in clouds, many researchers consider web application model for energy efficiency research [125, 135].

With *map-reduce* model, a big task is split into smaller tasks and processed on multiple machines simultaneously. In this model, a large amount of network bandwidth and

computing power are necessary to process the workload where the underlying network support is crucial. For instance, network burst between mappers and reducers can be a bottleneck as the mappers send the processed data to the reducers at the same time if the tasks are equally distributed and processed at a similar speed. With a proper control logic in SDN, the burst traffic can be distributed dynamically which can eventually decrease the effect of the bottleneck [25].

*Streaming* model is for the application that generates data continuously that leads to transfer large flows of data across networks. Streaming applications include not only audio and video streaming application, but also the applications for recently introduced Internet of Things appliances both of which need a seamless network connection for real-time transmission. With the emergence of Internet of Things (IoT) where every device with sensors can generate and transfer its data to clouds for analysis and data mining, the support for streaming applications is acquiring more attentions. As streaming applications generate an enormous amount of data continuously, an enhanced methodology is necessary for seamless network transmission and real-time processing in clouds. Researchers have studied both multimedia streaming applications [96] and IoT stream data [126] for SDN-clouds.

In *batch processing*, jobs are processed in order. It may need to transfer data between jobs in which network transmission is necessary with a constant and reliable connection. In most batch processing tasks such as scientific computation, providing constant bandwidth is more crucial than the latency or the response time. Thus, the network controller should provide a different approach. For example, Seetharam et al. [107] presented the SDN-enabled network management system for multiple clouds in the university campus for scientific workloads.

### 2.4.5 Resource Configuration

Resource configuration refers to the heterogeneity of hardware resources composing of a cloud data center. Resource configuration should be considered in designing a new method for SDN-clouds, because the result of the method may differ depending on the configuration. For example, a VM consolidation method for energy efficiency designed for homogeneous configuration may not be effective on heterogeneous environment, if

the algorithm did not consider the power models of various host specifications.

In *homogeneous* configuration, all the hardware in the data center is assumed as having the same specifications. Physical machines and switches in an edge rack share the same hardware specification across the entire data center, and all the racks have the identical number of hosts and switches. Homogeneous configuration is useful for research in energy efficiency in order to simplify energy modeling of hardware, because various hardware specifications make the modeling more complicated.

*Heterogeneous* configuration sets up with different types of hardware. Homogeneous configuration is impractical in a real world, as most data centers would upgrade their systems periodically and purchase newer machines to expand the capacity which results in having the heterogeneous configuration in the end. Homogeneous configuration is useful though in a research study with a limited testing environment which can alleviate the complexity of the evaluation process, thus many researchers studied with homogeneous hardware configuration for their preliminary experiments. Research on network performance for inter-DCN generally considers heterogeneous configuration as the inter-networking system is involved with various networking operators who use different networking devices. Some works also include heterogeneous configuration for the energy efficiency research in order to model the power consumption of various switch devices [48].

### 2.4.6 Evaluation Method

Evaluating cloud data centers is challenging with its high complexity made up by a tremendous number of hosts and switches, and their connections. In order to test the new approaches with limited research resources, researchers use two main methods for evaluation.

*Simulation* is the most affordable method to test a new approach with acceptable accuracy. Cloud data centers can be set up in simulators with the arbitrary configuration without much cost. It is also easy to implement the proposed approach in simulation and to evaluate the result quickly. However, the result of the simulation can be inaccurate when the configuration becomes far different from the validated environment, as the simulated results are statistically calculated from certain settings. Several simulation

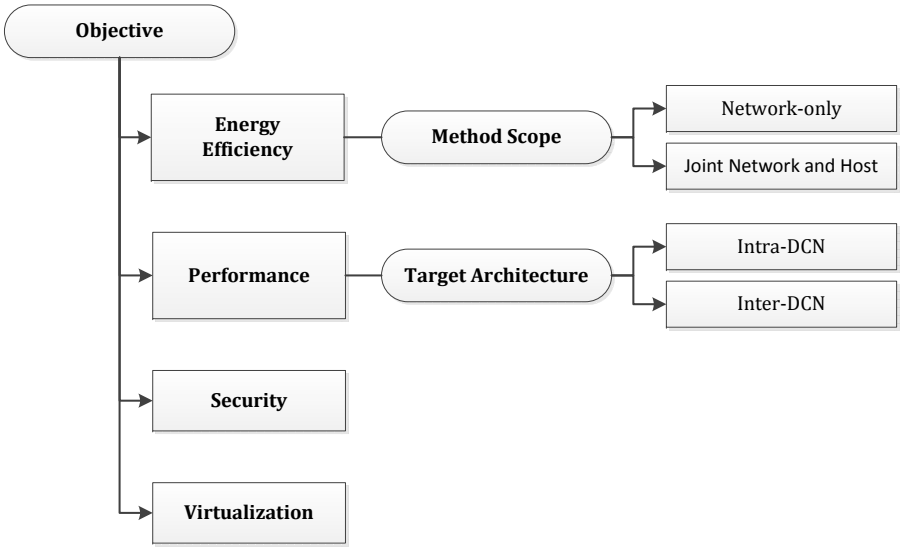



Figure 2.3: Sub-categories used for our literature review.

tools have been implemented including Mininet [68] , and more details will be explained in Section 2.6. Evaluation with the network simulator integrated with an empirical SDN controller is also considered as a simulation method although the developed SDN controller can be adapted to the real system directly, since the evaluation still relies on the simulation tool.

*Empirical* evaluation can outcome more practical results compared to the simulation as the test is undergoing on the real system. Although the result is more accurate, evaluating in large scale is impractical on the real systems, because the cost can be too expensive and the management of the cloud data center elements can be significantly complex. In several studies including ElasticTree [48], CARPO [125], and CometCloud [96] the empirical approach has been used for evaluation.

## 2.5 Current Research on SDN usage in Cloud computing

In this section, we present relevant studies conducted in SDN-enabled cloud computing following the taxonomy proposed in Section 2.4. We categorize the state-of-the-art primarily based on the main objective of the paper using the taxonomy described in the previous section. A research work typically aims to address more than one objective at the same time with a different priority. We adopt the main objective of the study for the

classification in this section. Figure 2.3 presents sub-categories used for further classification of the surveyed studies in this section. The detailed literature in each category is presented below.

### 2.5.1 Energy Efficiency

Energy efficiency is one of the utmost research topics in SDN-enabled cloud computing. Based on our observation, SDN is exploited in two ways to improve energy efficiency of cloud data centers: network optimization and joint host-network optimization. Table 2.1 presents a summary of reviewed studies for energy efficiency.

Network optimization amends network elements of DCN including switches and network adapters in hosts. Link rate adaptation is a similar approach to Dynamic Voltage and Frequency Scaling (DVFS) that lowers the link rate of a network device in case of under-utilization [1]. Dynamic topology alteration changes the network topology dynamically depending on the network traffic load [48]. Network flows in under-utilized links are consolidated into the smaller number of links resulting in the alteration of the network topology. Correlation-aware optimization considers the correlation of network flows for traffic consolidation [125]. Both dynamic topology alteration and correlation-aware optimization are feasible with the introduction of SDN in cloud data centers.

Joint host-network optimization considers both host and network simultaneously to reduce the energy consumption for VM and network placement, consolidation, and overbooking. VM placement optimizes the initial placement of VMs in order to minimize the total power consumption of the entire data center. Similarly, VM migration and consolidation consider migration of VMs after the initial placement which can consolidate VMs into a smaller number of hosts. These techniques often use a host overbooking method that places more VMs into a host than its capacity to maximize resource utilization. Instead of optimizing separately, the combined techniques are applied in order to consolidate both network and hosts at the same time.

Table 2.1: Summary of current research on SDN usage for energy efficiency in cloud computing.

| Project | Description | Author | Organization |
|---|---|---|---|
| ElaticTree | Traffic consolidation | Heller et al. [48] | Stanford University, USA |
| CARPO | Correlation analysis, traffic consolidation | Wang et al. [125] | The Ohio State University, USA |
| DISCO | Distributed traffic flow consolidation | Zheng et al. [134] | The Ohio State University, USA |
| FCTcon | Dynamic control of flow completion time in DCN | Zheng and Wang [133] | The Ohio State University, USA |
| GETB | Energy-aware traffic engineering | Assuncao et al. [29] | University of Lyon, France |
| VMPlanner | VM Grouping, VM consolidation | Fang et al. [35] | Beijing Jiaotong University, China |
| VM-Routing | VM placement problem to a routing problem, shortest path | Jin et al. [60] | Florida International University, USA |
| PowerNetS | Correlation analysis, VM placement, migration | Zheng et al. [132, 135] | The Ohio State University, USA |
| S-CORE | VM management considering host-network | Cziva et al. [27] | University of Glasgow, UK |
| QRVE | Energy efficient VM placement and routing | Habibi et al. [47] | Amirkabir University of Technology, Iran |
| ODM-BD | Big data workload slicing in edge-cloud environment | Aujla et al. [6] | Tharpar University, India |

### A Network Optimization

Several studies have been conducted in optimizing DCN to reduce the energy consumption of data centers.

ElasticTree [48] is presented that dynamically changes DCN topology and adjusts links and switches for power saving. It is designed for the fat-tree topology to consolidate traffic flows into a minimum number of links to utilize as few switches as possible. After consolidating the data flows into the smaller number of links, the unused switches can be switched off to save more energy. The authors evaluated the proposed method for trade-offs between performance, robustness and energy savings using real traces collected from production data centers. An empirical prototype was implemented with latency monitor and feeding various traffic patterns for evaluation.

In CARPO [125], correlation between traffic flows is considered in addition to the traffic flow consolidation. If the traffic flows are less correlated to each other, those flows can

be placed on the same network to maximize the overall utilization. By placing less correlated traffic flows together, each link can hold more flows in the limited capacity which leads to turn more switches off. Wikipedia traces were observed for correlation analysis, and the authors found that different data traffic does not hit the peak at the same time, and the off-peak utilization is less than the peak. Thus, off-peak utilization is used for the consolidation to maximize the energy saving while still minimizes the performance degradation. CARPO also includes link rate adaptation [1] that changes the speed of each port based on the link utilization. The proposed method is implemented on the empirical prototype in small scale and on a simulation environment for performance evaluation with Wikipedia workload. The results show that the proposed correlation-aware trafic consolidation method can outperform with web application workload compared to ElasticTree.

Zheng et al. [134] proposed a distributed traffic management framework with the same correlation analysis method. The framework is capable of analyzing the correlation between flows and consolidating the flows based on the scalability, energy savings, and performance. The new approach focused more on the scalability of the correlation-aware flow consolidation and tried to reduce the size of calculation for analyzing the correlation and the decision making. The authors proposed flow or switch based traffic consolidation algorithms that optimize each flow/switch with a limited visibility on the flow or switch, instead of taking the entire DCN flows into account. As the new approach omits the less-related flows in their analysis and consolidation, it can be applicable to a larger scale DCN. The system is empirically implemented and tested on OpenFlow based switch with 2-pod fat-tree topology, as well as evaluated the performance on a simulation setup with Wikipedia web application workload.

The same authors studied a dynamic strategy to control flow completion time [133]. This study focused on the flow transmission time while achieving energy efficiency. The effort on balancing between energy saving through traffic consolidation and the performance guarantee of the network transmission time brought a new framework to control flows with different requirements and adjust bandwidth demands accordingly. It was evaluated on the real hardware system and the simulation.

Recently, Assuncao et al. [29] proposed energy aware traffic engineering methods

based on SDN services. The authors designed SDN services to manage switches and control network traffic. The idea is to switch off idle switches after redirecting network traffic into an alternate path that is enough for serving the combined network traffic. A proof-of-concept platform is implemented in a small scale real system, while the large scale evaluation was performed in simulation. The authors showed the proposed architecture could reduce energy consumption by consolidating network traffic into the smaller number of links while maintaining QoS of the network. Traffic consolidation however may incur QoS degradation in case of a network burst when the amount of packets exceeds the capacity of the link. The authors suggested a continuous and frequent monitoring of the network in SDN controller, e.g., checking the network burst every second, and showed that the system can turn the link on once it detected an over-utilized link.

All the aforementioned studies consider intra-DCN architecture focusing on the energy consumption within a data center in their network-only optimization.

#### B Joint Host-Network Optimization

More recently, researchers proposed joint host-network optimization methods considering both host and network resources simultaneously. SDN can be utilized for VM resource provisioning and optimization. Medina and Garcia explained various migration techniques of VMs in cloud data center [80]. The authors used three categories for classifying migration methods: process migration, memory migration, and suspend/resume migration. Kapil et al. [62] also presented a detailed survey on the issues on live VM migration. The authors used downtime, migration time, and transferred data size to measure the performance of the live migration techniques and categorized them into post-copy approaches and pre-copy approaches.

VMPlanner [35] also optimizes VM placement and network routing with three approximation algorithms. The joint VM placement and routing problem are formulated with three NP-complete problems: traffic-aware VM grouping, distance-aware VM-group to server-rack mapping, and power-aware inter-VM traffic flow routing. The authors formulated the problem and proved that minimization of energy consumption considering both VM placement and network routing is a NP-complete problem. The proposed solution is to integrate approximation algorithms to solve these three problems. The solution

includes VM grouping that consolidates VMs with high mutual traffic, VM group placement that assigns the VM to the rack of VM group, and traffic flow consolidation to minimize the inter-rack traffic.

Jin et al. [60] addressed joint host-network optimization problem with an integer linear algorithm. The problem is first formulated as an integer linear problem, then converted VM placement problem to a routing problem to combine both problems. To solve the problem, the depth-first search (DFS) is used to determine the best host for VM placement. The proposed algorithm is implemented on the OpenFlow-based prototype with fat-tree topology and evaluated with a number of workloads in prototype as well as a simulation environment.

More recently, PowerNetS was proposed that optimize both VM and network traffic using correlation [132, 135]. Inspired from CARPO [125], the authors extended the network optimization to joint host-network optimization. Correlation of VMs are leveraged for VM consolidation in addition to CARPO's traffic consolidation. The detailed power model is also included to estimate power consumption by a switch chassis, each port on the switch, idle power of a server, and maximum power of the server. Using Wikipedia and Yahoo traces, correlation coefficients between traffic flows were measured and presented. The authors then designed PowerNetS framework based on the observed correlation and implemented both prototype and simulation. The system was evaluated with the same workload traces from the correlation analysis, and the results were compared with non-correlation based VM placement, optimal solution, and CARPO results.

Cziva et al. [27] presented a SDN-based VM management platform for live VM migration based on the network-wide communication cost. A prototype was implemented on a real system in a cloud data center testbed. The hierarchical system architecture is presented to support OpenFlow switches and Libvirt hypervisors, and Ryu SDN controller is used for host and network topology discovery, L2 switching management, gathering statistics, and calculating link weights. They also proposed VM orchestration algorithms that perform VM migration to reduce VM-to-VM communication cost. The algorithm is recurring periodically and hierarchically to reduce the network traffic cost. It is firstly applied to the core switch layer (the most expensive communication cost), and then gradually applied to the lower layers. The evaluation showed that overall cost was converged

to the minimum after several rounds of migrations.

Habibi et al. [47] proposed a VM placement and network routing approach for energy efficiency and QoS. The system consists of monitoring and flow detection, QoS-aware routing, network topology storage, and VM placement. The algorithm exploits dynamic flow routing to maintain network QoS and VM placement and migration methods to consolidate VMs for energy efficiency. The authors explained inevitable trade-off between energy efficiency and QoS, and proposed a combined approach to address both problems. They measured network throughput, energy consumption, and utilization of the data center on a simulation environment implemented on Mininet tool.

Aujla et al. [6] proposed workload slicing decision-making scheme for data-intensive applications in multi-edge cloud environment exploiting SDN technology. They consider energy consumption, communication cost, SLA, bandwidth, and revenues in their system model for the optimal decision of data migrations between the central and edge clouds. The SDN-based controller decides flow paths between data centers and edge nodes as well as migration of data among multiple data centers. The proposed scheme is evaluated on simulation with Google workload traces.

### 2.5.2 Performance

Several approaches have been proposed to empower the network performance of cloud computing utilizing SDN technologies. We categorize them into intra-DCN and inter-DCN approaches based on the target architecture in the study. Intra-DCN approaches are to improve the performance within a data center including network throughput, latency, and availability on the network transmission. Inter-DCN approaches are mainly for improving the QoS for wide area network (WAN) between tenants and cloud data centers. Many studies [32, 54, 107, 120, 127] rely on the dynamic per-flow routing feature of SDN which can change the route of a network traffic dynamically based on the network condition, e.g., finding an alternate route in case of a network burst to meet QoS requirements. Although these approaches can improve the performance of network within/between data centers, it may introduce a new challenge incurred from the performance of SDN controllers and switches. Because the centralized SDN controller utilizes a large amount of computing and memory resources to manage all the switches in a network, it is diffi-

cult to scale out for a growing network size. We have reviewed several studies focusing on improving network performance in the following subsections. Table 2.2 summarizes reviewed works for performance in this subsection in which the detail is presented below.

### A Intra-DCN Performance Improvement

Wang et al. [120] at Princeton University proposed an OpenFlow-based load balancing technique substituting the expensive dedicated load-balancer (LB) server in a data center. A front-end LB is normally used in a data center to redirect client requests to one of application server replica for even distribution of workloads across multiple servers. Instead of using an expensive LB server, the authors proposed OpenFlow switches to perform the same load balancing in a data center, by altering the network route for each incoming traffic to be distributed across the servers. The challenge incurred in this approach is how to deal with the excessive size of traffic rules installed in a switch for each flow for different servers. The authors presented an algorithm that finds wildcard rules for the same group of flows directing to the same server, so that it can achieve a target distribution of the traffic with scalability. The proposed method was implemented and evaluated on NOX OpenFlow controller.

Ishimori et al. [52] proposed QoSFlow, a QoS management system that controls multiple packet schedulers within a Linux kernel to provide a flexible QoS controllability over the integrated SDN network. The traffic schedulers in Linux kernels, such as HTB and SFQ, can become a part of OpenFlow network in QoSFlow, which allows SDN controller to manage the QoS using the Linux traffic schedulers. QoSFlow was designed to be an extension to the standard OpenFlow switches to schedule data flows passing through the switch. The proposal was implemented and tested in commodity switches and showed that various Linux schedulers worked on OpenFlow switches. Although the maximum bandwidth has been reduced after applying QoSFlow because of the control overhead of the OpenFlow packets, the approach can provide an easy framework to implement QoS-aware SDN network in a data center with the classic Linux traffic schedulers.

Wendong et al. [126] proposed another approach to improve network QoS by combining SDN with Autonomic Networking technology which can be applied in a DCN. The proposed approach applies QoS requirements to SDN automatically by configur-

Table 2.2: Summary of current research for performance improvement in cloud computing with SDN.

| Project | Description | Author | Organization |
|---|---|---|---|
| OF-SLB | Server load balancing | Wang et al. [120] | Princeton University, USA |
| QoSFlow | QoS management system for traffic engineering | Ishimori et al. [52] | Federal University of Para, Brazil |
| AQSDN | Autonomic QoS management system | Wendong et al. [126] | Beijing University of Posts and Telecom., China |
| SDN-Orch | SDN-based orchestration for host-network resources | Martini et al. [72] | CNIT and Sant'Anna School, Italy |
| C-N-Orch | Cloud and network orchestration in a data center | Gharbaoui et al. [42] | Sant'Anna School and University of Pisa, Italy |
| Orch-Opti | SDN-based virtual resource orchestration system for optical DCN | Spadaro et al. [110] | UPC, Spain |
| OpenQoS | QoS-aware network traffic controller | Egilmez et al. [32] | Koc University, Turkey |
| B4 | SDN-WAN for geographically distributed data centers | Jain et al. [54] | Google Inc. |
| CNG | Enhanced networking of distributed VMs | Mechtri et al. [79] | Telecom SudParis, France |
| ADON | Network management system for scientific workload | Seetharam et al. [107] | University of Missouri, USA |
| CometCloud | SDN-enabled cloud federation for smart buildings | Petri et al. [96] | Rutgers University, USA |
| SD-IC | Inter-connection for federated SDN-clouds | Risdianto et al. [104] | GIST, Korea |
| Orch-IC | Network resource orchestration for inter-clouds | Kim et al. [64] | ETRI, Korea |
| VIAS | SDN overlay bypass for inter-cloud VPN | Jeong and Figueiredo [58] | University of Florida, USA |
| CL-Orch | Cross-layer network orchestration signaling framework | Cerroni et al. [21] | University of Bologna and Sant'Anna School, Italy |
| SVC | SDN-based vehicular cloud for software updates distribution | Azizian et al. [7] | Univeristy of Sherbrooke, Canada |
| BDT | Optimization model for bulk data transfers | Wu et al. [127] | The University of Hong Kong, Hong Kong |
| SDN-TE | Fault-tolerant cloud resource management | Amarasinghe et al. [5] | University of Ottawa, Canada |

ing the controller. The autonomic QoS control module is designed to configure queue management methods, packet schedulers, and their parameters on an OpenFlow-based

controller, and the flows are managed by finding a matching packet header and changing the forwarding rule accordingly. Also, the authors introduced Packet Context, an information set of the packet characteristics, which is carried in the IP header to mark the packet. Using this mark, packets are prioritized in a queue of each switch port to provide a different QoS based on the Packet Context. The proposal requires customizing IP packet headers which makes it difficult to implement in a general-purpose network, however can be realized for intra-DCN traffics with a higher degree of customization possibility.

Researchers also studied to improve VM acceptance rate and network blocking rate of a cloud data center by provisioning both computing and networking resources jointly with SDN technology. Martini et al. [72] presented a management system for both computing and networking resources in virtualized cloud data centers. The system integrates SDN with VM-host management to orchestrate both resources and improve the service acceptance rate and the data center's utilization level while maintaining the quality of user experience. The system consists of resource selection and composition, coordinated configuration and provisioning, and monitoring and registration functions. The authors proposed a network traffic estimation combining instantaneous and historical values to predict the network load. Also, the system uses a server-driven or a network-driven resource allocation algorithm that attempts to find a server or a network first for a VM request. Both simulation and empirical setups are used for evaluating the proposed system and algorithms.

SDN-based orchestration system for cloud data centers is proposed by Gharbaoui et al. [42]. to control both computing and network resources. The system architecture includes Orchestration Layer on top of SDN controller and VM manager that perform resource selection, provisioning, and monitoring. The authors also propose joint resource selection algorithm for both computing and networking resources. The algorithm selects a physical machine to allocate a VM first and estimates network traffic of the edge switch. If the estimated traffic is overloaded, another physical machine is tried until the available physical machine is found. The proposed system and algorithm are evaluated extensively on simulation environment by measuring VM request rejection ratio and link utilization overloading ratio.

Spadaro et al. [110] presented SDN-based VM and virtual network orchestration system for optical DCN. The orchestrator module is integrated in OpenStack and manages OpenDaylight SDN controller via north-bound API of the controller. An optimization algorithm for provisioning virtual data centers (i.e., VMs and virtual networks) is also proposed to increase the request acceptance ratio by jointly mapping VMs and virtual links. The authors consider joint optimization of cloud and network resources for intra-DCN domain and evaluate the performance by measuring acceptance demands and blocking probability.

**B Inter-DCN Traffic Engineering**

OpenQoS is an OpenFlow-compatible SDN controller designed to support the end-to-end QoS for multimedia traffic between end-users and streaming servers [32]. The network flows for the multimedia traffic are dynamically re-routed to meet the QoS requirement. The incoming traffic flows are grouped into two categories: multimedia flows and the other data flows, where multimedia flows have specific QoS requirements. The multimedia streaming flows are placed on a dedicated route differentiated from the other flows which are transferred through the typical shortest path. The proposal exploits SDN's flow separation and dynamic routing features which differ from the traditional QoS approaches utilizing a resource reservation or priority routing approach. In OpenQoS, multiple flows are bounded to a group by filtering packet header fields. However, due to the processing cost for packet filtering in switches, it should be wisely defined and aggregated. The system was implemented and tested on a small-scale test bed. The fundamental idea of OpenQoS can be extended to the intra-DCN and inter-DCN traffic prioritization in clouds.

B4 is a private WAN connecting geographically distributed data centers implemented by Google [54]. Google adopted SDN concepts in B4 to separate the control plane from the data forwarding plane to enable the network to be programmable. B4 is designed to fulfill the unique characteristics of Google's inter-DCN traffic, such as its massive bandwidth requirements, elastic traffic demand, and full control of the edge servers and network. SDN principles and OpenFlow are implemented to support standard routing protocols and traffic engineering. It can leverage competing demands at traffic congestion,

leverage the network capacity using multipath forwarding, and reallocate bandwidth dynamically when the link failure or application demand changed. In their evaluation, the network utilization of many links reached to about 100% with all links running on average at 70% of utilization. Compared to the traditional network where the average utilization is typically 30-40% due to overprovisioning, B4 resulted in 2 to 3 times efficiency improvements with the same fault tolerance.

Mechtri et al. [79] studied SDN for inter-cloud networking. The authors presented general purpose SDN controller for inter-cloud networking gateway that can be configured and managed by authorized customers. It enables multiple VMs in geographically distributed data centers to interconnect through the SDN-enabled gateway and customize the network for allocation and configuration based on the network requirement of the VMs. The cloud broker in the middle is introduced in order to manage the VMs in distributed data centers and the network infrastructure connecting the VMs. It controls switches in each data center involving the interconnection to enable the network configuration to fulfill the requirement. The proposed algorithm is evaluated in simulation.

ADON is a SDN-based network management system for hybrid campus cloud architecture to run data intensive science application without network bottleneck [107]. As these science applications running in the private cloud of the university sometimes need an excessive resource from public clouds, some workloads should be delivered to the external public cloud. In such case, network bottlenecks can happen while competing for remote resources by multiple applications. The authors deployed SDN controller to prioritize the requests based on the QoS requirements and observed the characteristics of each scientific application. Using per-flow routing and bandwidth allocation feature of OpenFlow, application flows are assigned to a different route with prioritized bandwidth by the application type. The testbed is implemented in practice in two campus locations in addition to Mininet emulation study.

Petri et al. [96] proposed SDN integration for inter-cloud federations to compute smart building sensor data. The authors focused on data-intensive application in need of significant data processing in real-time. The proposed architecture utilized SDN concept over the federated clouds which connects multiple clouds to provide resource elasticity and network programmability. The proposed architecture was applied to smart building

scenario aimed at improving processing times and costs for the building power usage optimization. The prototype has been empirically implemented and deployed on three sites in the UK and the US, and validated with workloads obtained from real sensors. The evaluation showed that in-transit processing using SDN capability reduced overall task execution time for time-constraint workloads.

Risdianto et al. [104] discussed the SDN usage to interconnect federated cloud data centers. The approach combined the usage of both L2 tunnel-based overlay virtual networking and L3 BGP-leveraged routing exchange for connecting multiple data centers. With the usage of SDN, the proposed system can flexibly configure the selected sites and their forwarding path for better redundancy and load balancing. Leveraging the two techniques enabled the flexible selection between L2 and L3 for interconnection of data centers. ONOS SDN controller [91] is used to deploy L3 routing exchange, and OpenDaylight [92] is for L2 management in the implemented prototype.

Kim et al. [64] presented SDN orchestration architecture for integrating SDN-enabled clouds and transport networks. The paper investigates coordination methodology for inter-DCN transmission utilizing SDN functionality to manage both cloud and intercloud networks. Intra-DCN and inter-DCN are managed under separate control domains, named Cloud SDN (C-SDN) and Transport SDN (T-SDN) respectively. On top of them, Virtual Network Coordinator and Transport Network Coordinator oversee orchestrating transport network between data centers, such as creating global virtual network for VMs in different data centers. The proposed system is implemented on their testbed using OpenDaylight, ONOS, and OpenStack.

Jeong and Figueiredo [58] proposed SDN-enabled inter-cloud virtual private networking technique to increase the flexibility of the overlay flows and alleviate the network overhead caused by the tunneling protocols. SDN is integrated into the overlay controllers that creates the virtualized network for inter-DCN and selectively bypass the tunneling packets. The approach is applied to containers running on VMs where the virtual private network is configured among multiple VMs located in different providers and data centers. The prototype is implemented on Ryu controller that controls the OpenVSwitches running on the VMs and evaluated the throughput among the containers running within and across cloud providers.

Cerroni et al. [21] proposed signaling framework architecture following SDN principle for multi-domain data transport network orchestration to provide different QoS requirements of applications. The application-driven configuration of network resources is handled by Application-Oriented Module (AO-M) which implements network resource description parser, SIP proxy server, and network module that controls underlying network service plane (i.e., SDN controller layer). The framework is implemented and validated on empirical testbed in multiple locations with various commercial equipment.

More recently, Azizian et al. [7] proposed using SDN and cloud computing to distribute software updates of vehicles. SDN controller controls both the inter-DCN flows of multiple data centers and the base station networking device where the update information is sent through from the cloud data center to the vehicle.

Bulk data transfers across geographically distributed cloud data centers have been studied by Wu et al. [127]. Bulk volumes of data are transferred between data centers for VM migrations, replication of large contents, and aggregation of MapReduce frames. In this work, the bulk transfer problem has been formulated to an optimal chunk routing problem to maximize the aggregated utility gain before the deadlines [127]. Three dynamic algorithms are proposed that exploit bandwidth allocation, route adjustment, and low complexity heuristics amended from the first algorithm. Time-constrained bulk transmissions are considered in the problem formulation. The proposed system changes the flow table in a core switch connected to a gateway in data centers to alter the bandwidth reservation or routing schedules. The authors evaluated the proposed algorithms on 10 emulated data centers on a real system and measured the job acceptance rate, the total number of accepted jobs on different urgency, and the scheduling delay.

For the purpose of network availability, Amarasinghe et al. [5] proposed a fault-tolerant resource management scheme for clouds using SDN. The authors presented reactive traffic engineering techniques for network failure restoration using network monitoring and dynamic routing features of SDN. The system identifies unexpected link failure and recovers the network by reconfiguring the forwarding switches. The prototype was implemented on POX controller and evaluated on Mininet emulation environment.

Table 2.3: Summary of current research for virtualization in cloud computing with the usage of SDN.

| Project | Description | Author | Organization |
|---|---|---|---|
| FairCloud | Trade-offs sharing networks in cloud computing | Popa et al. [98] | UC Berkeley, USA |
| QVIA-SDN | Virtual infrastructure allocation in SDN-clouds | Souza et al. [30] | Santa Catarina State University, Brazil |
| Opti-VNF | Optimal VNF allocation in a SDN-cloud | Leivadeas et al. [69] | Carleton University, Canada |
| Dyn-NFV | Dynamic NFV deployment with SDN | Callegati et al. [19] | University of Bologna, Italy |
| E2E-SO | End-to-end NFV orchestration for edge and cloud data centers | Bonafiglia et al. [15] | Politecnico di Torino, Italy |

### 2.5.3 Virtualization

SDN plays a key role for network virtualization and network function virtualization in cloud computing. Network virtualization is to segment the physical network resources in cloud data centers into smaller segmentation and lease it to cloud tenants like leasing VM in clouds enabled by host virtualization. NFV utilizes a generic computing resource, such as VM, for providing specific network functions that require high computing power. Instead of purchasing expensive dedicated hardware for CPU-intensive network functions such as firewall, NAT, or load balancing, NFV can provide a cheaper alternative that utilizes a generic hardware with virtualization technology.

While SDN intends a clear separation of network control plane from the forwarding plane to enable the programmability of networks, NFV shifts the paradigm of the network function deployment through advanced virtualization technologies [19]. In the concept of NFV, network functions are provisioned in virtualized resources instead of being tightly coupled in the dedicated hardware, which enable to provision and migrate network functions across the infrastructure elastically and dynamically. Although NFV can be realized without the aid of SDN, the integration of SDN with NFV can accelerate the NFV deployment process by offering a scalable and flexible underlying network architecture [89].

A summary of reviewed works for virtualization objective is presented in Table 2.3, and the detail of each work is explained below.

FairCloud was proposed to virtualize the network in cloud data centers similar to using VM for computing power virtualization [98]. The authors referred the challenges of sharing the network in cloud computing into three aspects: minimum bandwidth guarantee, achieving higher utilization, and network proportionality. Network proportionality was described as the fair share of the network resources among cloud tenants where every tenant has a same proportion of the network. According to the authors, the fundamental trade-offs are necessary between three aspects. For example, if we aim to guarantee minimum bandwidth, the network proportionality cannot be achieved, and vice versa. A similar trade-off is necessary between network proportionality and high utilization. For network proportionality, network bandwidth should be evenly shared by cloud customers if they use the same type of VMs and network plans even if their actual bandwidth usages are different. Thus, strict network proportionality lowers the overall bandwidth utilization of the data center if the disparity of network usage exists between customers. In consideration of these trade-offs, three network sharing policies are proposed and evaluated in simulation.

Souza et al. [30] studied a QoS-aware virtual infrastructure (VMs and their network connections) allocation problem on SDN-clouds as a mixed-integer program. The authors formulate the online virtual infrastructure allocation in SDN-enabled cloud data centers. In order to solve the mixed integer problem, the authors used a relaxed linear program, rounding techniques, and heuristic approaches. They introduced a new VM selection method that considers not only a geographical location of the available zone, but also the end-to-end latency requirement of the VMs. The formula also includes the constraints of server and links capacity, forwarding table size, and latency. The evaluation was performed under simulation environment by measuring five metrics: revenue-cost ratio, data center fragmentation, a runtime for allocation, acceptance ratio, and mean latency of the allocated virtual infrastructure.

In NFV where networking middleboxes (e.g., NAT, firewalls, intrusion detection) are turned into software-based virtual nodes, virtualized network functions (VNFs) are decoupled from dedicated hardware and can be run on any generic machines similar to running VMs on a physical machine. The survey by Mijumbi et al. [82] focused on Network Function Virtualization studies and its relationship with SDN. NFV architectures

and business models were provided in addition to the detailed explanation of relationship with cloud computing and SDN. The main survey covered standardization effort on NFV and the major collaborative projects in both industry and academia. Esposito et al. [34] presented a survey on slice embedding problem on network virtualization. The authors defined slice embedding problem as a subproblem of resource allocation that comprised of three steps: resource discovery, virtual network mapping, and allocation. For each step, the surveyed literature was characterized by constraint type, type of dynamics, and resource allocation method.

As VNFs can be placed at any hardware, VNF allocation problem has received increasing attention with the emergence of NFV technology. Recently, Leivadeas et al. [69] presented an optimal VNF allocation method for a SDN-enabled cloud. The authors considered single or multiple services provided to a single or multiple tenant in the model. NFV orchestrator controls both the SDN controller and the cloud controller to select the optimized place to allocate the VNFs. The formulated problem includes both servers and switches in a cloud to minimize the operational cost of the cloud provider. The optimal solution is presented by mixed integer programming, and four heuristics are proposed. The proposed algorithms are evaluated on simulation to measure the operational cost of the cloud provider, the number of utilized nodes and links, and the utilization. The paper showed the optimal solution of VNF allocation problem and proposed simple and basic heuristics. More delicate heuristics can be studied and proposed to complement their study for further cost saving or energy efficiency.

Callegati et al. [19] presented a proof-of-concept demonstration of dynamic NFV deployment in cloud environment. The system is capable of dynamic SDN control integrated with cloud management for telecommunication operators and service providers implementing NFVs enabling orchestration of intra-DCN and inter-DCN. The authors consider single or multiple VMs hosting various VNFs that dynamically adapt to the network condition. The proof-of-concept is implemented on Ericsson Cloud Lab environment running Ericson Cloud Manager on top of OpenStack.

Bonafiglia et al. [15] also presented open-source framework that manages NFV deployment on edge and cloud data centers along with inter-domain networks that connect data centers. This work considers both intra and inter-DCN architecture as well as orches-

Table 2.4: Summary of current research for security in cloud computing with the usage of SDN.

| Project | Description | Author | Organization |
|---|---|---|---|
| GBSF | Game based attack analysis and countermeasure selection | Chowdhary et al. [22] | Arizona State University, USA |
| Brew | Security framework to check flow rule conflicts in SDN | Pisharody et al. [97] | Arizona State University, USA |

tration of cloud and network resources in a data center. For edge and cloud data centers, OpenStack is used to control the resources, whereas OpenDaylight or ONOS controller manages inter-DC SDN networks. On top of these heterogeneous domain controllers, Overarching Orchestrator (OO) oversees all domains to provide end-to-end service orchestration. OO interacts with edge/cloud and network domains through OpenStack and SDN Domain Orchestrator respectively, that handles a specific domain to communicate with the underlying infrastructure controller. The system also supports network function deployment in any domains i.e., either cloud or SDN domains. The system is validated with OpenStack, ONOS and Mininet setup by experimenting NAT function deployment on either cloud or SDN network.

### 2.5.4 Security

Many studies have been proposed for enhancing security utilizing SDN features to detect and prevent DDoS attacks [129], and some researchers also explained the security vulnerability of SDN controller itself. However, it is difficult to find much literature specifically targeting the cloud computing environment. Although the general approaches using SDN for security can be applied to the cloud computing, we will exclude those general approaches in this survey as our intention is solely on cloud computing. Table 2.4 presents the list of surveyed studies for security.

Yan et al. [129] presented a comprehensive survey on how to prevent DDoS attack using SDN features. The capabilities of SDN can make DDoS detection and reaction easier while the SDN platform itself is vulnerable to security attack noted in its centralized architecture. The authors discussed on both sides; the detailed characteristic of the DDoS attack in cloud computing and defense mechanism using SDN, and DDoS attacks

launching on SDN and the prevention approaches.

A security framework in SDN-cloud environment was proposed by Chowdhary et al. [22] to prevent DDoS attacks using dynamic game theory. The framework is based on reward and punishment in the usage of network bandwidth so that the attackers' bandwidth will be downgraded dynamically for a certain period. The framework implemented on top of ODL SDN controller that functions through the north-bound API of the controller, and evaluated with Mininet.

Recently, Pisharody et al. [97] from the same institution proposed a security policy analysis framework to check the vulnerability of flow rules in SDN-based cloud environments. They describe possible conflicts among flow rules in SDNs forwarding table that can cause information leakage. The framework detects flow rule conflicts in multiple SDN controllers. The detection mechanism is extended from the firewall rule conflict detection methods in traditional networks. The system is implemented on OpenDaylight SDN controller and tested in empirical systems.

### 2.5.5 Summary and Comparison

All studies covered in the survey are summarized and compared in Table 2.5 based on our taxonomy in Figure 2.2. Researchers are actively studying for energy efficiency and performance optimization using SDN in clouds. Scope is varied depending on the proposed method, some focusing on network-only method, while others considering joint computing and networking resource optimization. For studies on energy efficiency, all surveyed papers consider intra-DCN architecture in their model which focuses on power saving within a data center. This reflects the energy trend in recent years that enormous amount of electricity is consumed by data centers which has been increasing rapidly [84]. On the other hand, for performance improvement, many studies consider inter-DCN architecture exploiting SDN on WAN to enhance the QoS and network bandwidth. Although some studies consider the network performance within a data center, there are more opportunities to exploit SDN technology in WAN environment where limited network resources have to be provisioned for cloud tenants. Using SDN's dynamic configuration and network optimization, cloud tenants can acquire more availability and reliability in intra-DCN resulting in better QoS.

Table 2.5: Characteristics of SDN usage in cloud computing.

| Project | Objective | Scope | Arch | | App | | | Rsrc | | Eval | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | intra | inter | web | str | bat | hom | het | sim | emp |
| ElaticTree [48] | Energy | Network | ✓ | | | | | ✓ | | | ✓ |
| CARPO [125] | Energy | Network | ✓ | | ✓ | | | ✓ | | | ✓ |
| DISCO [134] | Energy | Network | ✓ | | ✓ | | | ✓ | | ✓ | ✓ |
| FCTcon [133] | Energy | Network | ✓ | | ✓ | | | ✓ | | ✓ | |
| GETB [29] | Energy | Network | ✓ | | | | | ✓ | | ✓ | ✓ |
| VMPlanner [35] | Energy | Joint | ✓ | | | | | ✓ | | ✓ | |
| VM-Routing [60] | Energy | Joint | ✓ | | | | | ✓ | | ✓ | ✓ |
| PowerNetS [132] | Energy | Joint | ✓ | | ✓ | | | ✓ | | | ✓ |
| S-CORE [27] | Energy | Joint | ✓ | | | | | ✓ | | | ✓ |
| QRVE [47] | Energy | Joint | ✓ | | | | | ✓ | | ✓ | |
| ODM-BD [6] | Energy | Joint | | ✓ | | | | | ✓ | ✓ | |
| OF-SLB [120] | Performance | Network | ✓ | | | | | ✓ | | ✓ | |
| QoSFlow [52] | Performance | Network | ✓ | | | | ✓ | ✓ | | | ✓ |
| AQSDN [126] | Performance | Network | ✓ | | | ✓ | | ✓ | | | ✓ |
| SDN-Orch [72] | Performance | Joint | ✓ | | | | | ✓ | | ✓ | ✓ |
| C-N-Orch [42] | Performance | Joint | ✓ | | | | | ✓ | | ✓ | |
| Orch-Opti [110] | Performance | Joint | ✓ | | | | | | ✓ | | ✓ |
| OpenQoS [32] | Performance | Network | | ✓ | | ✓ | | ✓ | | | ✓ |
| B4 [54] | Performance | Network | | ✓ | | | | | ✓ | | ✓ |
| CNG [79] | Performance | Joint | | ✓ | | | | | ✓ | ✓ | |
| ADON [107] | Performance | Network | | ✓ | | | ✓ | | ✓ | | ✓ |
| CometCloud [96] | Performance | Network | | ✓ | | ✓ | | | ✓ | | ✓ |
| SD-IC [104] | Performance | Network | | ✓ | | | | | ✓ | ✓ | ✓ |
| Orch-IC [64] | Performance | Network | | ✓ | | | | | ✓ | | ✓ |
| VIAS [58] | Performance | Joint | | ✓ | ✓ | | | | ✓ | | ✓ |
| CL-Orch [21] | Performance | Network | | ✓ | | | | | ✓ | | ✓ |
| SVC [7] | Performance | Joint | | ✓ | ✓ | | | | ✓ | ✓ | |
| BDT [127] | Performance | Network | | ✓ | | | | ✓ | | | ✓ |
| SDN-TE [5] | Performance | Network | | ✓ | | | | ✓ | | ✓ | |
| FairCloud [98] | Virtualization | Network | ✓ | | | | | ✓ | | ✓ | ✓ |
| QVIA-SDN [30] | Virtualization | Joint | ✓ | | | | | ✓ | | ✓ | |
| Opti-VNF [69] | Virtualization | Joint | ✓ | | | | | ✓ | | ✓ | |
| Dyn-NFV [19] | Virtualization | Joint | ✓ | ✓ | | | | | ✓ | | ✓ |
| E2E-SO [15] | Virtualization | Joint | ✓ | ✓ | | | | | ✓ | | ✓ |
| GBSF [22] | Security | Network | ✓ | | | | | | ✓ | ✓ | |
| Brew [97] | Security | Network | ✓ | | | | | | ✓ | | ✓ |

**Arch**: Target architecture - intra (Intra-DCN) or inter (Inter-DCN); **App**: Application model - web (web application), str (streaming), or bat (batch processing); **Rsrc**: Resource configuration - hom (homogeneous) or het (heterogeneous); **Eval**: Evaluation method - sim (simulation) or emp (empirical).

For application model, many studies have no target application explicitly considered in the proposal which has no tick symbol in the table. These studies propose generic approaches so that any applications running on the cloud can be beneficial from the proposed method. For energy efficiency, a number of studies consider web application model reflecting the popularity of web applications hosting on clouds. Also, it is accessible to acquire web application workloads because many datasets are publicly available

online, including Wikipedia[1] and Yahoo![2] traces. On resource configuration, most studies for energy efficiency consider homogeneous resources to simplify the research problem because consideration of heterogeneous resource type leads to adding extra parameters to the problem formula. There are a number of studies using both simulation and empirical method for evaluation, while most studies choose either one of them. Details of available evaluation methods are explained in the following section.

## 2.6 Evaluation Methods and Technologies

For accelerating innovation and development of SDN-enabled cloud computing, tools and toolkits are required to build a testbed for testing OpenFlow and SDN systems in a cloud data center. The testbed also has the capability to measure the energy consumption to evaluate proposed algorithms. In this section, simulation tools and empirical methods are explained.

### 2.6.1 Simulation Platforms and Emulators

Simulation platform provides a reproducible and controlled environment for evaluation with ease of configuration and alteration. For cloud computing, many simulation tools have been introduced to evaluate new approaches to managing and controlling the cloud data center and various scenarios. CloudSim [18] is a popular cloud simulator implemented in Java, providing discrete event-based simulation environment capable of simulating cloud data centers, hosts, VMs, and brokers. Various scenarios can be implemented in CloudSim, including VM placement policy, VM migration policy, brokering policy, and other data center management policy. It also supports workload simulation executing in the VMs. With its easy-to-use discrete event-based architecture, additional elements can be added to send and receive simulation events, as well as extending the existing entities to provide extra functionality. However, CloudSim does not support network event in details.

In order to fulfill the lack of network simulation capability of CloudSim, Network-

[1] `www.wikibench.eu`
[2] `webscope.sandbox.yahoo.com/catalog.php?datatype=s`

CloudSim [41] is introduced to simulate applications with network communication tasks. Additional network elements are added in NetworkCloudSim, including network switches and links that receive network events and calculate estimated network transmission time. Although NetworkCloudSim includes extensive network functionality to simulate data center network and message-passing applications in a data center, the support of SDN is not considered in the design and implementation.

GreenCloud [65] is a NS-2 [85] based simulation framework that captures the energy aspect of cloud data centres including computing and communication elements. With the integration of NS-2 which can capture the network pattern accurately on the packet level, GreenCloud can also provide accurate network results. The simulation entities include hosts and switches with power consumption models such as DVFS. Workload models are also predefined and provided in the framework for three types of jobs, e.g., compute-intensive, data-intensive, and balanced. Although GreenCloud provides a comprehensive simulation environment to evaluate network aspects in clouds, evaluating SDN-based applications on GreenCloud is not straightforward because NS-2 and accordingly GreenCloud have not specifically considered SDN features in their design.

For SDN emulation, Mininet [68] is a popular emulation tool to enable testing SDN controllers. Mininet uses virtualization techniques provided by Linux kernel which is capable of emulating hundreds of nodes with arbitrary network topologies. As it uses a real kernel of Linux, it can produce more accurate results including delays and congestion generated at operating system level. Any OpenFlow controller can be tested in Mininet with a capability of executing Linux programs virtually in the emulated host in Mininet. NS-3 [103] is another discrete-event network simulator that provides a simulation for various network protocols on wired and wireless networks. Although Mininet and NS-3 are reliable network emulation and simulation tools, they are not suitable for testing cloud-specific features such as workload schedulers or VM placement policies.

Teixeira et al. [112] proposed a combination framework of Mininet with POX [99], a Python controller for OpenFlow, in order to support simulation of SDN feature in cloud computing environments. Mininet is used to emulate network topologies and data traffic in a data center running OpenFlow controller in POX. With the usage of Mininet and POX, it provides practical results and ready-to-use software in real SDN environment.

The simulation tool, however, is lacking support for cloud-specific features such as defining heterogeneous VM types or executing various application workloads at the simulated host.

### 2.6.2 Empirical Platforms

OpenStackEmu [14] is a testbed framework combining network emulation with OpenStack [93] and SDN. The authors combined SDN controller of OpenStack with another network emulator to enable emulating a large-scale network connected to the OpenStack infrastructure. It also included a data center traffic generator in the framework. Different VM migration, load balancing, and routing strategies can be evaluated on real VMs connected through the emulated network topology.

OpenDaylight (ODL) [92] and ONOS [91] are open source SDN controllers that support the SDN integration for OpenStack via plug-ins. Neutron, the networking module in OpenStack suite, can be configured to use an alternative SDN controller instead of Neutron's own functions. For instance, ODL implements a specific feature called NetVirt (Network Virtualization) for OpenStack integration. By enabling the NetVirt feature in ODL and configuring Neutron to use ODL as a default SDN controller, an OpenStack-enabled private cloud can be used as a testbed for evaluating SDN features in cloud computing.

## 2.7 Summary

This chapter presented a taxonomy of SDN-enabled cloud computing and the survey of the state-of-the-art in building SDN-based cloud computing environments. We categorized necessary aspects of existing works in how to make SDN-enabled cloud computing focusing on networking aspects with SDN technology. The elements include architecture in the usage of SDN, the objective of the research, the application model, hardware configuration and evaluation method to test the proposed approaches. Each element in the taxonomy was explained in detail, and the corresponding papers were presented accordingly. We also described various research projects conducted for energy efficient cloud data centers. There are three main approaches to reduce the energy consumption

in data centers: host optimization, network optimization, and joint optimization. Recently many works are focusing on joint optimization that considers host and network simultaneously to decrease power usage and save operational cost. Afterward, network QoS management methods based on SDN were explained, following various research tools for simulation and energy modeling. Some tools focus on the network with SDN controller while others focus on hosts in the data center.

# Chapter 3

# Modeling and Simulation Environment for Software-Defined Clouds

*To accelerate the innovation pace of SDN-clouds, accessible and easy-to-learn testbeds are required which estimate and measure the performance of network and host capacity provisioning approaches simultaneously within a data center. This is a challenging task and is often costly if accomplished in a physical environment. Thus, a lightweight and scalable simulation environment is necessary to evaluate the network allocation capacity policies while avoiding such a complicated and expensive facility. This chapter introduces CloudSimSDN, a simulation framework for SDN-enabled cloud environments based on CloudSim. We present the overall architecture and features of the framework and provide several use cases. Moreover, we empirically validate the accuracy and effectiveness of CloudSimSDN through a number of simulations of a cloud-based three-tier web application.*

## 3.1 Introduction

TOOLS and toolkits are necessary to foster innovation and development that provide a testbed for experimenting with OpenFlow and Software-Defined Networking systems within a cloud data center. To this end, Mininet [68] is developed to emulate the network topology of OpenFlow switches. Thus, it enables testing different SDN-based traffic management policies in controller. Nevertheless, Mininet concentrates solely on network resources and does not provide any environment to test other cloud

This chapter is derived from: Jungmin Son, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, Xiaohui Ji, Young Yoon, and Rajkumar Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2015)*, Shenzhen, China, May 4-7, 2015.

resource management techniques such as VM placement along with network resources consolidation. To address this shortcoming, we introduce CloudSimSDN that enables the simulation of policies for the joint allocation of compute and network resources.

CloudSimSDN is a new simulation tool built on top of CloudSim [18] that has been briefly discussed in the context of Software-Defined Clouds [16] where resources are dynamically managed and configured in a data center via a centralized controller. In this chapter, we discuss the essence of CloudSimSDN and present a detailed description of its design and implementation. The framework is designed and built in such a way that is capable of evaluating resource management policies applicable to SDN-enabled cloud data centers. It simulates cloud data center, physical machines, switches, network links, and virtual topologies to measure both performance metrics to guarantee QoS and energy consumption to assure environment conservation and cost-reduction.

The CloudSimSDN accuracy is validated and its effectiveness is tested through a number of experiments. The experiments do not intend to provide a novel algorithm for traffic prioritization or host-network resource management but to prove the effectiveness of the simulator in a number of use case scenarios.

The remainder of the chapter is organized as follows. In Section 3.2, we describe the related works and highlight the uniqueness of our simulator. In Section 3.3, we emphasize the requirements of the simulation, then Section 3.4 provides the description of overall framework design and its components in detail. The validation process of the simulator is explained in Section 3.5, followed by an evaluation with use case scenarios for three-tier applications in Section 3.6. Finally, Section 3.7 summarizes the chapter.

## 3.2 Related Work

Recently, many cloud environment simulation tools were proposed to enable reproducible and controlled evaluation of new algorithms for management of cloud resources and applications. CloudSim [18] is a discrete event-based cloud simulator implemented in Java, enabling the simulation of data centers with a number of hosts. VMs can be placed in a host in accordance to VM placement policy. After creating VMs, workloads can be submitted and executed in VMs. Additional elements can be implemented and added to the

simulator to operate with other entities by receiving and sending events. CloudSim does not support network evaluation in details.

NetworkCloudSim [41] simulates applications with communication tasks in CloudSim. In this work, network elements such as switches and links are implemented and added in CloudSim and used to estimate network transmission time. However, they focused on modeling and simulating message-passing applications in a data center that does not include SDN and its dynamically configurable features. We emphasize support of SDN features such as dynamic network configuration and adjustable bandwidth allocation.

The iCanCloud simulator [88] is a solution aiming at the simulation of large scale cloud experiments. It focuses on enabling a cost-performance analysis of applications executing on the cloud. Network simulation is enabled by the INET framework, which enables the simulation of network infrastructures including devices (such as routes and switches) and protocols (such as TCP and UDP) [88]. It does not support the modeling and simulation of SDN controllers and related features.

GreenCloud [65] is a cloud simulator focusing on energy-efficiency research. It extends the NS2 simulator [85], and is able also estimate not only power consumption of computing resources but also from network resources. As for the previous cases, it cannot model and simulate features of SDN.

SPECI [111] is a simulator that focuses on modeling and simulating the data center middleware and failures in the underlying infrastructure. It focuses on analyzing the performance of the middleware under different network conditions. It does not support modeling of cloud applications or SDN features.

RC2Sim [23] is a tool for experimentation and functionality tests of cloud management software via simulation and emulation in a single host. Network is simulated via a module that calculates expected data transfer times given a user-supplied cloud network topology. Unlike the previous simulators, RC2Sim targets analysis of control commands to the cloud infrastructure (such as request for VM creation) rather than analysis of the performance of cloud applications using different policies and cloud environments.

Mininet [68] is a widely used SDN emulation tool to test SDN controllers. It emulates hundreds of nodes with different network topologies in a Linux machine using virtualization techniques provided by the Linux operating system, which presents more

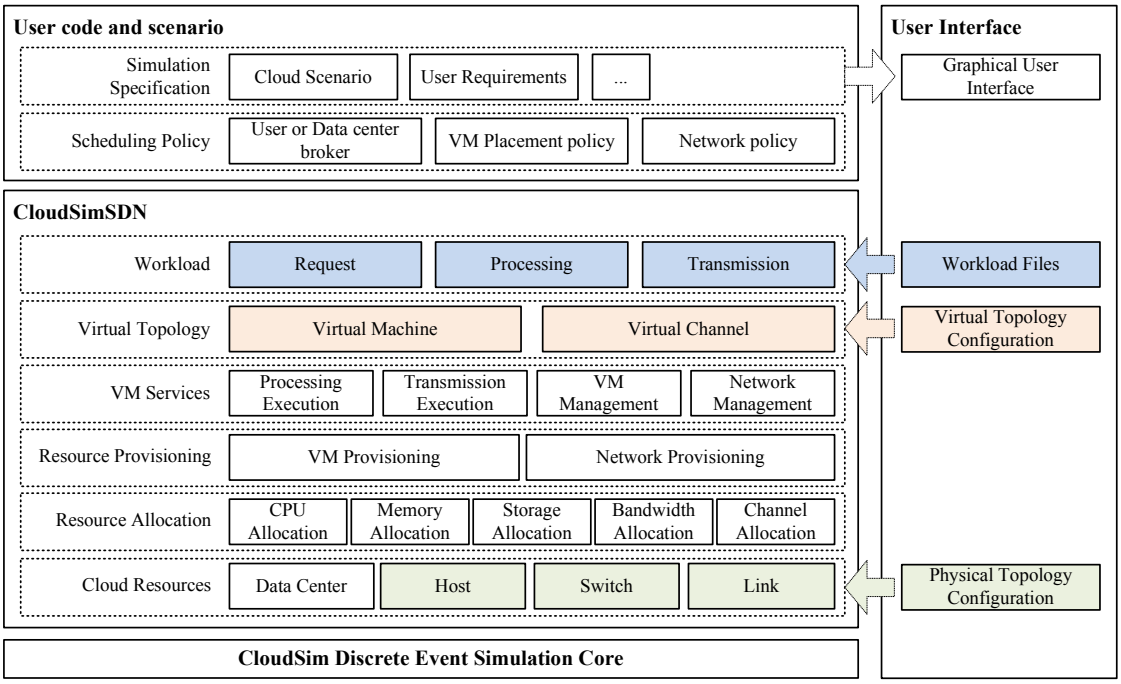



Figure 3.1: CloudSimSDN architecture.

accurate results reflecting delays and congestion at the OS level. An external OpenFlow controller can be attached and tested in Mininet. Similarly, Linux programs can be executed in a virtual node. However, Mininet, similar to NS-3[86], is not capable of testing cloud-specific features such as VM placement policies, workload schedulers, etc.

Teixeira et al. [112] introduced a framework to test SDN cloud-data center controllers using Mininet and POX, a Python controller for OpenFlow SDN standard [99]. They used Mininet to manage network topologies and data traffics and POX to implement the controller of the framework. Thus, it can provide practical results and provide software that is ready-to-use in a real SDN environment. However, it does not allow simulation of cloud-specific features such as different configuration of VM types and application execution.

## 3.3 Software-Defined Cloud Data Center Simulation: Goals and Requirements

Simulation is a valuable tool for initial research of new policies and techniques in distributed systems, as it enables reproducible experimentation in a controlled environment, which is hard to achieve in real cloud infrastructures. Simulation enables quick evaluation of strategies in different scenarios, which can be applied as an initial filter against approaches that underperform compared to existing approaches. As noted earlier, simulation tools exist to enable evaluation of policies for cloud infrastructures, although without support for SDN and all its benefits. Tools exist also that can simulate the effect of SDN controllers on response time of network packets, but without supporting cloud features.

As cloud infrastructures can benefit considerably from SDN and its capabilities, a tool that enables design and test of controller policies and its effect in cloud infrastructures is desirable, and this is the objective of the tool proposed in this chapter. Thus, to achieve our goals of reproducible and controlled experimentation of Software-Defined Cloud data centers, we identified the following requirements:

- Capacity to represent core data center computing elements;
- Capacity to simulate flows and different policies that can be implemented per flow in the infrastructure;
- Support for flexible description of virtual networks that can be deployed on top of the simulated physical network;
- Flexible model of applications, enabling representation of both data transfer and computational needs of the application; and
- Support for reuse of network descriptions (network topologies and data flows), possible via some standard file output format.

The above requirements drove the design and development of our framework, which we detail in the next section.

## 3.4 Framework Design

Our SDN simulator, CloudSimSDN, is built on top of the CloudSim toolkit [18]. It leverages CloudSim's capabilities of modeling computational elements of data centers and CloudSim's simulation engine. To enable modeling and simulation of SDN, we added a number of components to simulate network traffic and SDN controller behaviors.

Figure 3.1 shows the architecture of CloudSimSDN. Users of our framework supply user code and scenarios. Physical and virtual topology configurations can be supplied either as JSON files or as program codes (which are written in Java).

Besides infrastructure description, end-users' requests description, which compose the input workload for the simulation, are supplied in CSV files. Each workload should specify the submission time along with a list of job processing size and traffic data size. In addition to workload and topology configurations, scheduling policies should be provided, such as VM placement algorithm and network policies. Brokers can be programmed to simulate the behavior of end-users or data centers. Regarding these policies, a user can either utilize built-in policies or can develop their own (by extending abstract classes). The aforementioned user input feed to topmost elements of the architecture, namely *Virtual Topology* and *Workload*.

*VM Services* are in charge of managing VMs and network, by calculating application execution and packet transmission time between VMs. The next layer, *Resource Provisioning*, is composed of two modules. VM Provisioning is a core module to provision VMs within data center according to VM placement policy specified by simulator users. Network provisioning is performed according to the network policies in use in the simulation. The next layer, *Resource Allocation*, contains modules that allocate resources specified in the bottommost layer of the architecture, *Cloud Resources*.

Figure 3.2 contains a simplified class diagram for the main classes of our approach. These classes are discussed in the rest of this section.

### 3.4.1 CloudSim core logic

The original CloudSim core logic is used to simulate the basic compute elements that compose the cloud infrastructure. On CloudSim, physical hosts can be defined with spe-

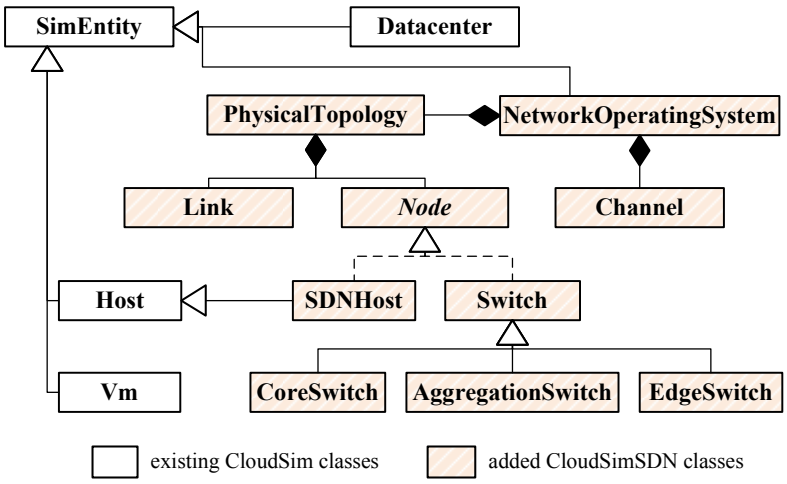



Figure 3.2: CloudSimSDN Class Diagram.

cific configurations and VMs are placed on the host that meets resource requirements such as CPU power, memory, and storage size. CloudSim simulates a range of elements of the cloud architecture, including data center, physical host, VM, VM scheduler, workload scheduler, etc. Although network bandwidth is one parameter of the configuration for physical machines and VMs, bandwidth in CloudSim is used only as a constraint for VM scheduler, and thus the proposed extension is necessary for modeling of SDN features for networking provisioning.

### 3.4.2 Abstracting physical and virtual topology

The physical topology includes hosts, switches, and links. VMs are placed in physical hosts, and network packets are transferred between hosts via switches. Hosts are specified with their computational power, memory, and storage size. Links connect a host to a switch or connect switches with specified bandwidth.

Similarly, virtual topology includes VMs and virtual links. VMs are described with the required computational power, memory, and storage size. Virtual links connect VMs with optional bandwidth amount. If the bandwidth is not specified in a virtual link, then the simulator assumes that communication between them occurs without bandwidth reservation. Case users specify virtual links between two points, bandwidth reservation between the points need to be enforced. Because the path connecting such two points could be chosen among different paths of the physical topology, the problem of map-

ping the virtual topology on the physical one needs to be solved with some objective in mind (such as minimizing the average path length or minimizing the number of network devices involved in the solution). This problem is known as *Virtual Network Embedding* (VNE) problem. There are various researches conducted to solve the VNE problem [39], which can be implemented and tested in our simulator.

### 3.4.3 Network modules

To simulate packet transfer between VMs, we developed a *Switch* class performing SDN-enabled switch function managed by a controller. Forwarding rules are installed by the controller's *Network Operating System*, and can be dynamically changed depending on the network traffic. The virtual links connecting switches and/or VMs are represented with a *Channel* class that defines the physical path capacity of such channels. The class holds a list of physical network elements, such as switches and hosts, along with physical links between those elements.

Once a network packet is generated from a VM and sent to the underlying host, it is forwarded to the destination host using a channel through forwarding routes. By default, a channel is shared by all packets if they are sent from the same source VM to the same destination VM. Since different virtual channels could share the same physical link, each physical link also maintains the list of channels passing through the link itself. If a new channel is created and added to the link, the link updates the shared bandwidth of all channels which passes through the link.

By using the *Network Operating System*, it is also able to create a dedicated channel for a specific traffic flow. As SDN allows the controller to differentiate network flows depending on the type of traffic, our framework also can create a channel for a specific flow with dedicated bandwidth allocation. In this case, an extra channel is created in addition to the default channel, and the packets with specific flow id are forwarded using the new channel.

*Network Operating System* class represents the central controller managing the overall network behavior of the simulation. It monitors all the network's channels and packets, and decides how to reconfigure each network element. User-defined network policies can be developed by extending this class. It also calculates the estimated arrival time

for each packet based on the allocated bandwidth for each channel and the number of packets sharing the same channel. If there is more than one channel sharing a link, each channel size is also included in the bandwidth calculation.

Functions and behaviors supported by SDN are implemented in the *Network Operating System* class. For example, if dynamic bandwidth allocation is necessary to be simulated, policies specifying how to allocate bandwidth to each flow are implemented in this class.

### 3.4.4 Calculating packet transmission time

Simulation of network requires that the transmission time for data transferred between hosts is calculated. Calculation is straightforward if the data is transmitted for one hop that is not shared with other hosts. However, it is more complicated to estimate travel time when the packet needs to be transferred to the host via multiple hops where some are shared by other hosts. In fact, data is fragmented into several packets involved in multiple fragmentation process on each network layer depending on protocols. The fragmentation processes are complicated and varied on different protocols.

Therefore we simplify the transmission process model and the estimation of transmission time. We introduce the class Channel, an end-to-end edge from sender to receiver consisting of multiple links. It is a path for data packets that are going through a series of queues of ports in different switches. The class Link is a physical medium between ports of switches or hosts. The class Transmission refers the transferring data between two hosts which travels through the channel.

In each link, bandwidth is first allocated to the priority channel if SDN is configured to allocate a specific amount of bandwidth to the channel. Afterwards, the remaining bandwidth is equally shared by the channels passing through the link. Thus, allocated bandwidth $BW_{c,l}$ for a channel $c$ in the link $l$ is defined as $BW_{c,l} = \frac{BW_l}{N_l}$, where the link ($l$) has available bandwidth ($BW_l$) shared by the number of channels ($N_l$).

As a channel is composed of multiple links, the transmission speed of the channel basically depends on the least bandwidth among the links. Even if some links have higher bandwidth, there would be a bottleneck when packets pass through a link with lower bandwidth. Thus, for the time period $\Delta t$, when no channel has been added or removed,

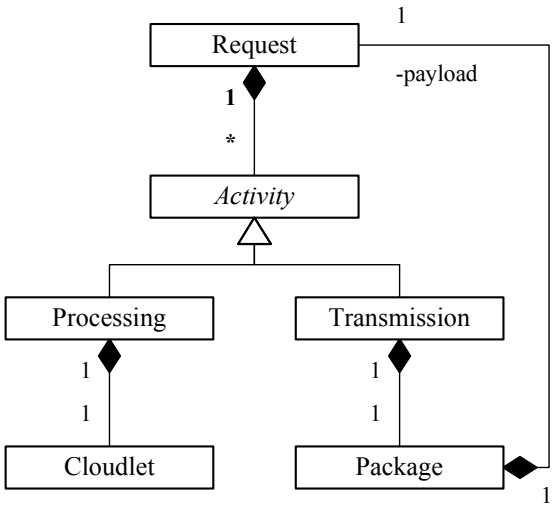



Figure 3.3: User request modeling.

the amount of data $D_c$ transferred from sender to receiver on a channel $c$ can be calculated with Equation 3.1:

$$D_c = \Delta t \times Min(BW_{c,l}) \tag{3.1}$$

When a new channel is added, Network Operating System informs all links where the new channel passes through, and existing channels are updated with a new lower bandwidth value. Channels and links are also updated when a data transmission is finished and the allocated channel is deleted. In this case, the remaining channels will have more bandwidth as there is one less channel using the link. Updated bandwidth values are used to calculate the size of data transferred for the next time period.

### 3.4.5 Abstracting user requests

In practice, a job associated with network transport can be abstracted as a combination of computation processing interleaved with packet transfers. For example, in a web service model, when a request is received at the front-end server, e.g. web server, the front-end server computes the request and creates another request to the mid-tier server, e.g. application server. In the same way, when the mid-tier server receives a request, it processes the request, and sends another request to the back-end server, e.g. database server.

In order to model a request containing both workloads and network transmissions, three classes are implemented: *Request*, *Processing* and *Transmission* (see Figure 3.3), where *Processing* and *Transmission* classes are implemented upon *Activity* interface. *Request* has a list of multiple number of *Activity* objects, which are implemented as either *Processing* or *Transmission*. Processing contains a computation workload (Cloudlet), and Transmission has a network transmission requirement (Package). Several Processing and Transmission objects can compose a Request which should appear in order. If Transmission is appeared after Processing in the Request, the Request is sent to the next VM that is supposed to execute the following Processing. For easy use, list of requests can be input in a CSV format in which has multiple pairs of Processing and Transmission.

In order to estimate network transfer time for each packet, we introduce a Queue in nodes for each flow. For example, if a flow is set up between two hosts, the queue should be set up in the sender's host as well as in all switches that packets go through.

## 3.5 Validation

Validation of CloudSimSDN is a focal point when it comes to the applicability of the simulation. In order to validate CloudSimSDN, we have conducted a series of experiments that compare CloudSimSDN and Mininet with the same workload. As noted earlier, Mininet is a network emulator for the creation of virtual network using the Linux kernel and measurement of data transmission time sent via OS protocol stack. Since it uses the actual Linux kernel for the network emulation, Mininet generates more realistic results, and is widely used to measure SDN-based traffic management policies in controller. Our goal is to first build scenarios with different data sizes and different shortest paths between hosts (including different network elements). Next, for each scenario we analyze how close is the data transfer time between hosts in CloudSimSDN and Mininet which can demonstrate the accuracy of the CloudSimSDN.

### 3.5.1 Mininet setup

Environment for Mininet experiments is set up in Python using Mininet Python API. Network topology in Mininet is created by adding and configuring hosts, switches, and

links, and then each host is scheduled to start data transmission at the same time with other hosts simultaneously. To achieve it, we developed: 1) monitoring agents to measure data transmission time between hosts; 2) Sender agents that generate dummy data with specified size and send it to the receiver agent. Before data transmission begins, the program waits until the given time to make sure all senders start transmission at the same time. When the receiver agent on the other side finishes receiving all data, it sends back an acknowledgement. Once done, the monitoring agent calculates the transmission time. Time clocks of the emulated hosts within Mininet are synchronized as they share the same system clock.

### 3.5.2 Testbed configuration

We created a tree topology of depth 2 with four hosts (see Figure 3.4). The root is a core switch which has two edge switches as child nodes. The leaves are four physical machines connected to the edge switches. Although a tree topology can be effortlessly configured, it can support a number of scenarios, e.g. data sending from VM1 to VM2 passes through only an edge switch, while data from VM2 to VM3 passes through the entire network. In addition, we created one VM in each physical machine in CloudSimSDN because Mininet does not allow VM simulation. Hence, each VM represents a physical machine in Mininet. The configured link speed between core and edge switches, and between edge switches and hosts, are shown in Table 3.1.

In each scenario in Table 3.2, each host is configured to send data with different sizes to the other hosts at the same time, which makes links be shared among multiple connections. As shown in Table 3.2, scenarios differ in the path that the data travels. All transmissions are set up to start at the same time, hence if the data size is not the same,

Table 3.1: Link configuration for the validation experiments.

| Link | Bandwidth |
|---|---|
| Core $\leftrightarrow$ Edge switches | 10 Mbps (1.25 MBytes/sec) |
| Edge switches $\leftrightarrow$ Hosts | 10 Mbps (1.25 MBytes/sec) |

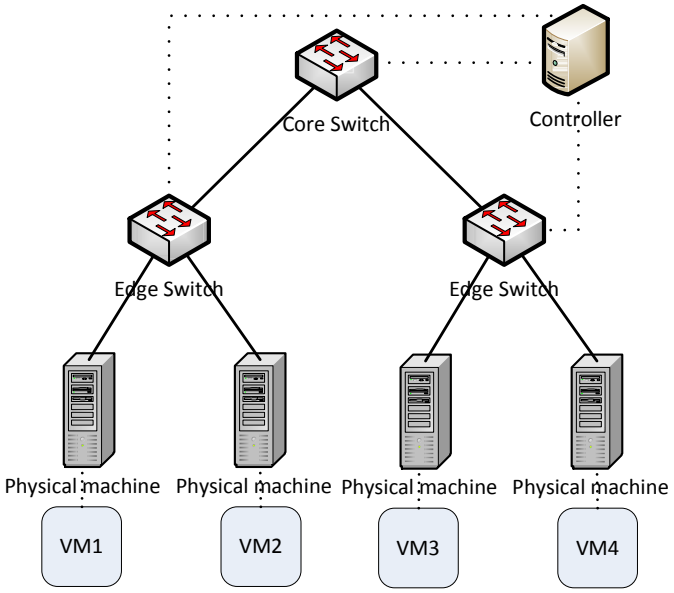



Figure 3.4: Physical topology configuration for the validation test.

Table 3.2: Various scenarios for validation.

| Scenario | Sender | Receiver | Data size |
|---|---|---|---|
| Scenario 1 | VM1 | VM4 | 10 MBytes |
| | VM2 | VM4 | 10 MBytes |
| | VM3 | VM4 | 10 MBytes |
| Scenario 2 | VM1 | VM4 | Varied in uniform distribution (a = 10 MBytes, b = 20 MBytes) |
| | VM2 | VM4 | |
| | VM3 | VM4 | |
| Scenario 3 | VM1 | VM2 | 10 MBytes |
| | VM2 | VM3 | 10 MBytes |
| | VM3 | VM1 | 10 MBytes |
| Scenario 4 | VM1 | VM2 | Varied in uniform distribution (a = 10 MBytes, b = 20 MBytes) |
| | VM2 | VM3 | |
| | VM3 | VM1 | |

some transmissions finish earlier than other transmissions, and then the links, which are shared by terminated transmission, will have more bandwidth for the rest of connections.

### 3.5.3 Validation results

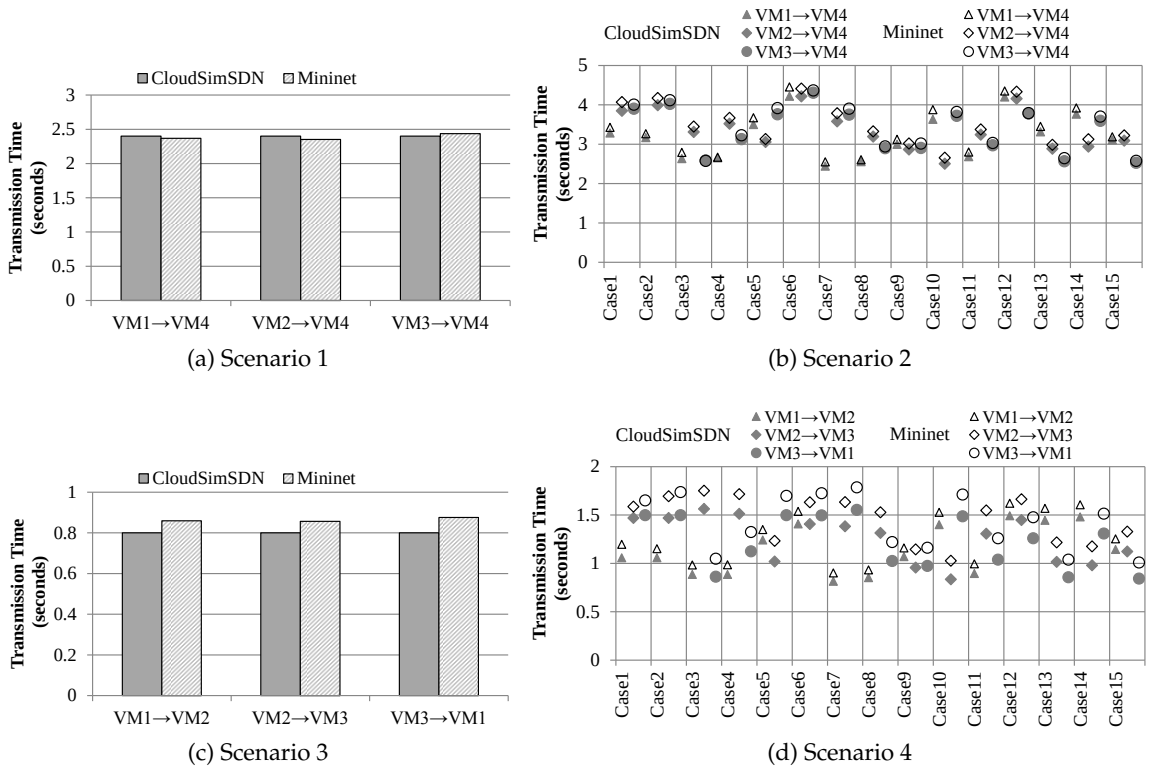
Figure 3.5 shows the measured transmission time in CloudSimSDN and Mininet for the four scenarios described in Table 3.2. In Scenario 1 for the fixed data size, the difference between CloudSimSDN and Mininet is at most 2.5%. When we have variable data size and the same path in Scenario 2, for the majority of cases (for each of which data size is randomly generated based on the distribution described in Table 3.2) the differences is below 4.6%. In Scenario 3, where the path includes more network elements, the difference slightly increased compared to Scenario 1. For the case of Scenario 4, with the same path as Scenario 3 and variable data size, there is only a narrow increase in the difference. This is because factors that affect the network performance, such as TCP window size, OS layer delay, fragmentation latency, etc., are abstracted away from the simulation model. However, this slight loss of accuracy comes with extra advantage of enabling larger-scale evaluation (as our framework does not limit the number of simulated hosts) and also representation of the whole software stack (including the application running on the cloud platform) in the evaluation scenario, as we demonstrate in the next section.

## 3.6 Use Case Evaluation

We focus on two use cases (built in the context of multi-tier web applications) to demonstrate the simulator capabilities and to highlight the advantages of adopting SDN in data centers. The use cases are joint host-network energy efficient resource allocation and traffic prioritization to improve QoS of priority users. Note that such evaluation can only be done with CloudSimSDN (not Mininet) as our simulator supports both compute and network simulations.

### 3.6.1 Joint host-network energy efficient resource allocation

The first use case evaluates the energy savings in SDN-enabled cloud data center via VM consolidation. If resources are consolidated, unused hosts and switches can be switched off by the controller, which maximizes energy efficiency. As turning off and on a host or a switch takes booting time, CloudSimSDN is capable of configuring the minimum




(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

Figure 3.5: Comparison of CloudSimSDN with Mininet for average transmission time for each scenario.

Table 3.3: VM configurations used for joint host-network energy efficient resource allocation use case.

| VM Type | Cores | MIPS | Bandwidth |
|---|---|---|---|
| Web Server | 2 | 2000 | 100 Mbps |
| App Server | 8 | 1500 | 100 Mbps |
| DB Server | 8 | 2400 | 100 Mbps |
| Proxy | 8 | 2000 | 500 Mbps |
| Firewall | 8 | 3000 | 500 Mbps |

duration of idle time to consider it as powered off. In this experiment, we tested different VM placement policies and estimated the energy consumption of hosts and switches. For the test, a data center with 100 hosts and 11 switches is created in CloudSimSDN. Each host is configured with 16 processing cores, and each of them has a capacity of 4000 MIPS (Million Instructions Per Second). The link bandwidths of host-network connections are set to 1 Gbps.

Table 3.4: Energy consumption and the maximum number of simultaneously utilized hosts for different VM placement policies.

| Algorithm | Energy consumption (Wh) | | | Max Nodes | |
|---|---|---|---|---|---|
| | Hosts | Switches | Total | Hosts | Switches |
| Worst Fit (A) | 2,396,380 | 112,492 | 2,508,871 | 100 | 11 |
| Best Fit (B) | 1,848,038 | 92,493 | 1,940,532 | 30 | 4 |
| Energy saving (A-B) | 548,341 (23%) | 19,999 (18%) | 568,340 (23%) | - | - |

500 VM creation requests are generated based on randomly selected VM types specified in Table 3.3, and each request has different start time and lifetime following exponential distribution and Pareto distribution respectively [83]. The network workload is also created for the execution time of VMs to ensure switches are working throughout the VM lifetime. We assumed that VMs are fully utilized and continuously generate network traffic.

We evaluated two commonly used heuristics for VM placement: Best Fit and Worst Fit. The Best Fit policy selects a host whose resources are the most utilized but still available to accommodate allocation requests. In this approach, VMs tend to be consolidated to a smaller number of hosts, and network traffic between hosts can be reduced as more VMs are placed within a host. In contrast, the Worst Fit algorithm selects the freest host which has the maximum available resources, in which VMs can maximize their computational power. To find the most or the least utilized host, we used a normalized unit to combine CPU requirements (total MIPS) and bandwidth constraints, since the two dimensional requirements should be considered at the same time. Power consumption for hosts and switches are modeled based on the works by Pelley at al. [95] and Wang et al. [125], respectively.

We compared the result of the two algorithms in terms of two metrics: energy consumption of hosts and switches, and the maximum number of simultaneously utilized nodes. As shown in Table 3.4, the result depicts that overall 23% of energy consumption of the data center can be saved when the Best Fit is applied for VM placement. Although this result is mainly attributed to the hosts that saved 23% of their power consumption in Best Fit, consolidation of network traffic and deactivation of idle switches also saved

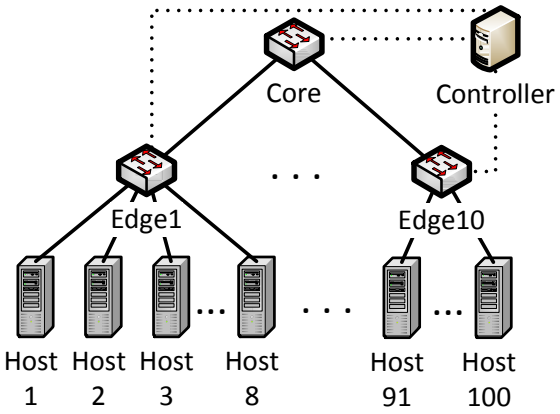



Figure 3.6: Physical topology for traffic prioritization use case evaluation.

Table 3.5: VM configurations for traffic prioritization use case.

| VM Type | Cores | MIPS |
|---|---|---|
| Web Server | 2 | 2000 |
| App Server | 2 | 1500 |
| DB Server | 8 | 2400 |

18% of energy usage from switches.

### 3.6.2 Traffic prioritization

In traditional cloud data center networks, prioritizing network traffic based on the user type was difficult due to complexity and overhead of configuring network elements in such a dynamic environment. However, it is viable in SDN-enabled cloud data center to allocate bandwidth to premium users. This is because the controller is fully aware of network topology and traffic and is capable of controlling queues in SDN switches and dynamically assigning flows to network paths. In this use case, we demonstrate how CloudSimSDN effectively models this capability of SDN.

In simulation, we modeled a data center with depth 3 tree topology and 100 hosts (see Figure 3.6). That is, one core switch connected to 10 edge switches, and each edge switch connected to 10 hosts. Each host is configured with 16 processing cores and 8000 MIPS processing capacity. Physical links are configured with 1Gbps (125 MBytes/sec) bandwidth and 0.5 msec latency.

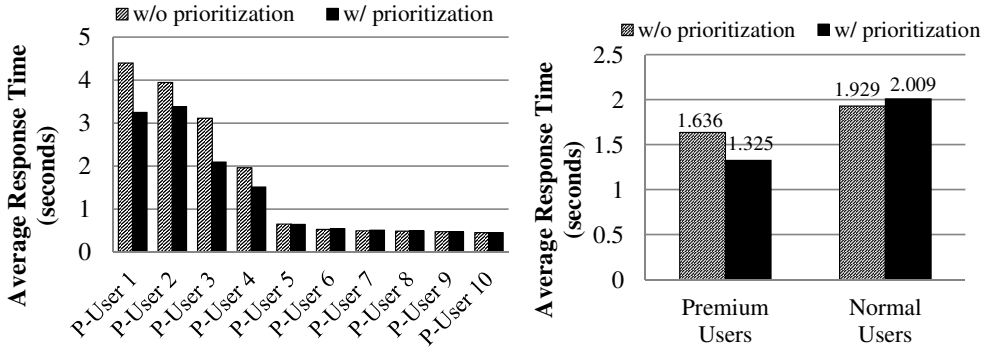
Table 3.6: Characteristics of requests used for traffic prioritization evaluation. Requests are based on the model proposed by Ersoz et al. [33].

| | Distribution | Parameters |
|---|---|---|
| Request inter-arrival times | Log-normal Dist. | $\mu$=1.5627, $\sigma$=1.5458 |
| Packet sizes | Log-normal Dist. | $\mu$=5.6129, $\sigma$=0.1343 (Ch1)<br>$\mu$=4.6455, $\sigma$=0.8013 (Ch2)<br>$\mu$=3.6839, $\sigma$=0.8261 (Ch3)<br>$\mu$=7.0104, $\sigma$=0.8481 (Ch4) |
| Workload sizes | Pareto Dist. | location=12.3486, shape=0.9713 |

There are 50 different customers using the cloud infrastructure in total, 10 users among them are premium users. Each user has an application running on three VMs: Web Server, App Server, and DB Server. Configuration of each VM is shown in Table 3.5. As this experiment aims at evaluating application processing and network performance, the sizes of RAM and storage are not considered as constraints. In the simulation environment, the controller can create separate virtual channels for different data flows. The idea is to allow priority traffic to consume more bandwidth than normal traffic. Thus, virtual channels between VMs are dynamically segmented into two different channels: priority channel and standard channel. By default, without traffic prioritization a standard channel is used to transfer data between VMs regardless user priority where the bandwidth is evenly shared among all packets in the same channel. In contrast, by enabling traffic prioritization feature, a specific amount of bandwidth is exclusively and dynamically allocated for the priority channel, and thus such a bandwidth becomes unavailable for other channels.

For each user, different workloads are generated synthetically based on a typical web service model [33]. Table 3.6 shows the characteristics of synthetic data used for the evaluation. Each request consists of five application processing and four data transmissions in between. At first, processing is done in the Web Server, and then the request is passed to App Server via network transmission. Similarly, App Server has processing and requests data to DB Server. DB Server processes data and return to App Server. Finally, Web Server receives processed data from App Server and responds to the end-user.

When traffic prioritization is enabled, for each priority channel we exclusively pro-




(a) Average response time for premium users.

(b) Average response time for different user types.

Figure 3.7: Effect of traffic prioritization.

vide minimum of 333 Mbps bandwidth. If the bandwidth demand from priority channels exceeds the link capacity, it is equally shared among the priority channels, and no bandwidth is allocated to the standard channels. To make the experiment simple, traffic prioritization is simplified in the simulation without considering sophisticated traffic shaping methods. We measured the response time for each request from submission at the Web Server until return to the end-user.

Figure 3.7a shows detailed performance improvement for each premium user when traffic prioritization is enabled. While response times for Premium User 5 to 10 have remained almost same, Premium User 1 to 4 experienced improvement in application performance. The reason is when the Premium User 1 to 4 are entered, the system experienced higher load and therefore assigning the exclusive bandwidth to the flow of priority requests decreased the response time. This dynamic allocation of bandwidth per-flow is an important feature of SDN to control the QoS in data centers.

In addition, as shown in Figure 3.7b, average application response time for premium users decreased from 1.636 to 1.325 seconds when traffic prioritization is enabled, in which performance is improved by 19.0% on average. On the other hand, overall response time for normal users is slightly increased from 1.929 to 2.009 seconds.

Via this use case, we show how cloud providers can use SDN flow management capability to offer services with various QoS levels. As demonstrated, there is a certain amount of bandwidth reserved for the priority channel that allows priority requests to be served in much shorter time. However, still policies need to be developed to dynami-

cally derive this certain amount of bandwidth based on the changes in the workload, the user QoS requirements (maximum response time), and the priorities of users.

## 3.7 Summary

Given that the infrastructures where SDN operates are large-scale, methods that enable evaluation of SDN configurations before the controller configurations are crucial, and it can be achieved via simulation. To this purpose, we introduced the design and implementation of a simulation framework for Software-Defined Cloud infrastructures in this chapter. The SDN controller is programmable in the simulator, as well as VM management policies and workload scheduling algorithms can be tested in the same framework.

We described our framework design and its components in detail. Validation experiments showed that our simulator is comparable to Mininet in terms of accuracy, and provides the extra features of supporting the arbitrary number of simulated hosts and the simulation of the whole cloud software stack up to the application layer. We also discussed two use cases demonstrating the potential of joint host and network energy-efficient resource allocation and three-tier application, to prioritize data traffic depending on the user type. The open source code of CloudSimSDN is online and free to download[1].

[1]Source code available at: `github.com/cloudslab/cloudsimsdn`

# Chapter 4

# SLA-aware and Energy-Efficient Dynamic Resource Overbooking

*Resource overbooking is one way to reduce the usage of active hosts and networks by placing more requests to the same amount of resources. In this chapter, we propose dynamic overbooking strategy which jointly leverages virtualization capabilities and SDN for VM and traffic consolidation. With the dynamically changing workload, the proposed strategy allocates more precise amount of resources to VMs and traffics. This strategy can increase overbooking in a host and network while still providing enough resources to minimize SLA violations. Our approach calculates resource allocation ratio based on the historical monitoring data from the online analysis of the host and network utilization without any pre-knowledge of workloads. We implemented it in simulation environment in large scale to demonstrate the effectiveness in the context of Wikipedia workloads. Our approach saves energy consumption in the data center while reducing SLA violations.*

## 4.1 Introduction

OVER-provisioning of resources (hosts, links and switches) is one of the major causes of power inefficiency in data centers. As they are provisioned for peak demand, the resources are under-utilized for the most time. For example, the average utilization of servers reported to be between 10-30% for large data centers [10,132], which results in a situation where considerable capacity of data center is idle. Therefore, VM placement, consolidation, and migration techniques have been effectively applied to improve the server power efficiency [37] for the servers which are not energy proportional.

This chapter is derived from: Jungmin Son, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya, "SLA-aware and Energy-Efficient Dynamic Overbooking in SDN-based Cloud Data Centers," *IEEE Transactions on Sustainable Computing (T-SUSC)*, vol.2, no.2, pp.76-89, April-June 1 2017.

Similarly, provisioning of network capacity for peak demand leads to energy waste, which can be reduced through the effective use of Software-Defined Networking (SDN). With SDN, now cloud data centers are capable of managing their network stack through software and consider network as one of the key elements in their consolidation techniques. SDN enables the isolation of network's control and forward planes. This way, routing and other control-related issues are set via a software controller, enabling the forward plane to quickly react and adapt to changes in demand and application requirements [90]. The software controller lies between applications and the infrastructure, and performs tasks that, before SDNs, were performed at individual hardware level (switches, routers). With the emergence of SDN, each individual traffic flow between VMs can be controlled and thus network traffics can be consolidated to less number of links by an overbooking strategy.

While overbooking strategies can save energy, they also increase the chance of SLA violation when either host or network is overloaded. If the consolidated VMs or traffics reach the peak utilization at the same time, insufficient amount of resources would be allocated which will delay the workload processing. The main objective of our approach is to ensure both SLA satisfaction and energy saving without compromising one for the other. We aim to reduce SLA violation rate while increasing energy savings.

In this chapter, we **propose** dynamic overbooking algorithm for joint host and network resource optimization that, in comparison to previous works, has three novelties. Firstly, our approach employs a dynamic overbooking strategy that dynamically adapts to the workload instead of using a fixed percentile. Secondly, it is designed to work without the prior knowledge of the workload. Lastly, we consider initial placement and consolidation strategies together to find the most effective combination for energy saving and SLA satisfaction.

The chapter is organized as follows. We explain the detailed background of SDN and its usage in the context of cloud computing in Section 4.2, and the state-of-the-art approaches for energy savings in cloud data centers in Section 4.3. Section 4.4 formulates the power model and the energy optimization problem. Section 4.5 depicts the overall framework and its three components: Resource Utilization Monitor, Initial Placement Policy, and Migration Policy. In Section 4.6, we explain the strategies for SLA-aware and

energy efficient dynamic overbooking. Section 4.7 presents experiment environment and evaluation results. Finally, Section 4.8 summarizes the chapter.

## 4.2 Background

The current computer networks have reached a point where they are cumbersome to manage and not scaling to requirements of cloud data centres. Utilizing Software Defined Networking (SDN) in cloud data centres is a new way of addressing the shortcomings of current network technologies. In traditional network, distributed routers are core controllers for network management. While the routers can cooperate with each other by communicating network information, the decision is made by a single router with its discrete control logic without consideration of the entire network.

In contrast, SDN has a centralized controller capable of seeing the global view of the entire network. Therefore, traffic consolidation can be performed by the centralized control logic in consideration of energy consumption and SLA satisfaction comprehensively for the entire data center network. Information collected from the entire network is considered for traffic consolidation, and the overall impact on the whole data center is estimated in the control logic. This was not feasible in traditional network as the control logic in the distributed router has limited information and considers only local impact of the control decision.

The centralized control logic in SDN also allows to have both VM and network traffic at the same time for data center optimization. Instead of consolidating VM and network separately, both can be jointly taken into account. Before SDN, network was not considered in VM consolidation process since network cannot be centrally controlled with the global view of the entire data center.

SDN also brings dynamic configuration of the network by separating the control plane from the forward plane. In SDN, the software controller manages overall network through the control plane in each network device, while the forward plane is in charge of forwarding data packets according to forwarding rules set up by the control plane. As the control plane can be dynamically configured by the central controller, network can be quickly adjusted to the current network condition. For example, dynamic bandwidth

allocation for a specific flow is enabled with SDN which can help improve QoS of the network intensive applications.

In short, SDN offers more opportunities for traffic consolidation and energy-saving in data center networks. SDN-enabled cloud data center can make QoS enforcement more convenient in data center networks with responding to the rapidly changing network traffic [16]. Joint optimization of hosts and networks is feasible in SDN-enabled data center.

## 4.3 Related Work

There are several works that have explored energy-efficient cloud resource management with conventional networking [13]. In tihs chapter, we are only focusing at those works in the context of the use of SDN-based virtualized clouds.

ElasticTree [48] is an OpenFlow based network power manager which dynamically change the data center data traffic and adjust network elements for power saving. ElasticTree consolidates network flows to a minimum number of links, and the unused switches are turned off to save more energy consumption. Authors also considered robustness of the network that can handle traffic surges. Although ElasticTree addressed network power savings, VM placement optimization was not considered.

Abts et al. [1] argued that DCN can be energy proportional to the amount of data traffic as like CPU of a computer that consumes less power when it is in low utilization. They proposed link rate adaptation that changes dynamic range depending on the predicted traffic load. They showed that energy proportional networking is feasible by dynamically changing individual link rate. However, they did not address the approach that consolidates traffic and turning off links.

CARPO [125] is a similar approach to ElasticTree and saves data center network power consumption. For traffic consolidation, CARPO adapted correlation analysis between traffic flows so that if the traffic flows are less correlated, those flows can be consolidated into the same network link and more energy savings can be achieved. Additionally, CARPO considered link rate adaptation that alters the link speed of each port depending on the traffic amount. When the traffic is decreasing, link speed slows down

to save more energy.

Recently, researchers started to consider both DCN and host optimization simultaneously. Jiang et al. [59] investigated VM placement and network routing problem jointly to minimize traffic cost in data center. VM placement and routing problem are formulated and solved using on-line algorithm in dynamically changing traffic loads. The proposed algorithm leveraged Markov approximation to find near optimal solution in feasible time.

Jin et al. [60] also considered both host and network factors jointly to optimize energy consumption. They formulated the joint host-network problem as an integer linear program, and then converted the VM placement problem to a routing problem to effectively combine host and network optimization. Finally the best host for placing VM is determined by depth-first search. Prototype is implemented on OpenFlow based system with fat-tree topology and evaluated with massive test cases via both simulation and real implementation.

VMPlanner [35] is presented by Fang et al. that optimizes VM placement and network routing. They addressed the problem with three algorithms: traffic-aware VM grouping, distance-aware VM-group to server-rack mapping, and power-aware inter-VM traffic flow routing [35]. VMPlanner groups VMs with higher mutual traffic and assigns each VM group to the same rack. Then, traffic flow is aggregated to minimize the inter-rack traffic so that the unused switches can be powered off.

PowerNetS [132] is presented by Zheng et al. and finds the optimal VM placement considering both host and network resources using correlation between VMs. Also, detailed power model is introduced which includes power consumptions of chassis, switch, each port as well as the idle and maximum power consumption of a server. PowerNetS measures correlation coefficients between traffic flows and applies them for VM placement and traffic consolidation.

Unlike these techniques, our proposed work uses dynamic overbooking ratio which dynamically changes based on the workload in real-time. This ensures that, with the changes in workload, data center status, and user requirement, our approach can both save energy and maintain SLA satisfaction.

## 4.4 Problem Formulation

The energy efficient host-network resource allocation problem can be formulated as a multi-commodity problem [48]. The objective of the problem is to minimize the power consumption of hosts, switches and links in a data center.

### 4.4.1 Power Models

The following notations are used for the problem formulation.

- $s_i$ : The $i$th switch in the data center;
- $l_i$ : The $i$th link in the data center;
- $h_i$ : The $i$th host in the data center;
- $vm_{j,i}$ : The $j$th virtual machine on host $i$;
- $C(h_i)$ : The capacity of host $i$;
- $C(l_i)$ : The capacity of link $i$;
- $rd(vm_{j,i})$ : The resource demand of the $vm_{j,i}$;
- $f_{j,i}$ : The flow $j$ on link $i$;
- $d(f_{j,i})$ : The data rate of flow $j$ on link $i$;
- $|VM|$ : The total number of VMs in the data center;
- $|H|$ : The total number of hosts in the data center;
- $|L|$ : The total number of links in the data center;
- $\sigma_i$ : The number of VMs placed on host $i$;
- $n_i$ : The number of flows assigned to link $i$;
- $CC(X, Y)$ The Correlation Coefficient between two variables $X, Y$;
- $P(h_i)$ : Power consumption of host $i$;

- $P(s_i)$ : Power consumption of switch $i$;
- $P_{idle}$ : Idle power consumption of host;
- $P_{peak}$ : Peak power consumption of host;
- $u_i$ : CPU utilization percentage of host $i$;
- $P_{static}$ : Power consumption of switch without traffic;
- $P_{port}$ : Power consumption of each port on switch;
- $q_i$ : The number of active ports on switch $i$;

Power consumption of host $i$ is modelled based on the host CPU utilization percentage [95]:

$$P(h_i) = \begin{cases} P_{idle} + (P_{peak} - P_{idle}) \cdot u_i & \text{if } \sigma_i > 0, \\ 0 & \text{if } \sigma_i = 0. \end{cases} \tag{4.1}$$

Idle power consumption is constant factor consumed by hosts no matter how much workload it received. It can be reduced only if the host is turned off. Meanwhile a host consumes more energy when it processes more workload which leads to higher CPU utilization. In this research we adopted linear power model described in [95]. As hosts are homogeneous, power consumption of a host will be same to another if the CPU utilization is same.

Power consumption of switch $i$ is calculated based on the active ports [125]:

$$P(s_i) = \begin{cases} P_{static} + P_{port} \cdot q_i & \text{if } s_i \text{ is on}, \\ 0 & \text{if } s_i \text{ is off}. \end{cases} \tag{4.2}$$

Similar to host's energy consumption, a switch also has static part in its power usage regardless of its network traffic. On top of the static consumption, it consumes more energy when more ports are active with a traffic passing through the switch. We use linear model addressed in [125], where energy consumption of a switch is proportional to the number of active ports in the switch.

### 4.4.2 Problem Formulation

The problem is to optimize the host and network energy consumption jointly in each time period as described below. $|VM|$ VMs are placed in $|H|$ hosts for the time period where $|L|$ links are connected.

$$\textbf{minimize} \sum_{i=1}^{|H|} P(h_i) + \sum_{i=1}^{|S|} P(s_i)$$

and **minimize** *SLA violation*

subject to:

$$\sum_{i=1}^{|H|} \sigma_i = |VM| \tag{4.3}$$

$$\forall h_i \in H, \sum_{j=1}^{\sigma_i} rd(vm_{j,i}) \leq C(h_i) \tag{4.4}$$

$$\forall l_i \in L, \sum_{j=1}^{n_i} d(f_{j,i}) \leq C(l_i) \tag{4.5}$$

$$\forall i, \sum_{j=1}^{|VM|} \theta_{i,j} = 1, \text{where } \theta_{i,j} = \begin{cases} 1 & \text{if } vm_{j,i} \text{ is placed in } h_i \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The objectives are to minimize total energy consumption (energy consumed by hosts and switches) in a data center, and at the same time to minimize the SLA violations. As the two distictive objectives have different measurements, we measure them separately to minimize both objectives at the same time. In this work, SLA violation is quantified to the percentage of the requests exceeding the expected response time. We measured the response time of each request with a baseline algorithm without overbooking, and used it as the expected response time to count the number of requests violating SLA.

The constraints are that resources given to VMs in a host cannot exceed the capacity of the host, and the total data flow rate in a link cannot exceed the capacity of the link, and each VM is placed only once in a host.

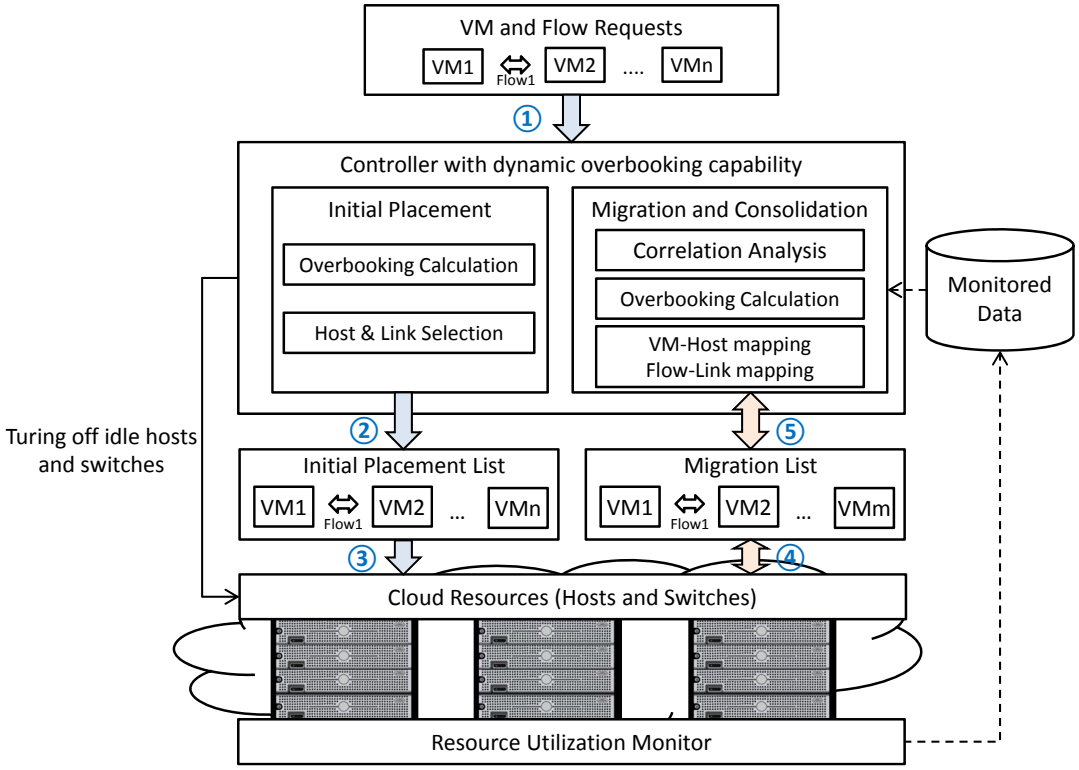



Figure 4.1: Resource Allocation Architecture.

## 4.5 Resource Allocation Framework

The architecture aims to minimize SLA violation and maximize energy saving at the same time without pre-knowledge of the workload. Our proposed architecture is illustrated in Figure 4.1, which benefits from overbooking through SLA-aware VM and flow consolidation. *Overbooking Controller* is in charge of controlling the initial placement and consolidation process. One of the main components is initial placement policy which decides where to place a VM when it is admitted to the data center and creates the initial placement list. Another top-most component is the migration policy that decides a destination host for a VM when the current host is overloaded. It refers a migration list created based on the monitored data and decides which host to migrate to.

For both components, proper overbooking ratio is identified using link information and correlation analysis between VMs' resource utilization. Then a host is discovered which can provide the identified overbooked capacity. Correlation analysis uses monitoring data collected from hosts, VMs and network traffic. This data is also used to build

a migration list which consists of highly utilized VMs in the overloaded hosts to be migrated to another host decided by the Migration policy. The consolidation policy uses current link traffic and host utilization for VM and flow placement and consolidation.

**Resource Utilization Monitor:** This component is in charge of monitoring the utilization levels of resources. Each physical resource can monitor its utilization by itself, such as CPU utilization of each host or bandwidth utilization of the link between switches. The utilization metrics monitored by each physical resource are collected at this component to provide relevant history data to the migration policy. It also collects utilization data of VMs and virtual links to decide the most suitable host for the VM.

**Initial Placement:** When VM and virtual link creation requests arrived at the cloud data center, Initial Placement decides where to create the new VM. At this stage no history or workload information is provided to the data center. Instead, only initial VMs and their connection configurations are available to decide where to place the VM. If VMs are created by the same user at the same time, for example, those VMs have a higher probability to generate traffic between each other. Using this information in addition to the VM configuration, this component decides a host that has sufficient host and network resource to serve the request.

**Migration and Consolidation:** In case of overloading, some VMs in the overloaded host must be migrated to another host in order to minimize SLA violation. Otherwise, VMs in the overloaded host can provide poor performance in computation or network which results in severe customer dissatisfaction. Migration and Consolidation component selects VMs to be migrated in overloaded hosts and decides where to migrate by analyzing historical utilization data of hosts, links and VMs. At first, migration list composing of VMs to be migrated is created based on the monitoring data collected from VMs, hosts and switches. Once the migration list is ready, it analyzes correlation level to other VMs and hosts using historical utilization data. This data is used to pick a migrating host in consideration of overbooking capacity and energy savings.

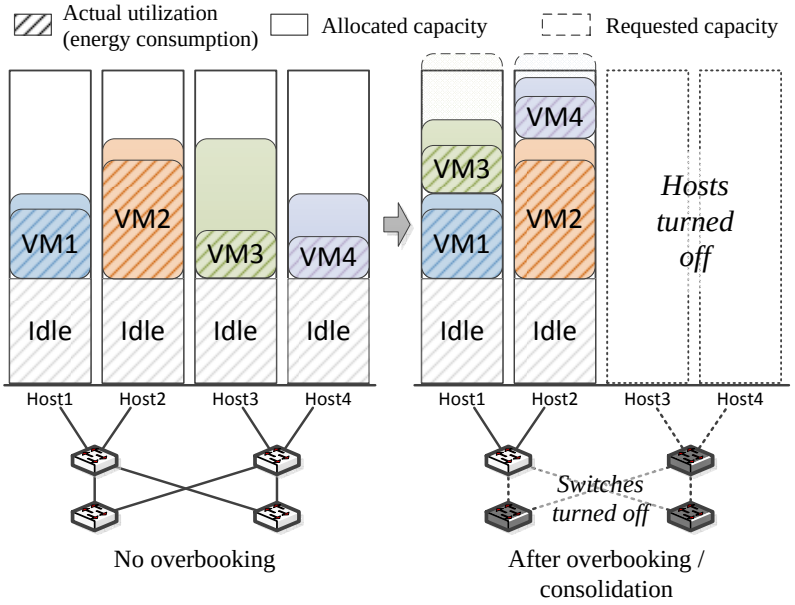



Figure 4.2: Example of consolidation with overbooking

## 4.6 Resource Overbooking Algorithm

In cloud data center, consolidation of VMs and network traffics into a smaller set of physical resources leads to saving power consumption by turning off unused hosts and switches when the physical devices are homogeneous. Although large-scale data centers may consist of heterogeneous devices with different batches of servers and switches, within a single rack, devices are mostly homogeneous. Therefore, we focus on homogeneous configuration to simplify the problem and develop our strategies. A VM is placed to a host by allocating the requested amount of resources, such as CPU cores, memory, disk, and network bandwidth. As in most cases resources are over-provisioned, allocating less resource than requested can help consolidate more VMs and traffics. For clear description, a simple example of overbooking and consolidation in concept is illustrated in Figure 4.2.

Before overbooking and consolidation, VM1-4 are placed in four hosts separately and connected through four switches. If all the four VMs have data traffic, all the switches should be active and consume electrical power along with the four hosts. For VM3 and VM4, we can see that the actual utilization is far lower than the allocated capacity. After overbooking, less amount of resource is allocated to VM3 and VM4 which now can be

consolidated to Host1 and Host2. After migration of VM3 and VM4 to Host1 and Host2 respectively, the hosts without VMs can be turned off, and the connected switches also can be switched off.

We tackled the resource allocation problem described in Section 4.4 through two stages: (1) initial placement stage, and (2) migration and consolidation stage. Initial placement is to find a suitable host for a VM when the VM is created in the cloud data center, whereas VM migration is occurred when the VM needs to be migrated to another host due to a host being either overloaded or underutilized. Note that a different algorithm can be selected for each stage, thus multiple combinations of the two stages are available in the proposed system. The following subsections explain different algorithms for each stage.

For the initial placement the following conditions hold:

- We have no prior knowledge regarding the workload, host utilization, VM utilization, and data rates of flows.
- Although we have no information regarding correlation coefficient between two VMs, it is likely that for the case of web application, workload of connected VMs is correlated.
- If the initial placement strategy places connected VMs on the same host, there is less opportunity for overbooking leading to smaller overbooking ratio. However, this still allows for more saving for network communication cost. Overbooking ratio determines the percentage of original requested resources by users (either in terms of VM or bandwidth).

### 4.6.1 Connectivity-aware Initial VM Placement Algorithms

Initial VM placement algorithms consider connectivity between VMs as explained below.

***ConnCons*: connected VMs to be consolidated in one host.**

At the beginning of the placement, the algorithm (pseudo code is shown in Algorithm 1) groups VMs based on their connectivity. Then, it sorts the groups based on their resource

**Algorithm 1** *ConnCons* initial placement

```
1: Data: IRAR: User-defined initial resource allocation ratio constant.
2: Data: VM: List of VMs to be placed.
3: Data: F: List of network flows between VMs.
4: Data: H: List of hosts where VMs will be placed.
5: VMG ← list of VM groups in VM based on connections in F;
6: sort VMG in descending order of the sum of bandwidth requirements in each group;
7: for each VM group vmg in VMG do
8:     for each vm in vmg do
9:         H_conn ← List of hosts where other VMs in vmg are placed;
10:        if H_conn is empty or length(vmg) = 1 then
11:            Place vm in the most-full host in H;
12:        else
13:            sort H_conn in ascending order of free resources;
14:            done ← false;
15:            for each h in H_conn do
16:                C_h ← free resource in host h;
17:                rd ← adjusted resource demand of vm calculated with IRAR;
18:                if rd < C_h then
19:                    Place vm in h;
20:                    C_h ← C_h − rd;
21:                    done ← true;
22:                end if
23:            end for
24:            if done=false then
25:                Place vm in the host in H with average shortest distance from vmg;
26:            end if
27:        end if
28:    end for
29: end for
```

requirements (sum of VM's resource demands) in decreasing order. Once the list is ready, it picks a VM ($vm_{k,i}$) from the top of the list. If it is not connected to other VMs or if the connected VMs have not been placed yet, we place it using most-full bin-packing algorithm. Otherwise, it consolidates the VM to the same server ($h_i$) where the connected VMs are placed if the following constraint can be met:

$$\sum_{j=1}^{\sigma_i}(rd(vm_{j,i})) + IRAR \times rd(vm_{k,i}) < C(h_i) \tag{4.7}$$

where Initial Resource Allocation Ratio ($IRAR$) indicates the proportion of the actually allocating resource to the requested resource at initial stage. Note that Resource Allo-

cation Ratio ($RAR$) can be regarded as the reverse of overbooking ratio, e.g. 70% $RAR$ means that the host will allocate 70% of the requested resource to the VM. Thus, with lower $RAR$ value hosts allocate less resource to a VM resulting in placing more VMs in a host and higher chance of SLA violation. $IRAR$ is a predefined constant in the system configuration and can be changed manually.

This method is basically derived from CARPO [125] system that correlated VMs are consolidated into the same or nearby host. In addition to the principle of CARPO, we propose dynamic overbooking strategy that changes overbooking ratio dynamically adapting to the workload.

If there exist multiple connected VMs that have already been placed on different hosts, the most-full host will be selected. Otherwise, it searches for a host with the shortest distances from the connected VM hosts. If multiple VMs have already been placed on different hosts, a host with average shortest distance will be selected. Next, if there are multiple choices (hosts with the same distance and network path with same number of hops), the algorithm uses the most-full first bin-packing algorithm for both hosts and candidate links. In addition, the selected candidates have to meet constraints in Equation (4.7) and constraint in Equation (4.8) for each selected Links of $l_i$:

$$\sum_{j=1}^{n_i}(d(f_{j,i})) + IRAR \times d(f_{k,i}) < C(l_i) \tag{4.8}$$

If the constraint cannot be met, the algorithm selects the next host candidate until all the VMs are placed. Note that for this algorithm the $IRAR$ are likely to be set to a higher value as utilizations of VMs in a server are likely to be correlated.

***ConnDist*: connected VMs to be distributed into different hosts.**
At the beginning of the placement, the algorithm (pseudo code is shown in Algorithm 2) sorts the VMs based on their resource requirements in decreasing order. Once the list is ready, it picks a VM ($vm_{k,i}$) from the top of the list. Then, if it is not connected to other VMs or if the connected VMs have not been placed yet, it will be placed using most-full bin-packing algorithm. Otherwise, it ignores servers where the connected VMs are placed, and searches for a server with the average shortest distances from the hosts of

**Algorithm 2** *ConnDist* initial placement

1: **Data**: $VM$: List of VMs to be placed.
2: **Data**: $H$: List of hosts where VMs will be placed.
3: sort $VM$ in descending order of the resource requirements;
4: **for each** $vm$ in $VM$ **do**
5: $VM_{conn} \leftarrow$ List of connected VMs of $vm$;
6: $H_{conn} \leftarrow$ List of hosts where other VMs in $VM_{conn}$ are placed;
7: **if** $H_{conn}$ is empty **then**
8: Place $vm$ in the most-full host in $H$;
9: **else**
10: $H_{noconn} \leftarrow H - H_{conn}$;
11: Place $vm$ in the most-full host in $H_{noconn}$ with the same constraint in Algorithm 1;
12: **end if**
13: **end for**

connected VMs. Next, if there are multiple choices, the algorithm uses the most-full bin-packing algorithm for both host and link candidates which meets constraint in Equations (4.7) and (4.8).

If the constraint cannot be met the algorithm selects the next host candidate until all the VMs are placed. Note that for this algorithm the $IRAR$ is likely to be set to lower values as we consolidate connected VMs to different servers. Therefore, utilization of VMs placed on a same server is less likely to be correlated.

### 4.6.2 VM Migration Algorithms with Dynamic Overbooking

Based on the collected information, this algorithm:

- selects overloaded hosts with the utilization over the threshold (e.g., 0.7) and moves the most utilized VM to the migration list,
- selects the underutilized host with the utilization under the threshold (e.g., 0.1) and move their VMs to the migration list,
- selects the overloaded links with the average bandwidth usage over the threshold (e.g., 70% of the link capacity) and move the VMs in the link with highest data rates into the migration list.

It is worth mentioning that the migration of flows happens at the final stage. The reason

is that over-utilized VM migration can resolve the link congestion.

After that, the algorithm sorts the VMs in the migration list based on their resource requirements in descending order. Then, it picks a VM ($vm_{k,i}$) from the top of the list. For the selected VM, VM migration algorithm selects a candidate host in which the VM can be placed. In the host selection, dynamic overbooking algorithm is applied as a constraint to make sure the VM and its network link fulfil enough capacity to process the workload, and at the same time limit to minimal amount for consolidation. For the host capacity, Equations (4.9) and (4.10) is applied.

$$\sum_{j=1}^{\sigma_i}(rd(vm_{j,i})) + DRAR_h \times rd(vm_{k,i}) < C(h_i) \tag{4.9}$$

$$DRAR_h = Min_{DRAR} + \frac{Max_{DRAR} - Min_{DRAR}}{Max_{DRAR}} \times \frac{1}{\sigma_i}\sum_{j=1}^{\sigma_i} CC(vm_{j,i}, vm_{k,i}) \tag{4.10}$$

In addition to host capacity constraint, network constraint is also applied as constraint to select the target host and link: Equations (4.11) and (4.12).

$$\sum_{j=1}^{n_i}(d(f_{j,i})) + DRAR_l \times d(f_{k,i}) < C(l_i) \tag{4.11}$$

$$DRAR_l = Min_{DRAR} + \frac{Max_{DRAR} - Min_{DRAR}}{Max_{DRAR}} \times \frac{1}{n_i}\sum_{j=1}^{n_i} CC(d(f_{j,i}), d(f_{k,i})) \tag{4.12}$$

As you can see from Equations 4.10 and 4.12, in this algorithm we dynamically calculate the Dynamic Resource Allocation Ratio ($DRAR$) based on the correlation of VMs in the host. As explained in the previous section, Resource Allocation Ratio ($RAR$) is a term that defines the percentage of actually allocated resource compared to the requested resource. It can be regarded as a reserve of overbooking ratio. $DRAR$ is applied not only

as constraints of the VM admission, but also actual resource allocation for the migrating VM. This will allow us to save more energy by resource overbooking and honoring more SLA by dynamically changing overbooking ratio.

$DRAR$ is applied as constraints to decide the admission of the migration and to allocate resources to VMs. For example, for 100% $DRAR$, the host allocates 100% of the requested resource to the VM. If $DRAR$ decreased to 70% in another host, it gives only 70% of the requested resource thus the host can consolidate more VMs.

To determine $DRAR$, we use correlation coefficient derived from historical utilization data, VM's average utilization of the previous time frame, and preliminarily defined variables to decide the portion of each parameter. Correlation between VMs is calculated with Pearson Correlation Coefficient, which ranges between -1 and 1. Lower the coefficient, lower the correlation. If the coefficient is closer to 1, it indicates the VMs are more correlated. As it ranges from -1 to 1, we use Equation (4.13) to normalize the range between 0 and 1.

$$CC(X,Y) = \left(\frac{\mathbf{Cov}(X,Y)}{\sqrt{\mathbf{Var}(X)\mathbf{Var}(Y)}} + 1\right)/2 \tag{4.13}$$

Additionally, minimum and maximum Dynamic Resource Allocation Ratio ($Min_{DRAR}$ and $Max_{DRAR}$) are defined to limit the range of the $DRAR$. $Min_{DRAR}$ is differentiated with the average utilization of the VM for the previous time window. Thus, $DRAR$ is affected by not only the correlation, but also the actual utilization of the VM. In order to decide the share of each parameter, $\alpha$ and $\beta$ are defined in Equation (4.14).

$$Min_{DRAR} = \alpha \times U(vm_{k,i}) + \beta \tag{4.14}$$

where $\alpha$ specifies the portion of the utilization of the VM and $\beta$ specifies the guaranteed proportion of the requested resource to be allocated to the VM. $\alpha$ and $\beta$ are defined in the experiment configuration along with the $IRAR$. On the other hand, $Max_{OR}$ is configured to 1.0 in the implementation to make it possible to assign 100% of the requested resources.

Algorithm 3 shows overall migration procedure for each VM to a candidate host with the constraints using $DRAR$ calculation. The complexity of the alogorithm is $O(|VM_h|)$ which can run for at most the number of hosts in the data center if all hosts are over-

**Algorithm 3** VM migration with dynamic overbooking

**Data**: $\alpha$: User-defined constant for historical utilization fraction in $Min_{OR}$.
**Data**: $\beta$: User-defined constant for minimum Resource Allocation Ratio.
**Data**: $t_{start}, t_{end}$: Start and end time of the previous time window.
**Data**: $vm_{mig}$: A VM to be migrated.
**Data**: $h$: A candidate host.
**function** MIGRATE($vm_{mig}, h$)
    $VM_h \leftarrow$ all VMs in the host $h$;
    $u_{mig} \leftarrow$ utilization matrix of $vm_{mig}$ in $(t_{start}, t_{end})$;
    $S_{corr} \leftarrow 0$;
    **for each** $vm_i$ in $VM_h$ **do**
        $u_i \leftarrow$ utilization matrix of $vm_i$ in $(t_{start}, t_{end})$;
        $S_{corr} \leftarrow S_{corr} + CC(u_{mig}, u_i)$;
    **end for**
    $DRAR_h \leftarrow$ calculate with Equation (4.10) ;
    $C_h \leftarrow$ free resource in host $h$;
    $rd \leftarrow$ requested resource of VM $vm_{mig}$;
    $rd_{DRAR} \leftarrow DRAR_h \times rd$;
    migrated $\leftarrow$ false;
    **if** $rd_{DRAR} < C_h$ **then**
        $DRAR_l \leftarrow$ calculate with Equation (4.12) ;
        $C_l \leftarrow$ free resource of the link of host $h$;
        $d \leftarrow$ requested resource of the flow of $vm_{mig}$;
        $d_{DRAR} \leftarrow DRAR_l \times d$;
        **if** $d_{DRAR} < C_l$ **then**
            Migrate $vm_{mig}$ to $h$;
            $C_h \leftarrow C_h - rd_{DRAR}$;
            $C_l \leftarrow C_l - d_{DRAR}$;
            migrated $\leftarrow$ true;
        **end if**
    **end if**
    **return** migrated
**end function**

loaded. Thus, the overall complexity of the migration process is $O(|VM| \cdot |H|)$ for entire data center which is reasonable for online decision.

With the base of dynamic overbooking constraints explained above, three consolidation algorithms are proposed with different host selection methods: most correlated, least correlated, and most underutilized host to be chosen. Consolidating a VM to the most correlated host can make a higher chance to reduce network traffic since the VMs in the host have more correlation on network traffic to the migrating VM. However, it will increase the chance of overloading of the host, as the correlated VMs will have higher pos-

sibility to reach the peak at the same time. For this reason, we also propose an approach that consolidates to the least correlated host. If the workload has less network traffic but more computational processing, migration to the least correlated host will reduce the chance of the host overloading. For comparison, migration to the most underutilized host without consideration of correlation is also tested. Note that the dynamic overbooking constraint is applied to every algorithm but with different host selection preferences. These three algorithms are explained below.

***MostCorr*: VM to be migrated to the host holding the linked VMs.**
If the VM to be migrated is not connected to other VMs or if the connected VMs in the list have not been placed yet, it will be placed using most-full bin-packing algorithm. Otherwise, it consolidates the VM to the same server where the connected VMs are placed and if the aforementioned constraints can be met. If not, it searches for a host with the shortest distances from the connected VMs' hosts. If there are multiple choices, it uses bin-packing (most-full first) both for candidate links and hosts to choose the destination if the aforementioned constraints can be met. Details are described in Algorithm 4.

***LeastCorr*: VM to be migrated to the least correlated host.**
The algorithm is similar to the previous consolidation strategy, but selects the host with the lowest average correlation coefficient between the migrating VM and the VMs in the host. It calculates correlation coefficient for each host with at least one VM and sorts the list in ascending order. Then, the least correlated host is selected with the $DRAR$ constraints (3) applied. If no host is found among non-empty hosts, it selects the first one from empty hosts. With this algorithm, the connected VMs are likely to be placed into a separate host which will incur more network communication, whereas the chance of host overloading will be reduced.

***UnderUtilized*: VM to be migrated to the underutilized host.**
In this algorithm the migrating VM is placed to the least utilized host. Firstly underutilized hosts list is prepared among non-empty hosts, and the first VM in the migration list with the highest utilization is placed to the first host in the list which is the most

**Algorithm 4** *MostCorr* migration algorithm with dynamic overbooking

```
1: Data: VM: Selected migration VM list.
2: Data: H: List of hosts.
3: sort VM in descending order of requested CPU resources;
4: for each vm in VM do
5:     VM_conn ← List of connected VMs of vm;
6:     H_conn ← List of hosts where other VMs in VM_conn are placed;
7:     if H_conn is empty then
8:         Migrate vm to the most-full host in H with the constraints in Algorithm 3;
9:     else
10:        sort H_conn in ascending order of free resources;
11:        migrated ← false;
12:        for each h in H_conn do
13:            migrated ← MIGRATE(vm, h);
14:            if migrated=true then
15:                break
16:            end if
17:        end for
18:        if migrated=false then
19:            Migrate vm to the most-full host in H with the constraints in Algorithm 3;
20:        end if
21:    end if
22: end for
```

underutilized. Same as the previous algorithms, it also dynamically calculates $DRAR$ based on the number of VMs in the host. $DRAR$ calculated from correlation is applied as constraints to check whether the host can accept the migration or not. In short, the most utilized VM in the migration list is to be placed in the least utilized host.

### 4.6.3 Baseline Algorithms

We compare our approach with the baseline algorithms explained below.

***NoOver*: No overbooking without any migration.**

The algorithm is a non-overbooking that allocates 100% of the requested resource to all VMs and network. It uses Most Full First bin-packing algorithm for VM placement, which allocates the VM to the most full host that has enough resource to serve the VM. When selecting a host, it does not consider any connectivity or correlation between VMs. Therefore, VMs can be allocated in any host regardless of their connectivity, e.g. the con-

nected VMs can be randomly placed in the same host or in different hosts depending on the available resource of each host at the moment. Migration is not implemented in this algorithm as a host will not exceed its capacity. This is used as a baseline at evaluation to calculate SLA violation rate and energy saving percentage.

***ConnNone*: Connectivity agnostic overbooking.**
For initial placement, this algorithm overbooks resources without consideration of the connectivity between VMs. This algorithm allocates less amount of resources to VMs and uses the Most Full First algorithm for VM allocation regardless of VM links. For example, ConnNone 70% is to allocate only 70% of the requested resource to the VM and place it to the most full host which can serve the 70% of the requested resource. Similarly, ConnNone 100% is to allocate 100% of the requested resource which is in fact same as NoOver algorithm.

***StaticMigration*: VM to be migrated to the most correlated host without dynamic overbooking.**
Similar to MostCorr, this algorithm also selects the correlated host first for a migrating VM with the same constraints described in Section 4.6 except for $DRAR$. Instead of using dynamically calculated $DRAR$, this algorithm uses a static overbooking ratio for the constraints of the host selection and the resource allocation. As a result, the new host will allocate the same amount of the resource to the migrating VM. This algorithm is implemented in order to refer to PowerNetS [132].

## 4.7 Performance Evaluation

The proposed algorithms are evaluated in simulation environment. We implemented the proposed methods in addition to other algorithms including non-overbooking and PowerNetS [132], and measured a response time of the workload and total energy consumption in the data center. SLA violation is checked through the response time of the workload. We measured the response time of each workload with a baseline algorithm without overbooking, and use them to compare with the response time of the proposed

algorithms. If the response time of a workload with a proposed algorithm is longer than the baseline one, the workload is as a SLA violation. Energy consumption is also compared with the no overbooking baseline algorithm.

### 4.7.1 Testbed configuration

In order to evaluate our approach, we implement the algorithms in CloudSimSDN proposed in Chapter 3. CloudSimSDN is a CloudSim [18] based simulation tool which supports various SDN features such as dynamic network configuration and programmable controller. We add monitoring components to the simulator to gather utilization information of VMs, hosts, and network traffics to be used at dynamic overbooking methods described in section 4.6.

The cloud data center simulated for our experiment consists of 128 hosts, each with 8 CPU cores, connected with Fat-Tree topology [4].

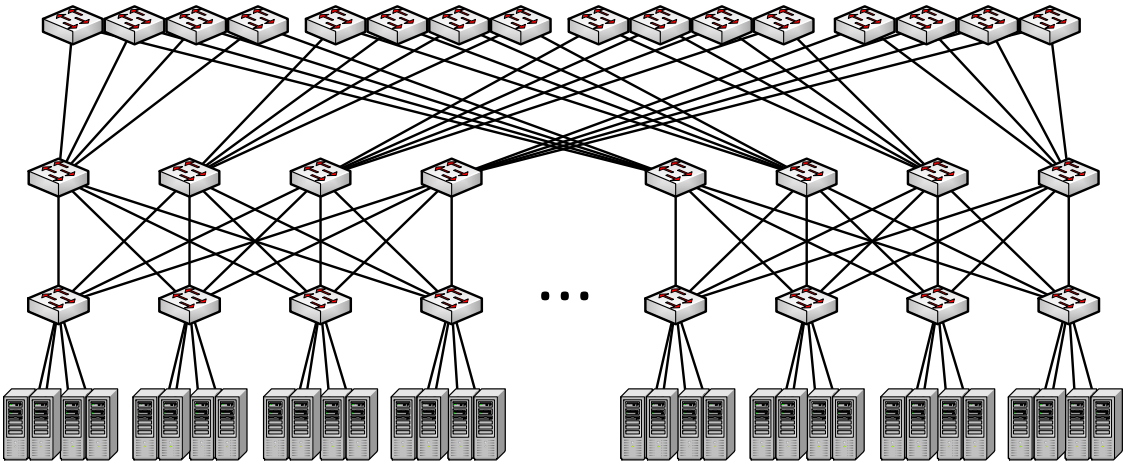


Figure 4.3: Network topology used in simulation

Figure 4.3 shows 8-pod Fat-Tree topology which we adopt in the experiment. Each pod consists of 16 hosts, 4 edge switches and 4 aggregation switches. On top of all pods, 16 core switches enables communication between pods by connecting aggregation switches in each pod. Other resource requirements such as memory and storage are not considered in the experiments to eliminate the complexity affecting the results.

Unless noted, initial placement overbooking algorithms (ConnNone, ConnCons, and ConnDist) have Initial Resource Allocation Ratio ($IRAR$) value set to 70% in all experi-

ments. For dynamic overbooking migration algorithms, we set $\alpha$ value being 12% and $\beta$ value being 40% (Equation (4.14)). Thus, Dynamic Resource Allocation Ratio ($DRAR$) is guaranteed to be at least 40% and dynamically changing up to 100% which is affected by the previous utilization average for 12% and correlation analysis for the rest 48%. $MAX_{OR}$ is set 100% to make sure VMs can receive the full resource when necessary.

For experiments with migration policy, the monitoring interval is set to 3 minutes to collect the utilization of VMs, hosts, flows, and links. Dynamic time window to run migration policy is configured to 30 minutes, thus migration is attempted every 30 minutes with the utilization matrix of 10 monitored points. These parameters are selected in consideration of the workload and the migration costs, but can be changed arbitarily for different workloads.

### 4.7.2 Workload

In a typical data center traffic varies hourly, daily, weekly, and monthly. Traffic characterization of a data center would allow us to discover patterns of changes that can be exploited for more efficient resource provisioning. To achieve our objectives for this activity, we have focused on Wikipedia data center analysis. We decided to investigate a Wikipedia workload by looking into *Page view statistics for Wikimedia projects* which are freely and publicly available. For each day and for all of Wikipedia's projects, the traces consist of hourly dumps of page view counts. To gain insight of the traffic for each project for the whole day (Sep 1, 2014 chosen for this case) we need to analyze traces which consist of 24 compressed files each containing 160 million lines (around 8 GB in size). We have utilized Map-Reduce to calculate number request per hour for each project more effectively and faster.

As Figure 4.4 shows, we can observe that workload varies per hour and that not all workload are reaching their peaks at the same time. We can assume that each project is hosted by a set of virtual machines; there exist VMs that their utilizations are not correlated. This observation can help us to propose more efficient resource provisioning cloud data center by placing non-correlated VMs in one host and thus accomplishing effective overbooking. Please note that we choose the Wikipedia workload because it reflects the real data-center workload as well as the obvious correlation between languages in the

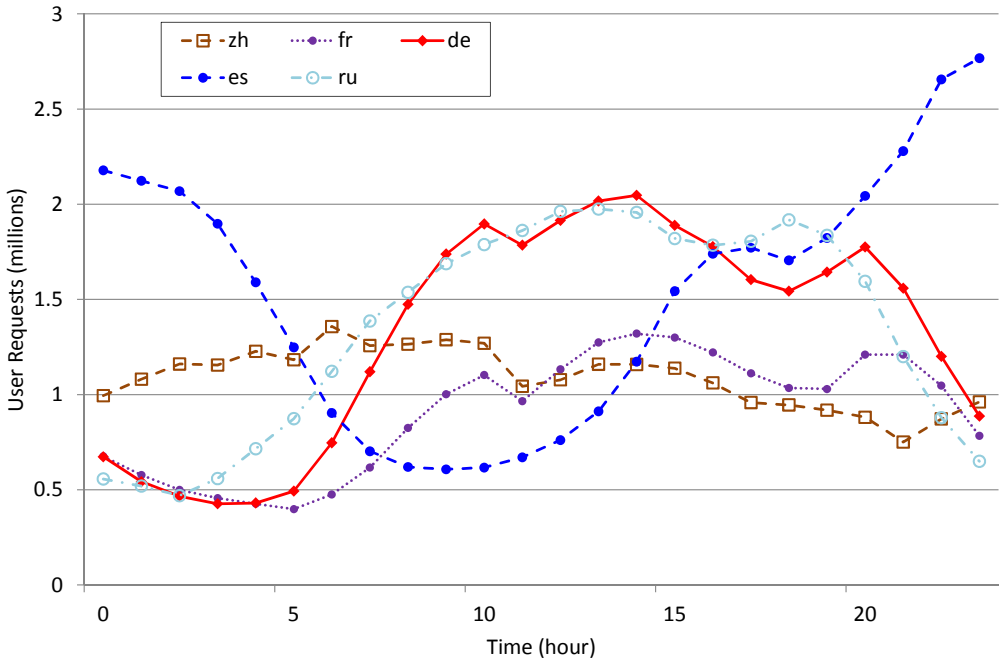



Figure 4.4: Wikipedia workload for different projects collected for 1st of Sep 2014.

trace. It is also freely available online with a large scale (159 million requests in total for 24 hours for 5 languages).

When the workload is supplied to the configured test bed, we can see the utilization of each VM follows the actual workload as depicted in Figure 4.5.

### 4.7.3 Initial placement

In this experiment set, we compare the initial placement algorithms without implementing any migration policy. We compare the algorithms in terms of SLA violation rates, energy consumption in hosts and network devices, and the energy savings.

**Investigating the impact of static overbooking on energy efficiency and SLA violation**

The aim of these experiments is showing the essence of designing dynamic overbooking algorithms that in comparison to static overbooking strategies reduce not only energy consumption but also SLA violations. First, we have conducted experiments to show how much energy we can save when we use static overbooking. Figure 4.6 shows SLA vi-

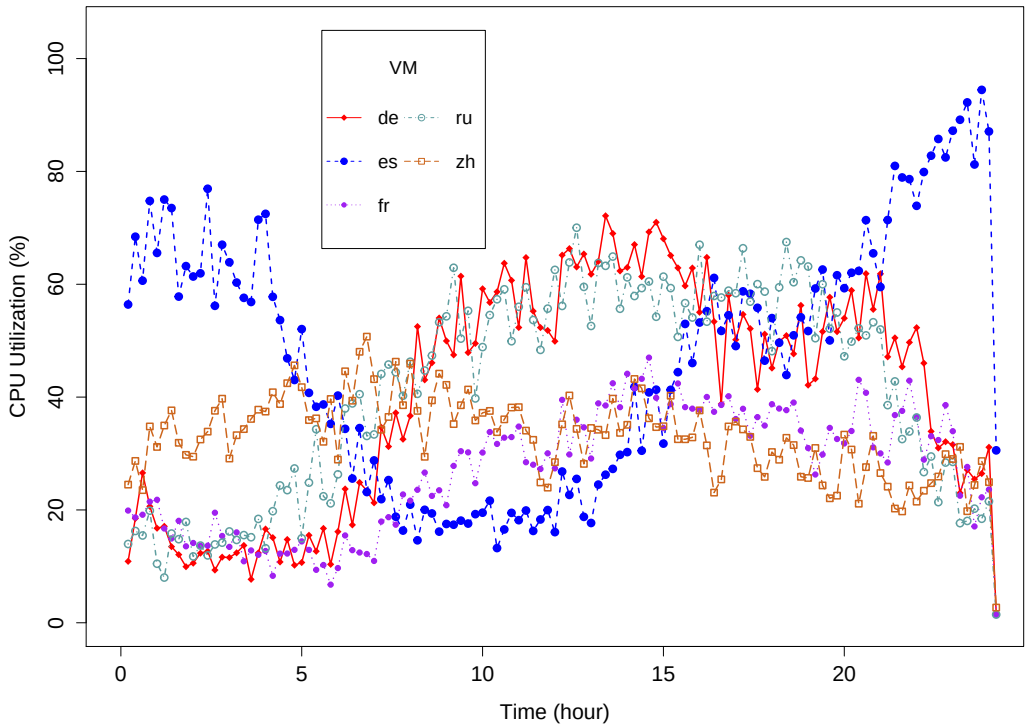
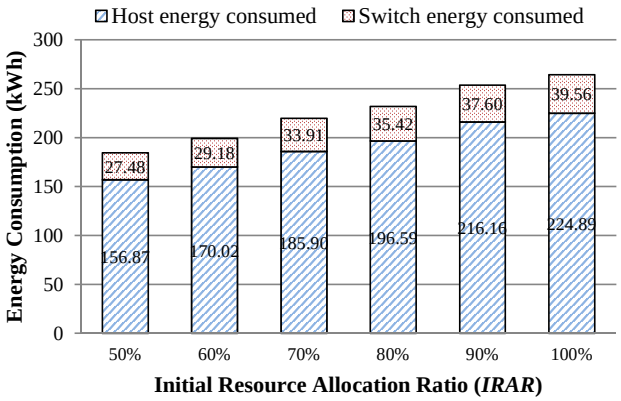


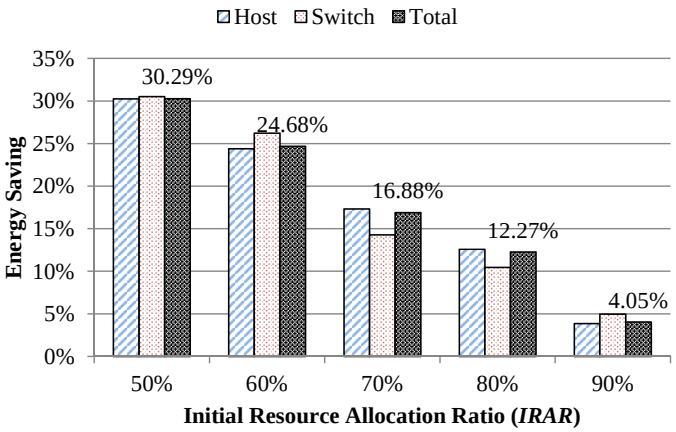

Figure 4.5: CPU utilization of VMs with Wikipedia workload

olation percentage and energy consumption of ConnNone which does static overbooking for initial placement. In ConnNone, resources are allocated to VMs with the fixed $IRAR$ without any consideration of connection between VMs. As shown in Figure 4.6a, overall energy consumption linearly decreases as $IRAR$ decreases. Allocating less resource lets hosts to accept more VMs, which leads to less number of active hosts. Network energy consumption also decreases with higher $IRAR$ value, because fewer hosts are communicating through less number of switches. Figure 4.6b shows the energy saving percentage of the static overbooking methods compared to the one without overbooking. For an extreme case when only 50% of the requested resources are allocated to VMs and networks, it can save 30.29% of the energy consumption in total. With 70% $IRAR$, the energy saving percentage reduced to 16.88% as less VMs can be placed in a single host with the higher $IRAR$.

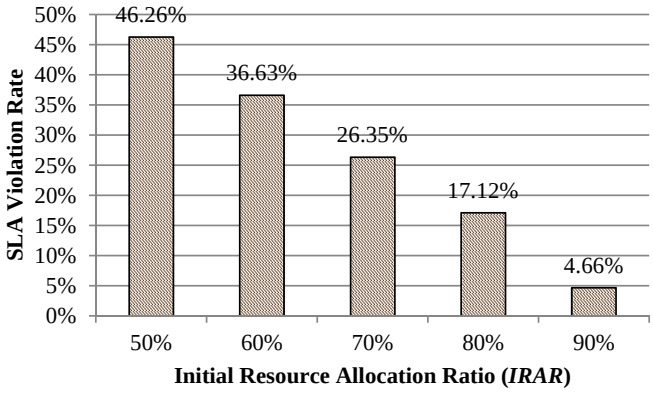
However, as Figure 4.6c shows, SLA violation increases significantly with lower $IRAR$ (note that the lower the $IRAR$ means that less resources were allocated to VM and vice versa). This is because the strategy has no consideration of either correlation or migration. As less resources are given to each VM, hosts and network are more frequently




(a) Energy consumption.




(b) Energy saving compared to baseline algorithm (NoOver).




(c) SLA violation percentage.

Figure 4.6: Energy consumption and SLA violation results of initial placement without consideration of connectivity.

overloaded, which leads to slower response of the workload and SLA violation. While the static overbooking with lower $IRAR$ can save more energy, it also increase the SLA

violation rate. Therefore, we need an algorithm that considers the trade-off between energy saving and SLA violation while dynamically adjusts overbooking ratio.

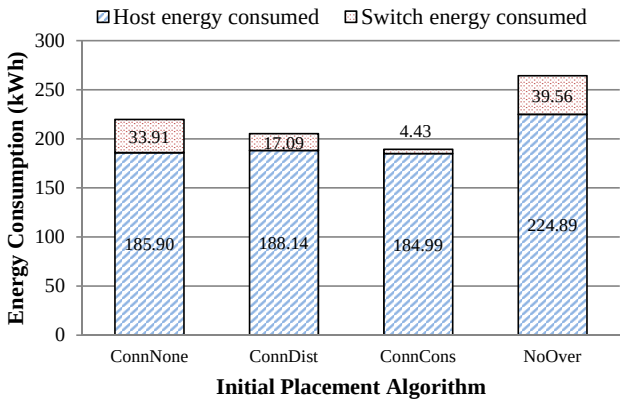
**Investigating the performance of different initial placement algorithms using static overbooking**

The aim is to compare the effects of placing connected VMs into a same host (ConnCons) with placing them in a different host (ConnDist). We expect that the connected VMs (especially for the 3 tier web applications) would have correlated workload, hence if they are placed in the same host, there is less chance for overbooking. However, if they are placed in the same host, network traffic and energy consumption would reduce since most network traffic between the connected VMs could be served within the host through memory instead of external network devices.

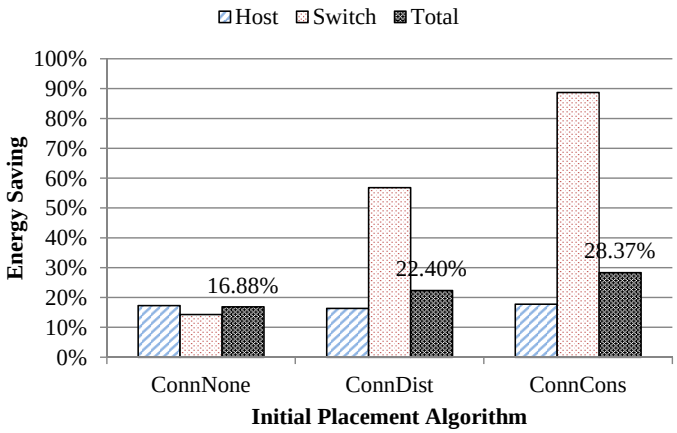
As shown in Figure 4.7a, energy consumption of the switches is significantly reduced under both ConnDist (connected VMs are placed in close distances but not to the same host) and ConnCons (connected VMs to the same host) compared to ConnNone which does overbooking but placing connected VMs to random hosts. Especially, in ConnCons only 4.43kWh electricity was consumed in switches which is almost one fourth of the switch energy consumption in ConnDist algorithm. It shows that network energy consumption is further reduced when connected VMs are placed in the same host.

Figure 4.7c shows SLA violation percentage of the ConnNone, ConnDist, and ConnCons algorithms with $IRAR$ setting at 70%. SLA violation percentages in ConnDist and ConnCons are still as high as ConnNone algorithm reaching at around 25% with 70% $IRAR$. Although ConnCons was expected to have less chance for overbooking that should result in more SLA violations, the experiment result shows that ConnDist results in a slightly more SLA violations than ConnCons algorithm. This is due to the characteristics of the workload that has less potential to the chance of overbooking.

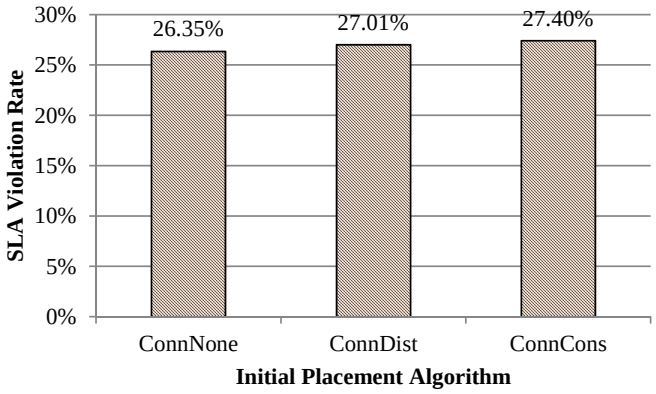
Figure 4.6b shows the energy saving percentage compared to the one without overbooking. As we discussed above, with ConnDist algorithm we can save over 50% of switches power usage, and with ConnCons the saving percentage reaches at almost 90% compared to non-overbooking method. Overall power saving is also increased in both ConnDist and ConnCons algorithms.




(a) Energy consumption.




(b) Energy saving compared to baseline algorithm (NoOver).




(c) SLA violation percentage.

Figure 4.7: Energy consumption and SLA violation results of different initial placement algorithms.

### 4.7.4 Migration policy

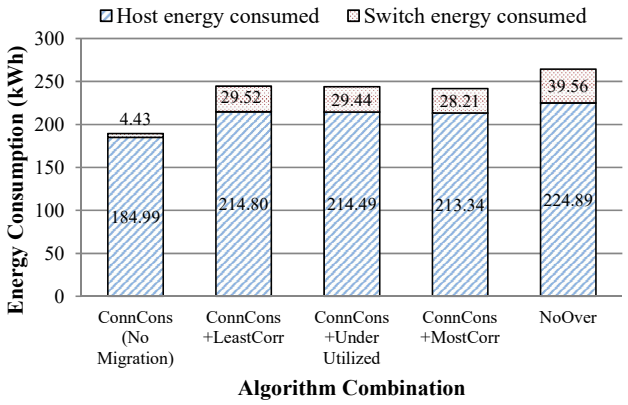
Migration policy plays important role when a host or network encounters overloading. Since overloading happens due to the lack of resources, migration of VMs from an overloaded host to a free host can resolve the overloading issue that might cause significant SLA violation. Several migration policies have been experimented with dynamic overbooking ratio.

**Investigating the impact of migration strategies**

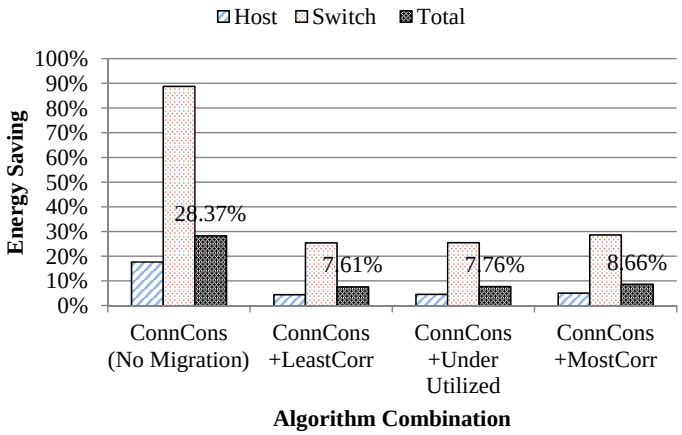
At first, we tested different migration strategies in the combination of ConnCons initial placement method. In this experiment we aim to find the effectiveness of the different migration strategies under the same initial placement. Figure 4.8a and 4.8b respectively show the total energy consumption and the percentage of energy saving of different migration algorithms. With any migration algorithm, energy consumption of hosts and switches increases compared to the one without migration. While ConnCons without migration can save 28.37% of power consumption, three algorithms with migration policies (ConnCons+LeasCorr, ConnCons+UnderUtilized, and ConnCons+MostCorr) can save between 7% and 8% of the total energy (Figure 4.8b). In detail, three migration algorithms use almost same amount of energy at both hosts and switches, and they still consume less power than the algorithm with no overbooking at all (Figure 4.8a).

However, as shown in Figure 4.8c, SLA violation decreases significantly when migration policies are implemented. While 27.40% of workloads violated SLA under ConnCons with no migration policy, just about 5% of workloads violated SLA when any migration algorithms was combined. In detail LeaseCorr migration algorithm results in the least SLA violation rate at 4.96%, and UnderUtilized policy results in 5.60% SLA violation rate, which is far less than the one without migration policy.

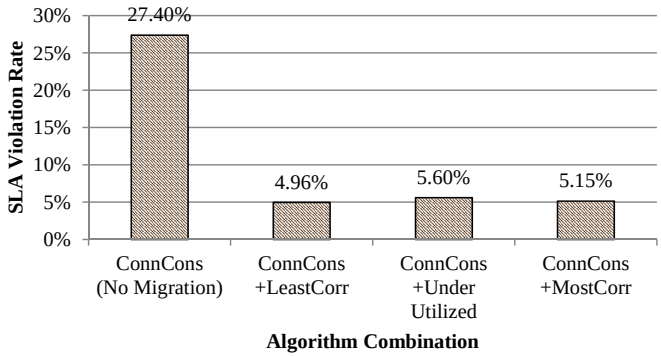
The results show the effectiveness of the dynamic overbooking strategy. As the migrating VM has been allocated to the host with dynamic overbooking ratio depending on the VMs in the host, it prevents highly correlated VMs to be consolidated into the same host. All three dynamic overbooking migration algorithms (MostCorr, LeastCorr, and UnderUtilized) show the similar results which significantly reduce SLA violation rate although they use various host selection methods to prioritize the candidate hosts.




(a) Energy consumption.



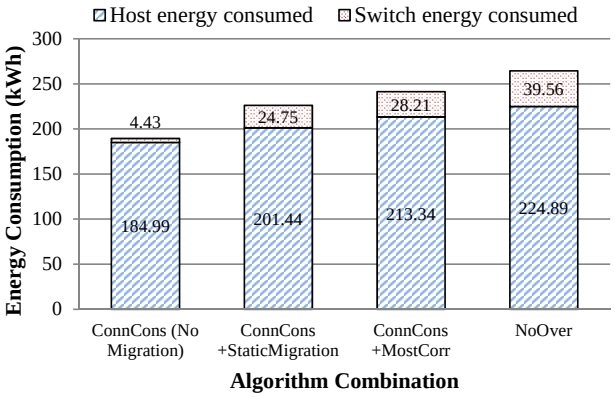

(b) Energy saving compared to baseline algorithm (NoOver).



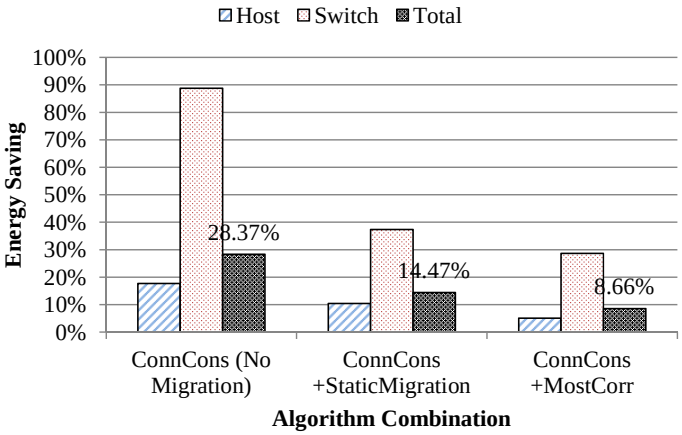

(c) SLA violation percentage.

Figure 4.8: Energy consumption and SLA violation results of different migration strategies implemented on ConnCons (connected VMs in the same host) initial placement algorithm.

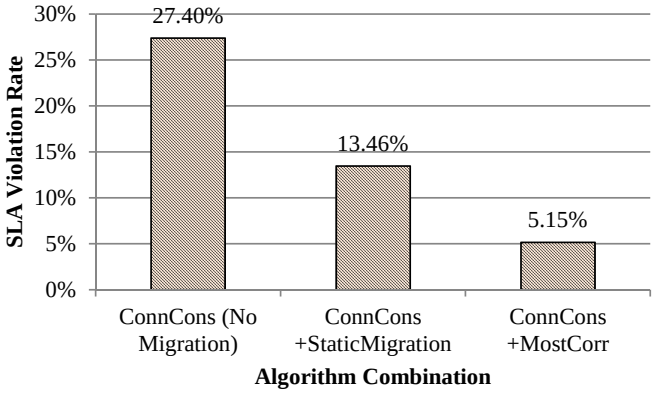
As we expected, dynamic overbooking can reduce the chance that all VMs in the host hit the peak at the same time, thus VMs acquire enough resources to process their workloads.




(a) Energy consumption.




(b) Energy saving compared to baseline algorithm (NoOver).




(c) SLA violation percentage.

Figure 4.9: Energy consumption and SLA violation results of dynamic and static overbooking algorithms

### Investigating the impact of dynamic overbooking ratio

Next, we investigated the impact of dynamic overbooking by comparing with a static

overbooking strategy under the same overbooking condition. The aim is to compare the effectiveness of our approach with a static overbooking algorithm similar to PowerNetS [132] which also implements overbooking with the consideration of correlation. Direct comparison to PowerNetS is not feasible because our approach is online algorithm without any prior knowledge of the workload, while PowerNetS acquired correlation of workloads in advance. Therefore, we use the results of ConnCons+StaticMigration combination which is the most analogous to PowerNetS. Both of them initially place connected VMs into closer hosts and migrate overloaded VMs to the nearest host where the connected VMs are placed. Note that ConnCons+StaticMigration algorithm is different from PowerNetS in the aspect that StaticMigration algorithm does not consider correlation threshold constraint which PowerNetS did implement. Thus ConnCons+StaticMigration algorithm would result in higher SLA violation rate than PowerNetS. We compare ConnCons+StaticMigration with ConnCons+MostCorr algorithm.

Figure 4.9 presents the difference of static overbooking and dynamic overbooking algorithms. As shown in Figure 4.9a and 4.9b, the static overbooking approach (ConnCons+StaticMigration) consumed slightly less energy than the dynamic method (ConnCons+MostCorr). In detail, 56.55 kWh is consumed across the whole data center for both hosts and network in the static overbooking method while 60.39 kWh is consumed in the dynamic overbooking which is 6.79% more than the static method. With the static algorithm, the overbooking ratio of the migrating VM is not changed in the new host when the in overloaded host is migrated to another host. Thus, regarding the entire data center more VMs can be placed in a host compared to dynamic overbooking which would allocate more resource for the migrating VM if correlated VM is in the migrating host.

The effectiveness of the dynamic overbooking can be clearly seen in SLA violation percentage presented in Figure 4.9c. SLA violation rate of the static overbooking algorithm (13.46%) is far higher than the dynamic algorithm (5.15%). Although our dynamic overbooking method consumed 6.79% more power, it dramatically reduced SLA violation rate by 61.74%.

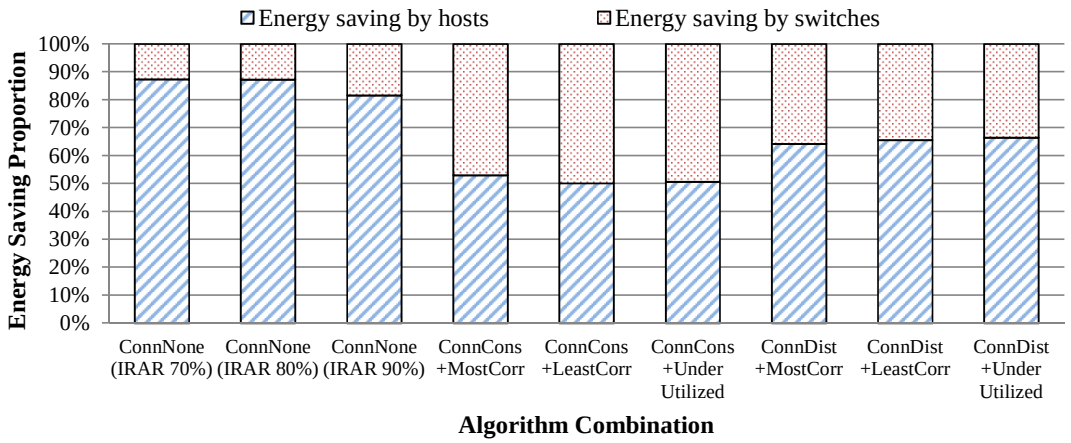
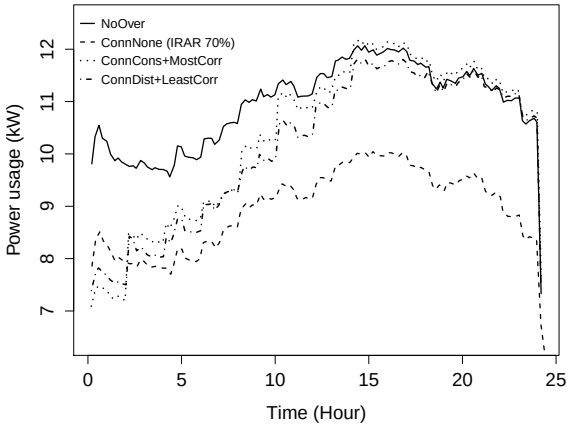


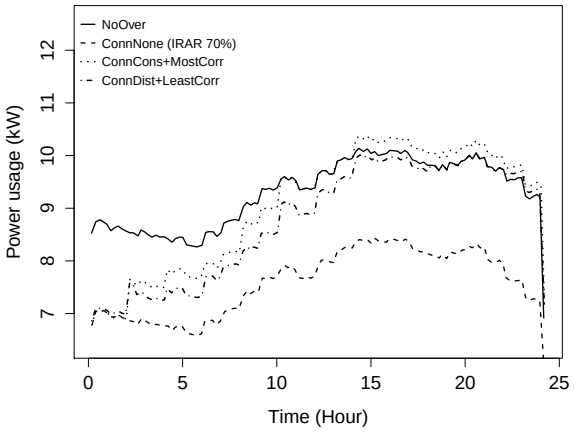

Figure 4.10: Energy saving origination

### 4.7.5 Analysis of energy consumption

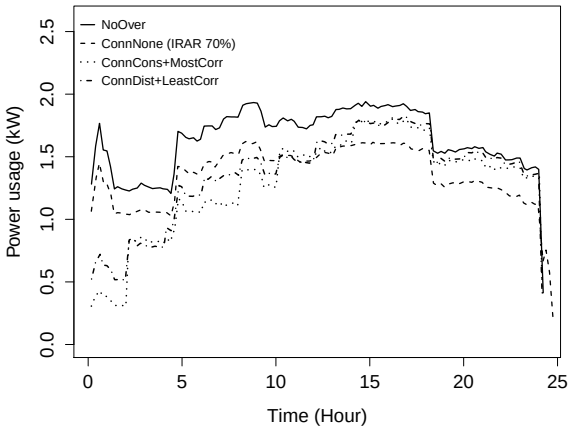
In this subsection, we investigate further details of the power consumption in different algorithm combinations. At first, we analyzed the origin of the energy savings where it comes from by showing the proportion of the saved energy in hosts and switches. Figure 4.10 shows the ratio of energy saving by hosts and switches. We tested ConnNone initial placement without migration in various $IRAR$ values (70%, 80%, and 90%) as well as the algorithms evaluated in the previous subsection. For each algorithm, energy consumption at hosts and switches is measured to calculate the saved energy compared to NoOver algorithm. For ConnNone algorithm, about 80% to 90% of the energy saving results from hosts while less than 20% from switches regardless of $IRAR$ value. However, when ConnCons initial placement (correlated VMs placed in closer hosts) is applied, energy saving ratio at switches increases significantly reaching at a half of the energy saving regardless of migration policy. This is because the consolidation of VMs can reduce significant amount of network traffic which leads to reducing the number of active switches. Interestingly, for ConnDist algorithm energy saving ratio of switches is lower than ConnCons but higher than ConnNone. As VMs in the same host are less likely peak at the same time in ConnDist algorithm, one host can hold more number of VMs than ConnCons algorithm which also affect the dynamic overbooking ratio adjusted by the correlation. In ConnDist initial placement, VMs in the same host would be less correlated which makes more VMs to be placed in one host at the migration stage, as $DRAR$ (Dynamic Resource Allocation Ratio) increases with lower correlation coefficient.




(a) Overall energy consumption.




(b) Energy consumed by hosts.



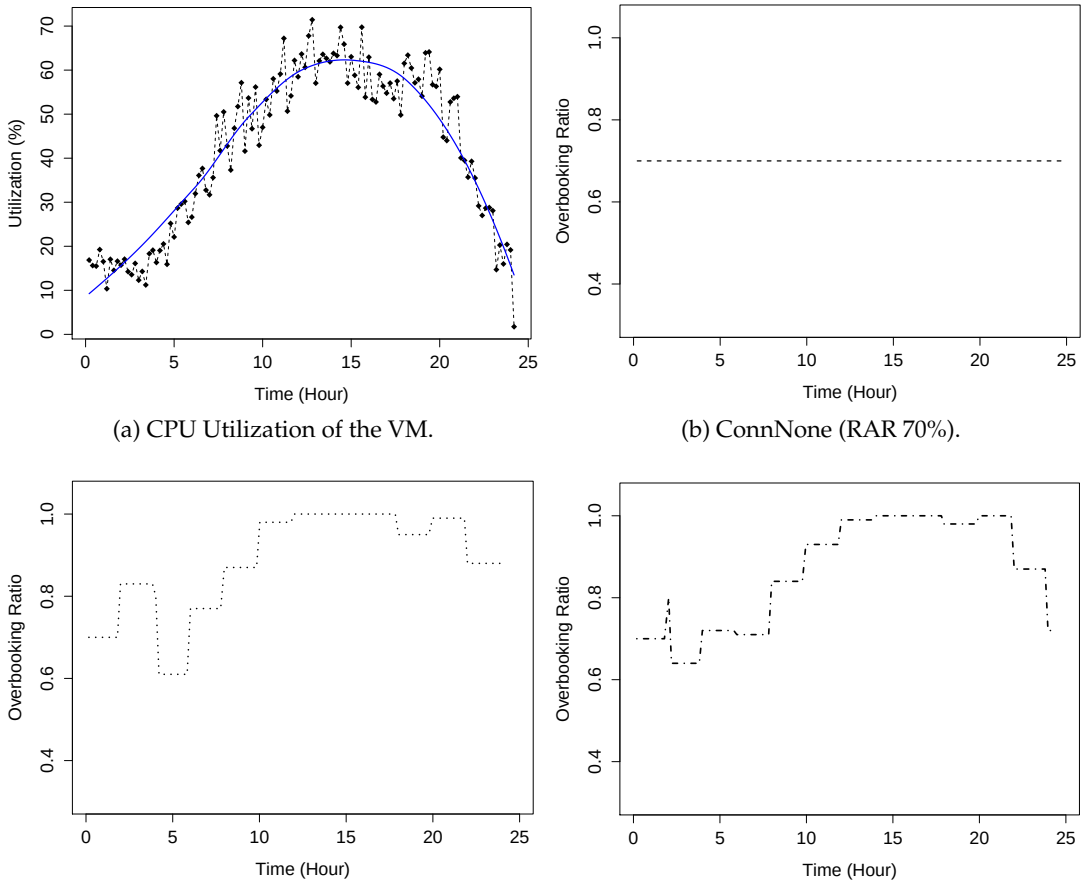

(c) Energy consumed by switches.

Figure 4.11: Energy consumption observation over time

### 4.7.6 Dynamic overbooking ratio

In order to investigate the impact of dynamic overbooking in energy consumption, we explored the power consumption of the whole data center (Figure 4.11a), energy consumption by hosts (Figure 4.11b), and by switches (Figure 4.11c) over the time. Compared to the baseline (NoOver), static overbooking method (ConnNone) uses constantly less amount of energy. Correlation-aware algorithms such as ConnCons+MostCorr and ConnDist+LeastCorr have less energy consumption in the beginning, but it converges to the baseline once time passes especially after the whole data center is highly loaded. For the network energy consumption, almost no energy is used with ConnCons algorithm at the beginning when most linked VMs are placed within the same host. However, as hosts get overloaded over the time, more switches are utilized which leads to consuming more energy. This result shows that how our algorithm reduces energy consumption and converges over time.

In this experiment, we investigated how overbooking ratio changes dynamically in correspondence with the workload. We randomly chose one sample VM from the previous experiments, and measured its CPU workload and Resource Allocation Ratio (RAR) in different algorithms. Figure 4.12 presents the CPU utilization level and the Resource Allocation Ratio of the VM changing over time. The first figure (4.12a) shows the CPU utilization of the VM without overbooking in correspondence with its workload. It is obvious that the VM consumes more CPU resource when there is more load. Figures 4.12b, 4.12c, and 4.12d show the Resource Allocation Ratio of the VM in different overbooking algorithms. For the static overbooking method without migration (ConnNone), the VM acquires only 70% of requested resource all the time constantly as the RAR sets to 70% without dynamic overbooking strategy. However, with our proposed dynamic overbooking algorithms (ConnCons+MostCorr and ConnDist+LeastCorr), RAR continuously changes over time following the actual workload. As we set up Initial Resource Allocation Ratio (IRAR) to 70%, the RAR starts at 0.7 in both algorithm, and dynamically fluctuates over time following the actual CPU utilization shown in Figure 4.12a. The result shows that the overbooking ratio reflects the real workload, so that the VM acquires more resources when necessary.




(a) CPU Utilization of the VM.

(b) ConnNone (RAR 70%).

(c) ConnCons+MostCorr.

(d) ConnDist+LeastCorr.

Figure 4.12: CPU utilization and Resource Allocation Ratio of a sample VM.

## 4.8 Summary

In this chapter, we presented dynamic overbooking strategies that allocate host and network resources dynamically adapting based on the utilization. The variety of workloads affects the dynamic overbooking ratio in real-time through correlation analysis of the VMs and network utilization. By leveraging the dynamic overbooking method, tightly suitable amount of resources can be allocated to VMs which will maximize energy cost savings by reducing the waste of over-provisioned resource, and at the same time minimize SLA violation by allocating enough resource for the actual workload. With the extensive experiments, we demonstrated that our approach can effectively save energy consumption of the cloud data center while reducing SLA violation rates compared to the baseline.

# Chapter 5

# Priority-aware Joint VM and Network Resource Provisioning

*In this chapter, we propose priority-aware resource placement algorithms considering both host and network resources. Our priority-aware VM allocation (PAVA) algorithm places VMs of the priority application to closely connected hosts to reduce the chance of network congestion caused by other tenants. The required bandwidth of a critical application is also guaranteed by bandwidth allocation with a configuration of priority queues on each networking device in a data center network managed by SDN controller. Our experiment results show that the combination of proposed approaches can allocate sufficient resources for high priority applications to meet the application's QoS requirement in a multi-tenant cloud data center.*

## 5.1 Introduction

THE emergence of Software-Defined Networking (SDN) enables a fulfillment of network QoS satisfaction by the introduction of dynamic network reconfiguration based on the network traffic. SDN has brought many opportunities in networking with centralized manageability and programmable control logic. In SDN, a controller oversees the entire network by gathering all information of every network device and manages the network traffics dynamically with the customized control logic. SDN integration in a cloud data center has shown to be effective to improve the energy efficiency [48,124,133], the network performance [71,96,119], the network availability [5], and the security [22]. It also enables network slicing and dynamic bandwidth allocation which can be exploited

This chapter is derived from: Jungmin Son and Rajkumar Buyya, "Priority-aware VM Allocation and Network Bandwidth Provisioning in SDN-Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, 2018 (accepted, in press).

for QoS satisfaction [3, 43].

In this work, we propose a novel VM and network allocation approach (PAVA+BWA) in the combination of a Priority-Aware VM Allocation (PAVA) algorithm considering network connection between VMs on the application level with a network Bandwidth Allocation (BWA) algorithm to differentiate the higher-priority flows over normal network traffics. These algorithms are to allocate enough resources for QoS-critical applications in cloud environments where the computing and networking resources are shared with other tenants. We distinguish such applications to give the higher priority over the other tenants for resource provisioning and network transmission.

We model an application based on its priority to differentiate in VM capacity and network bandwidth. Our approach can allocate sufficient computing and networking resources for a critical application with high priority even in a busy data center. We employ a network bandwidth allocation strategy enabled by SDN for the critical application traffic so that the application can complete network transmission on time regardless of the network condition. With our approach, QoS-critical applications can be served in-time on clouds, while other applications can still share the resources for the rest of time. It considers host and networking resources jointly to prevent the network delay which can cause QoS failure. The applications' QoS requirements are assumed to be provided to the cloud management at the time of the request. Using the metrics of QoS requirements including computing capacity and network bandwidth, the proposed algorithm determines where to place VMs and flows. The algorithm considers both computing and networking requirements jointly to select the host to place the VM and the links between hosts. After selecting the network links, we use dynamic bandwidth allocation to meet the networking requirement.

The key **contributions** of this chapter are:

- a priority-aware VM placement algorithm that places VMs of a critical application into proximity hosts with enough resources;
- a bandwidth allocation method for higher-priority flows to guarantee the minimum bandwidth in overloaded data center networks;
- a system that provisions both compute and network resources jointly to offer qual-

Table 5.1: Summary of related works.

| Work | VM allocation unit | VM placement method | VM type | Traffic management | Parameters | Energy efficiency |
|---|---|---|---|---|---|---|
| MAPLE [119, 121, 122] | Connected VMs | Network-aware | Homogeneous | Dynamic routing | Estimated effective bandwidth | x |
| EQVMP [123] | Connected VMs | Hop reduction | Homogeneous | Dynamic routing | VM traffic | ✓ |
| S-CORE [27] | Not Applicable (NA) | Migration only | Homogeneous | VM migration to close hosts | VM traffic | x |
| QVR [71] | NA | NA | NA | Dynamic routing & Bandwidth allocation | Delay, jitter, packet loss | x |
| FCTcon [133] | NA | NA | NA | Bandwidth allocation | Flow completion time | ✓ |
| DISCO [134] | NA | NA | NA | Flow consolidation | Traffic correlation, delay | ✓ |
| **PAVA+BWA** | Application level | Priority-aware | Heterogeneous | Bandwidth allocation | Priority, VM capacity, bandwidth requirement | ✓ |

ity of service to critical applications in SDN-enabled cloud data centers;

- a performance evaluation of our proposed algorithm that is compared with related approaches and depicted its effectiveness through detailed simulation experiments using both synthetic and real (Wikipedia) workloads.

This chapter is organized as follows. We discuss the state-of-the-art approaches in Section 5.2. The overall system architecture is explained in Section 5.3 followed by a detailed explanation of the proposed algorithm along with baselines. Section 5.5 presents experiment configuration and evaluation results. Finally, Section 5.6 summarizes the chapter.

## 5.2 Related Work

Many approaches have been proposed to network-aware VM placement and SDN-based network flow scheduling. Wang et al. proposed MAPLE [121, 122], a network-aware VM placement system that exploits an estimated bandwidth usage of a VM. In MAPLE,

authors calculated estimated bandwidth based on empirical traces collected from the servers. The algorithm uses First Fit Decreasing approach in order to determine a host to place the VM. MAPLE focuses on per-VM network requirement and assumes that all VMs have a homogeneous processing requirement.

MAPLE project is extended to MAPLE-Scheduler, a flow scheduling system that dynamically reschedule the network flows based on QoS requirement of each VM [119]. The authors implemented dynamic network flow scheduler to relocate flows based on the estimated bandwidth usage. At the beginning, the system finds a default route using equal-cost multi-path (ECMP) protocol [49] which is a widely adopted protocol for multi-path routing that distributes network traffics evenly over multiple paths. Then, the system detects potential flows that can violate the QoS and reschedules them to an alternate path where the QoS can be satisfied.

EQVMP is another VM placement approach that is aware of energy efficiency and QoS [123]. EQVMP partitions VMs into groups based on traffic matrix to reduce the inter-group traffic and balance the load in each group. VM groups are placed onto hosts using bin packing algorithm to reduce the number of hosts and minimize the energy consumption. After the placement, EQVMP performs load balancing which detects over-utilized network link and relocates flows in the link. The authors assume that the capacity requirement of VMs are homogeneous and considers only network requirement.

S-CORE has been proposed for SDN-based VM management that exploits a VM migration technique to minimize the network-wide communication cost for cloud data centers [27]. The system monitors VM traffic flows periodically and migrates VMs with large network traffics into close hosts to reduce the network cost. The prototype is implemented both on a KVM-based test-bed and in a simulation and evaluated with synthetic workloads. Although the system considers network traffics of VMs to minimize the total network cost through the network monitoring capability of SDN, it is incapable of dynamic traffic management in the congested network.

Lin et al. proposed QoS-aware virtualization and routing method (QVR) that isolates and prioritizes tenants based on QoS requirements and allocates network flows dynamically [71]. The system supports fine-grained network virtualization based on tenants and the network requirement. The flows are allocated dynamically onto physical links

considering the minimum arrival rate and maximum utilization. The system also uses adaptive feedback traffic engineering to ensure the end-to-end QoS. The proposed approach is applied to a wide area network and compared with other routing protocols in a simulation. Their approach is applicable to different network topology but more focused on wide area network.

FCTcon has been proposed to improve energy efficiency in data center networks by dynamically managing the flow completion time [133]. The proposed method consolidates network flows into a smaller set to save energy usage in a data center. In order to prevent performance degradation, the system calculates an expected flow completion time and dynamically adapts the flow bandwidth to meet the required completion time for delay-sensitive flows. In their system, the controller receives feedback of flow completion time from a monitor constantly monitoring the data center network and updates bandwidth allocation and flow paths to consolidate flows while preventing delay of sensitive flows.

DISCO is also proposed by the same authors to increase the scalability of traffic flow consolidation technique for energy efficient data center networks [134]. In this work, the authors addressed the trade-offs between scalability, network performance, and energy efficiency in a large-scale data center network. Network traffics within a data center can be consolidated into a smaller number of network links and switches to save power consumption by turning off unused switches. However, the high complexity of the consolidation algorithm in SDN's centralized model can result in a scalability issue for a data center. The authors proposed a distributed traffic management system to support a scalable traffic consolidation technique. The proposed system also considers network delay as a constraint to improve the network performance. Nevertheless, only network traffic is taken into account in this work without consideration of computing resources.

Table 5.1 summarizes related works and compares with our approach (PAVA+BWA). Unlike the aforementioned studies, our approach prioritizes critical applications over the other applications considering application-level resource requirements. Instead of considering individual VMs and flows, a set of multiple VMs consisting of the same application (e.g., web and database servers for a web application, or mappers and reducers for a MapReduce application) are taken into the consideration for resource provisioning.

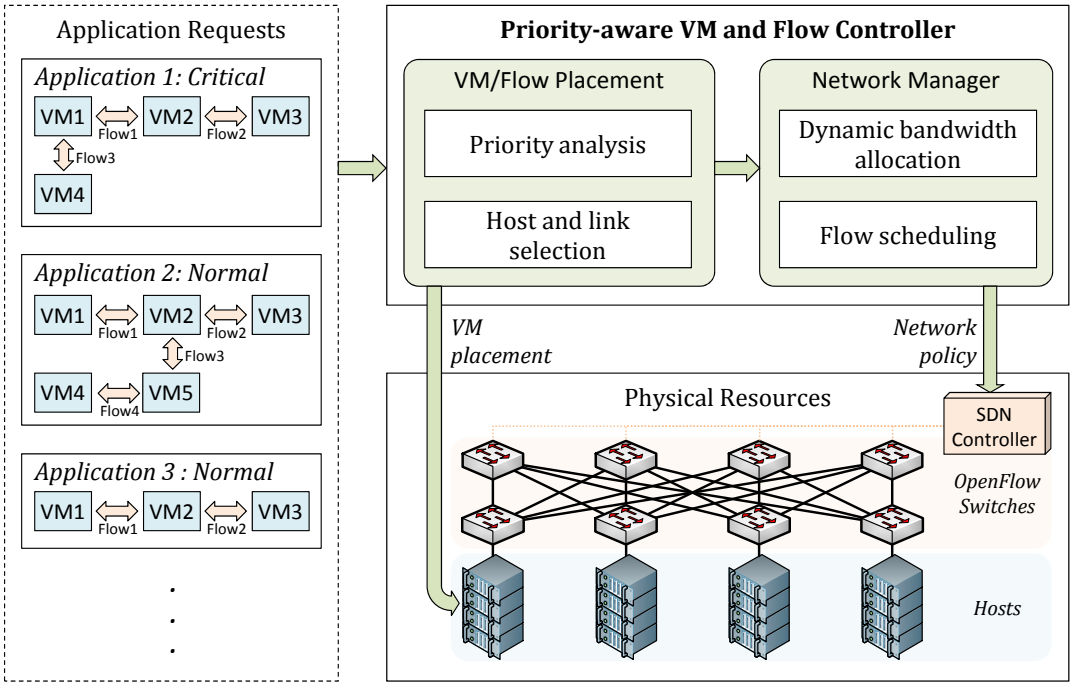



Figure 5.1: Architectural design of priority-aware VM and flow management system.

In our model, VMs can be heterogeneous, i.e., each VM can have different computing and networking requirements which are managed by our VM allocation and network bandwidth allocation system. Parameters considered in the proposed method includes priority, VM processing capacity, and flow bandwidth requirements. We also consider the energy efficiency of a data center which may be affected by the implementation of the algorithm.

## 5.3 System Architecture

Figure 5.1 shows overall architecture of our system along with flows of interaction between different components. In brief, application requests are submitted to the system with a priority information (critical or normal). Each application request consists of an arbitrary number of VMs and flows with detailed specifications. Priority of an application is then analyzed for VM and flow placement to provide sufficient resources for a critical application as well as to prioritize network traffic over normal applications. Based on the analyzed information, the host and link selection algorithm determines where to place VMs and flows. Flow information is also sent to the network manager to communicate

with the SDN controller for dynamic bandwidth allocation and flow scheduling. The detail of each component is explained below.

At first, application requests are created by cloud tenants to serve their applications and include VM types and flow information. A VM type consists of the number of processing cores, each core's processing capacity, the amount of memory, and the storage size. In this research, we focus on computing power and ignore memory and storage size in order to reduce the problem complexity. An application request also includes flow specification consisting of a source and destination VM and the required bandwidth. VM and flow specifications are modeled based on commercialized cloud providers such as Amazon AWS and Microsoft Azure who provide predefined VM types and customizable virtual networks. We add an extra entity, application priority, to this model to differentiate applications. Note that either critical or normal application request can be submitted at arbitrary time, and the system provisions the resources available at the time of the application submission. If a critical application is submitted after a normal one, the system allocates residual resources to the critical one with a different provisioning method.

Given the VM and flow requests, the system analyzes QoS requirements and prioritizes the VMs and flows accordingly. Prioritization can be performed in various ways, such as determined based on the VM capacity, network requirements, or specified by the tenants. Although our system model is capable of any prioritization method, in this chapter we assume that the application priority is specified by user in binary value (critical or normal) and supplied to the system along with the application request. This is to simplify the prioritization process and focus on the resource prioritization method.

After the analysis of an application priority, the system selects a host and links to place the requested application by running the VM placement algorithm, which is proposed in this chapter that jointly considers computing and networking requirements. Once the selection is done, the result is passed to the network manager for network resource configuration and to the physical resources for actual allocation of the VM and flow.

Network manager ensures to allocate minimum bandwidth for a critical application even in network congestion. It is in charge of SDN controller that dynamically allocates bandwidth for each flow. By default, the bandwidth of a link is equally shared among the flows passing the link, but a higher-priority flow can acquire more bandwidth through

implementing a priority queue on switches such as queuing disciplines (*qdisc*) and Hierarchy Token Bucket (*HTB*) implemented in Linux's traffic control (*tc*) suite. In order to control flows and their network bandwidths, the network manager communicates with SDN controller to provision the networking resources dynamically. SDN controller manages all switches in the data center through OpenFlow protocol.

## 5.4 Priority-aware Resource Provisioning Methods

In this section, we explain the strategy for computing and networking resource allocation in aware of application's priority. We model the priority of application discretely. Each application provides its priority as a *critical* (higher-priority) or *normal* (lower-priority) application. It can be manually set up by tenants who submit the application request to a cloud provider or automatically configured by cloud provider based on the application information. We assume that there is a reasonable mixture of critical and normal application requests in a data center so that not all the applications running in a data center are set to be critical applications. If all of them have the same priority, they will be served equally. Resources are allocated without any special consideration for normal applications, whereas critical applications are ensured to get sufficient computing power and prior network transmission over the normal applications.

### 5.4.1 Priority-Aware VM Allocation (PAVA)

For VM allocation, we propose Priority-Aware VM Allocation (PAVA) algorithm (shown in Algorithm 5) which allocates a closely connected host for a critical application with the information of network topology. In this method, we consider the priority of application as well as the network connectivity of physical hosts in a data center. Before allocating VMs, the system gathers the network topology of the data center. All hosts in a data center are grouped by their connectivity. For example, hosts in the same rack connected to the same edge switch will be grouped together. VMs are also grouped based on the application.

After grouping hosts, PAVA algorithm is running to find a suitable VM-host mapping to place a VM for a critical application. PAVA tries to place a set of VMs for the critical

application onto the host group which has more computing and networking resources. If the group of VMs cannot be fit in a single host group, VMs will be placed across multiple host groups with the closest proximity (e.g., under a different edge switch in the same pod). In this way, VMs in the same application will be closely placed to maintain the minimal number of hops for network transmission. For example, a network traffic between VMs placed within the same host is transferred through the host memory without incurring any traffic to networking devices. Similarly, if the VMs are hosted under the same edge network or in the same pod, a cost for network transmission between those VMs can be reduced with a lower probability of interference by the other applications. PAVA guarantees VMs for the critical application to be placed on not only the host with the sufficient capacity but also within the same or closer hosts in network topology to reduce the communication cost. We use First Fit Decreasing (Algorithm 7) for normal applications.

### 5.4.2 Bandwidth Allocation for Priority Applications (BWA)

As cloud data center's network infrastructure is shared by various tenants, providing constant network bandwidth is crucial for application's QoS to avoid performance degradation in traffic congestion caused by other tenants. We propose bandwidth allocation (BWA) approach to allocate the required bandwidth for a critical application. This approach utilizes per-flow traffic management feature for virtualized network [67, 109]. SDN controller can manage switches to allocate requested bandwidth by configuring priority queues in switches to ensure privileged traffic transmitting over the other lower-priority traffics.

After VM placement process is completed, the bandwidth requirement and the virtual network information of a critical application are sent to the SDN controller. SDN controller then establishes priority queues (e.g., Linux qdisc and HTB) for the critical application flows on every switch along the link. Network traffic generated from VMs of the critical application will use the priority queue so that the required bandwidth can be obtained for the critical applications. This method is only applied to critical applications in order to prioritize network transmission over the normal traffic in the shared nature of data center networking. It ensures that the critical application can get enough bandwidth

**Algorithm 5** Priority-Aware VM Allocation (PAVA)

**Data**: $vm$: VM to be placed.
**Data**: $rd$: Resource demand of $vm$;
**Data**: $app$: Application information of $vm$.
**Data**: $H$: List of all hosts in data center.
$H_{group} \leftarrow$ Group $H$ based on edge connection;
$Q_H \leftarrow$ Empty non-duplicated queue for candidate hosts;
$placed \leftarrow$ false;
**if** $app$ is a priority application **then**
    $H_{app} \leftarrow$ list of hosts allocated for other VMs in $app$;
    **if** $H_{app}$ is not empty **then**
        $Q_H$.enqueue($H_{app}$);
        **for each** $h_a$ in $H_{app}$ **do**
            $H_{edge} \leftarrow$ A host group in $H_{group}$ where $h_a$ is included ;
            $Q_H$.enqueue($H_{edge}$);
        **end for**
        **for each** $h_a$ in $H_{app}$ **do**
            $H_{pod} \leftarrow$ Hosts in the same pod with $h_a$;
            $Q_H$.enqueue($H_{pod}$);
        **end for**
    **end if**
    sort $H_{group}$ with available capacity, high to low;
    $Q_H$.enqueue($H_{group}$);
    **while** $Q_H$ is not empty **and** $placed$ = false **do**
        $h_q = Q_H$.dequeue()
        $C_h \leftarrow$ free resource in host $h_q$;
        **if** $rd < C_{h_q}$ **then**
            Place $vm$ in $h_q$;
            $C_h \leftarrow C_h - rd$;
            $placed \leftarrow$ true;
        **end if**
    **end while**
**end if**
**if** $placed$ = false **then**
    Use FFD algorithm to place $vm$;
**end if**

even in a congested network caused by the other application. Algorithm 6 explains the detailed procedure of BWA method. For all flows, the algorithm sets the default path using ECMP, which distributes network traffic based on the address of the source and destination hosts. For higher-priority flows, the algorithm sets up an extra flow rule in each switch along the path. The priority queue set for the higher-priority flow can guarantee the minimum bandwidth required by the application. For lower-priority flows, the

**Algorithm 6** Bandwidth Allocation for critical applications (BWA)

```
1: Data: F: List of network flows.
2: Data: topo: Network topology of the data center.
3: for each flow f in F do
4:     h_src ← the address of the source host of f;
5:     h_dst ← the address of the destination host of f;
6:     S_f ← list of switches between h_src and h_dst in topo;
7:     for each switch s in S_f do
8:         if f is a priority flow then
9:             s.setPriorityQueue(h_src, h_dst, f.vlanId, f.bandwidth);
10:        end if
11:    end for
12: end for
```

algorithm only sets a default path using ECMP without a further configuration.

### 5.4.3 Baseline algorithms

The proposed approaches are compared with three baseline algorithms: exclusive resource allocation, random allocation, and state-of-the-art heuristic.

Exclusive Resource Allocation (*ERA*) is to allocate dedicated hosts and networks exclusively for a critical application, thus resources are not shared with any other tenants. The application can fully utilize the capacity of the dedicated resources to process its workloads, as the required computing and networking resources can be obtained without any interference from other applications. However, all the benefits of cloud computing will be lost including elasticity and dynamicity in this method. It is impractical in reality because exclusively allocated resources will result in an extravagant cost for cloud providers which will be passed on to the customers. In this chapter, we use this algorithm only for measuring the expected response time of a critical application to calculate QoS violation rate. Details of how to calculate QoS violation rate is explained in Section 5.5.3.

Random allocation (*Random*) is to place a VM on a random host capable of providing enough resources for the VM. In this method, a host is randomly selected with no intelligence but solely based on the resource capacity.

The state-of-the-art heuristic baseline algorithm is to place VMs in First Fit Decreasing (FFD) order determined by the amount of required bandwidth, which is combined with a Dynamic Flow (DF) scheduling method for network traffic management. FFD and

DF are derived from MAPLE project [119, 121, 122], where the applications are equally considered for VM allocation and flow scheduling based on their bandwidth requirement regardless of applications' priority. Details of baselines are explained below.

**First Fit Decreasing (FFD)**

*FFD* searches a host to place the VM in the first fit decreasing order based on the bandwidth. It consolidates more VMs into a host with enough resources and does not distribute them across the data center. Thus, VMs are placed into a smaller set of hosts, whereas other empty hosts can put into an idle mode which can increase the energy efficiency of the entire data center. This baseline is derived from MAPLE [121, 122], but instead of Effective Bandwidth, we use the VM's requested bandwidth to determine the resource sufficiency of a host for a VM. In our system, we do not need to calculate a separate Effective Bandwidth because the required bandwidth is predefined in the application specification.

In addition to the power saving at hosts, FFD can also reduce the energy consumption of switches. When more VMs are placed on the same host, the possibility of in-memory transmission between VMs in the same host is increased which emits the network transmission over the switches. Although the algorithm does not consider the network condition in itself, it can affect the amount network traffic to some extent from the nature of the algorithm. The pseudo-code of the algorithm is presented in Algorithm 7.

**Dynamic Flow Scheduling (DF)**

On multi-path network topology, dynamic flow scheduling is a common approach to find an alternate path for a flow in case of network congestion or link error suggested in multiple studies [119, 123]. This method detects a congested link and relocates the flows in the congested link into an alternate path with more capacity. However, based on our observation, this approach is less effective for short-distance flows in Fat-tree topology due to Fat-tree's architectural advance which can achieve a network over-subscription ratio of up to 1:1 [4]. For an edge switch in Fat-tree, the number of downlinks to the connected hosts are same as the number of up-links to the aggregation switches, and the

**Algorithm 7** First-fit decreasing for bandwidth requirement (FFD)

```
1: Data: VM: List of VMs to be placed.
2: Data: H: List of hosts where VMs will be placed.
3: for each vm in VM do
4:     sort H with available resource, low to high;
5:     for each h in H do
6:         C_h ← free resource in host h;
7:         rd ← resource demand of vm;
8:         if rd ¡ C_h then
9:             Place vm in h;
10:            C_h ← C_h − rd;
11:            placed ← true;
12:            break;
13:        end if
14:    end for
15: end for
```

traffic flows are equally distributed based on the address of the source and the destination host. Thus, the network traffic is already balanced among the links between edge and aggregation switches in the same pod. The algorithm is still effective for inter-pod traffics because the number of links between aggregation and core switches is less than the ones between aggregation and edge switches. When the link to a core switch is congested, DF algorithm can relocate the flows into an alternate link to the other core switch with less traffic.

The pseudo-code of DF scheduling algorithm is described in Algorithm 8 which is derived from MAPLE-Scheduler [119]. DF is used as a baseline to compare with our bandwidth allocation approach. The algorithm is applied periodically to find the least busy path for higher-priority traffics, while normal traffics still use the default path determined by ECMP based on the source address.

## 5.5 Performance Evaluation

The proposed algorithms are evaluated in a simulation environment. Two use-case scenarios are prepared to show the effectiveness of PAVA and BWA: a straightforward network-intensive application and more practical 3-tier application. We measure the response time of workloads to check the impact of the algorithms on both critical and normal applica-

**Algorithm 8** Dynamic Flow scheduling algorithm (DF)

**Data**: $f$: A network flow to be scheduled.
**Data**: $h_{src}$: the address of the source host of $f$.
**Data**: $h_{dst}$: the address of the destination host of $f$.
**Data**: $topo$: Network topology of the data center.
$s \leftarrow$ next hop from $h_{src}$ for flow $f$ in $topo$
**while** $s$ is not $h_{dst}$ **do**
    $L \leftarrow$ available links on $s$ for flow $f$.
    $l_{Next} \leftarrow h_{src}$ **mod** $L$.size() (default path);
    **if** $f$ is a priority flow **then**
        **for each** link $l$ **in** $L$ **do**
            **if** $l$.utilization() $< l_{Next}$.utilization() **then**
                $l_{Next} \leftarrow l$;
            **end if**
        **end for**
    **end if**
    $s$.updateNextHop($f$, $l_{Next}$);
    $s \leftarrow l_{Next}$
**end while**

tion's performance. Energy consumption of the data center and the number of active hosts and their up-time are also measured to consider the cost of cloud providers.

### 5.5.1 Experiment configuration and scenarios

For evaluation, we implemented the proposed method and baselines on CloudSimSDN simulation environment which is described in Chapter 3. CloudSimSDN is an extension of CloudSim [18] simulation tool that supports SDN features for cloud data center networks. In CloudSimSDN, we generated an 8-pod fat-tree topology data center network with 128 hosts connected through 32 edge, 32 aggregation, and 16 core switches. Thus, each pod has 4 aggregation and 4 edge switches, and each edge switch is connected to 4 hosts. Figure 5.2 shows the topology of the configured cloud data center for the experiment. All physical links between switches and hosts are set to 125 MBytes/sec.

In the aforementioned simulation environment, we evaluate our approach in two scenarios.

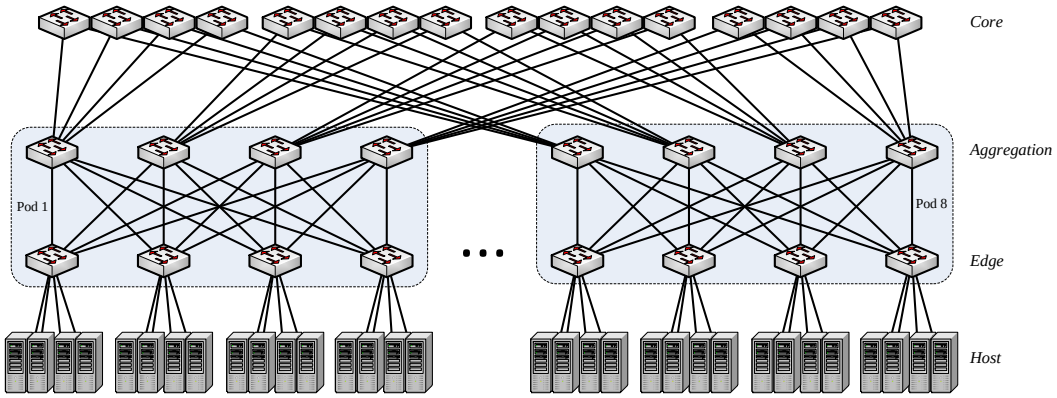



Figure 5.2: 8-pod fat-tree topology setup for experiments.

### A Scenario 1: synthetic workload

The first scenario is to place a critical application in an overloaded data center environment. To make the data center overloaded, 15 lower-priority applications, consisting of 16 VMs in each application, are firstly placed in the data center that constantly generates network traffics. After these VMs are placed, a higher-priority application consisting of the same number of VMs is submitted to the data center. Once all VMs are placed using the proposed PAVA algorithm, synthetically generated workloads are submitted to the critical application, which has both computing and networking loads. The BWA method is applied to transfer the networking part of the critical application workloads. This scenario is to test the effectiveness of PAVA and, especially, BWA in a condition that the higher-priority application is significantly interfered by other applications. Please note that the workloads are synthetically generated which keeps sending network traffics to each other within the same application, in order to evaluate the effectiveness of the network traffic management schemes.

### B Scenario 2: Wikipedia workload

The second scenario reflects a more practical situation where applications are placed on a large-scale public cloud that a massive number of VM creation and deletion requests are submitted every minute. Frequent VM creation and deletion result in a fragmented data center. Network traffics generated by the scattered VMs can increase the overall load of the data center network, which makes the network traffic management more critical in

applications' performance.

We create 10 different application requests modeled from three-tier web applications. Each application consists of 2 database, 24 application, and 8 web servers communicating to one another. One out of the 10 applications is set to be critical, while the rest to be normal. The size of VMs are varied based on the tier, e.g., database tier servers are defined having 2 times more processing capacity than application tier servers. Virtual networks are also defined between all VMs in the same application so that any VMs can transfer data to any other VMs in the same application. Required bandwidth for the critical application is set to the half of physical link bandwidth, while the normal application is set to be a fourth of the physical bandwidth to differentiate the priority.

We also generate workloads for the second scenario from three-tier application model [33] based on Wikipedia traces available from *Page view statistics for Wikimedia projects*. Every application receives approximately between 80,000 and 140,000 web requests generated from traces in a different language, each of which consists of processing jobs in VM servers and network transmissions.

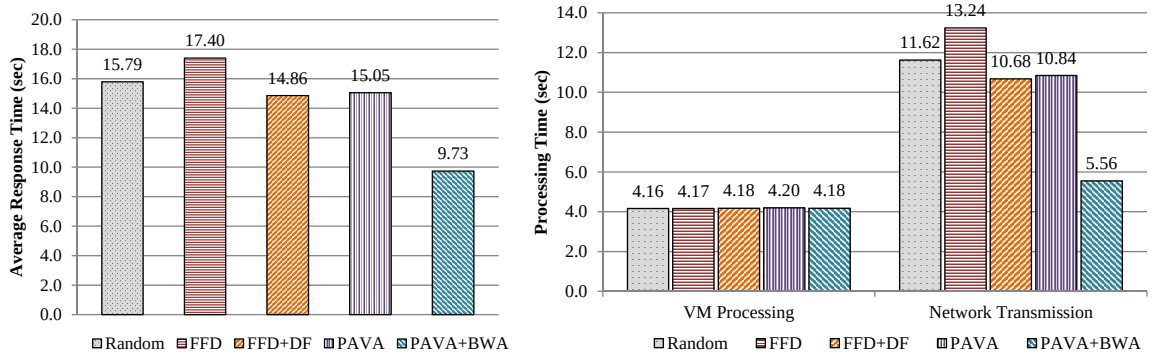
For both scenarios, we measure the response time of both critical and normal applications, the QoS violation rate of the critical application, and the power consumption of the data center.

### 5.5.2 Analysis of Response Time

At first, we evaluate the performance of the proposed algorithm by measuring the average response time of the critical application, and VM processing and network transmission time in detail with each algorithm. Note that QoS violation is not considered for calculating the averages in this subsection.

Figure 5.3 shows the results in Scenario 1 where the data center network is constantly overloaded by other applications. The average response time (Figure 5.3a) is significantly reduced by 38.4% in PAVA+BWA (both PAVA and BWA algorithms applied) compared to the Random algorithm, mainly resulting from 52.1% reduction in network transmission time (Figure 5.3b). VM processing time remains same regardless of algorithm combination which shows that VMs acquire enough processing resources.

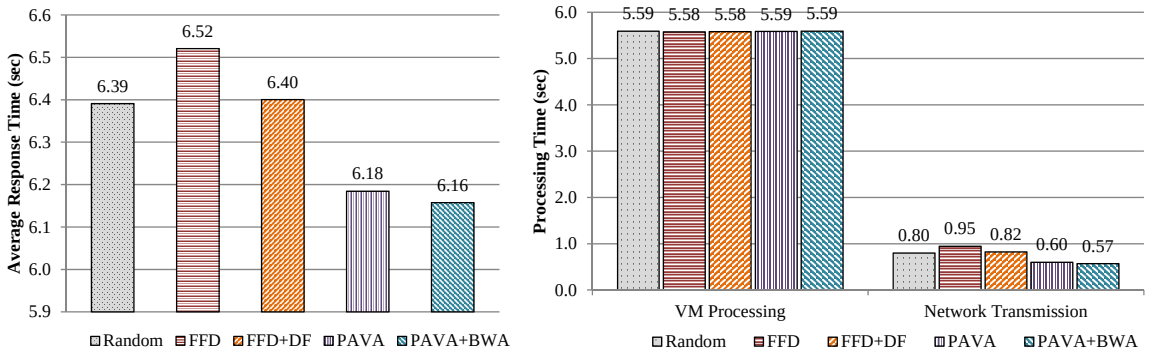
For FFD, the average response time is 10.2% increased compared to Random method




(a) Average response time of the critical application.

(b) VM processing and network transmission time of the critical application.

Figure 5.3: Performance matrices of the critical application in Scenario 1 (synthetic workload).




(a) Average response time of the critical application.

(b) VM processing and network transmission time of the critical application.

Figure 5.4: Performance matrices of the critical application in Scenario 2 (Wikipedia workload).

due to the increased network transmission time. Since FFD consolidates more VMs into a smaller number of hosts without consideration of their connectivity, network transmissions within the critical application are significantly interfered by other applications placed on the same host. Similarly, applying PAVA without network bandwidth allocation cannot improve overall performance substantially due to the consolidation of VMs into shared hosts, although the average response time is still shorter than the one from FFD.

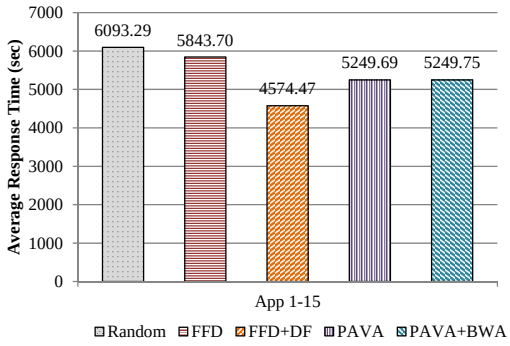
With the implementation of DF in addition to FFD, the average network transmission time is reduced to 10.68 seconds from 13.24 of FFD. Although dynamic flow scheduling

can find a less crowded path, it is ineffective in this scenario where all the alternate paths are busy. On the other hand, BWA provides the best result in network transmission time reduced to almost half of all the other methods. This shows that our bandwidth allocation method can significantly improve the critical application's network performance in the overloaded network environment.

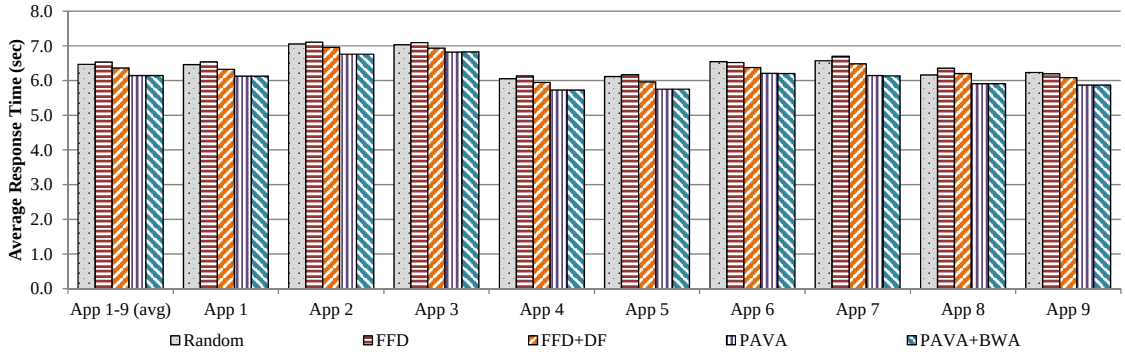
Figure 5.4 depicts the results from Scenario 2 where more complex applications and workloads are submitted to a large-scale cloud data center. In this scenario, the average response time is reduced by 3.3% in PAVA, and BWA is not shown as effective as the previous scenario. Unlike Scenario 1, the network is not frequently overloaded in the data center, which limits the effectiveness of BWA. On the other hand, PAVA becomes more effective on account of the proximity of VMs. As the VMs of the same application have been placed closely with PAVA, the network transmission time between them is reduced by 25% from 0.80 seconds in Random to 0.60 seconds in PAVA. The critical application's network workloads pass through only low-level switches (edge and/or aggregation switches) as the VMs are placed under the same edge network or the same pod, and thus not interfered by other traffics.

Similar to the previous scenario, FFD increases the average response time due to the VM consolidation to shared hosts. Implementation of DF also reduces the network transmission time, which makes the average response time of FFD+DF become similar to the Random method. VM processing times are almost same no matter which algorithm is being used. In short, the proposed algorithm (PAVA+BWA) improves the response time of the critical application by 34.5% for Scenario 1 and 3.8% for Scenario 2 compared to the state-of-the-art baseline (FFD+DF).

Additionally, we measure the average response time of normal (lower-priority) applications to see the effect of our algorithm on the other normal applications. Figure 5.5 shows the measured response time of normal applications in both scenarios. Compared to the Random algorithm, PAVA and PAVA+BWA actually result in improving the performance of lower-priority applications by reducing the average response time by 13.8% and 4.9% respectively in Scenario 1 and 2. The baseline algorithm FFD+DF also reduces the response time of lower-priority applications. In short, our algorithm maintains or even improves the performance of lower-priority applications, while improving the per-




(a) Scenario 1 (synthetic workload).



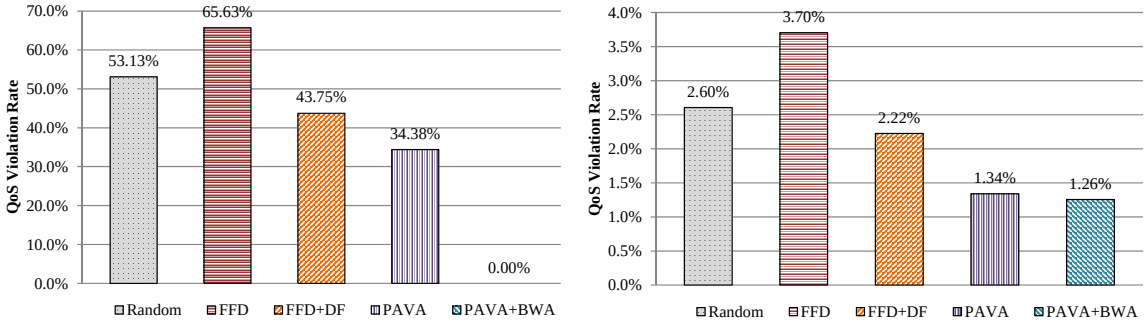

(b) Scenario 2 (Wikipedia workload).

Figure 5.5: Average response time of normal (lower-priority) applications.

formance of a critical application.

### 5.5.3 Analysis of QoS violation rate

QoS violation rate is calculated by comparing the response time from ERA algorithm with the one from other algorithms. We assume that the expected response time can be fully achieved in ERA because it allocates dedicated servers with enough resource for every VMs required in the application. We compare the response time of each workload and count the QoS violated workload if the response time exceeds the one from ERA. Equation 5.1 shows the calculation of QoS violation rate ($r_v$) from workloads set ($W$), where $t_X$ and $t_{ERA}$ denote the response time of a workload ($w_v$) measured from the designated algorithm and ERA respectively. It counts the number of workloads whose response time from the designated algorithm is exceeding the response time from ERA and divides by the total number of workloads.




(a) Scenario 1 (synthetic workload).

(b) Scenario 2 (Wikipedia workload).

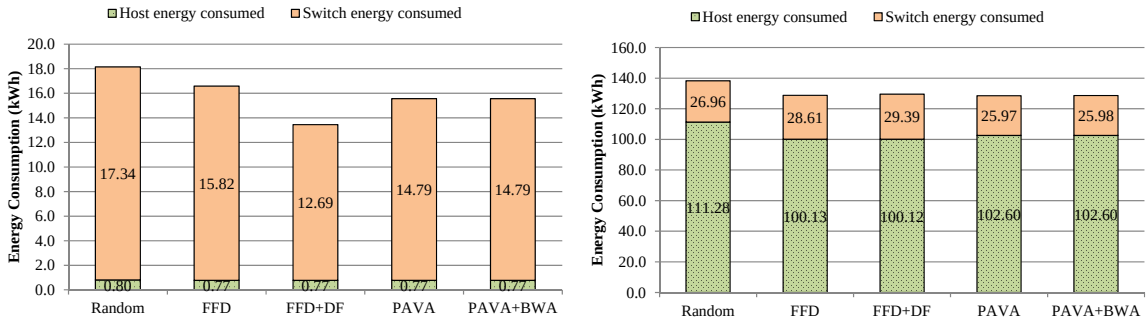
Figure 5.6: QoS violation rate of critical application workloads.

$$r_v = \frac{|\{w_v \in W | t_X(w_v) > t_{ERA}(w_v)\}|}{|W|} \tag{5.1}$$

Average QoS violation rate of the critical application is shown in Figure 5.6. In Scenario 1, PAVA results in 34.38% QoS violation whereas PAVA+BWA has no violation at all (see Figure 5.6a). As we discussed in the previous subsection, BWA is more effective in overloaded networks where other tenants generate heavy traffic loads. The baseline (FFD+DF) also reduce the QoS violation rate from Random's 53.13% to 43.75% but not as significant as BWA.

Similar results can be found in Scenario 2 (see Figure 5.6b) where PAVA and PAVA+BWA show the lowest QoS violation rate reaching 1.34% and 1.26% respectively. Interestingly, overall violation rate is much lower, ranging between 1.26% and 3.70% in Scenario 2 compared to between 0% and 65.36% of Scenario 1. This is due to the significant degradation of the network performance in Scenario 1 where network overload by other applications interferes the application. Although the QoS violation rate in Scenario 2 is not as high as in Scenario 1, the impact of our algorithm is still significant to improve the violation rate by 51.5% reducing from 2.60% to 1.26%. It is a crucial improvement for critical applications that should guarantee the QoS requirement. Although BWA is not as beneficial as Scenario 1, it can still reduce the violation rate by 0.08%p compared to PAVA alone.

Compared to the state-of-the-art baseline, our proposed algorithm combination, PAVA+BWA, can reduce QoS violation rate from 43.75% to 0% for heavy network traffic scenario and from 2.22% to 1.26% (reduction by 43.2%) for large-scale complex application scenario.




(a) Scenario 1 (synthetic workload). (b) Scenario 2 (Wikipedia workload).

Figure 5.7: Detailed power consumption of hosts and switches in a data center.

### 5.5.4 Analysis of energy consumption

Energy consumption is evaluated to find the influence of the proposed algorithm to the operational cost of a cloud data center. We measured the utilization of hosts and switches over time and used power model of hosts [95] and switches [125] respectively to calculate the overall power consumption, using the same model and method exploited in Chapter 4. Unused hosts and switches are assumed to be in an idle mode to save energy, and the power consumption of active hosts and switches is calculated based on the utilization of a host and the active ports of a switch.

Figure 5.7 shows the measured energy consumption of the entire data center and the detailed power consumption in hosts and switches for both scenarios. In Scenario 1 (see Figure 5.7a) both PAVA and PAVA+BWA save 14.2% of a total data center energy usage compared to the Random algorithm, whereas FFD and FFD+DF save 8.6% and 25.9% of power cost respectively. The difference mainly comes from switches, because workloads in Scenario 1 consist of marginally huge network traffics combined with a small computation load on VMs.

In Scenario 2, PAVA and PAVA+BWA consume the least amount of energy among all algorithms. For host energy consumption, all the four algorithm combinations (FFD, FFD+DF, PAVA, and PAVA+BWA) consume less energy compared to Random, since both PAVA and FFD consolidate VMs into the smaller number of hosts and turn off many unused hosts. For switches, however, FFD (28.61 kWh) and FFD+DF (29.39 kWh) consume more amount of energy compared to Random (26.96 kWh), while the consumption in PAVA (25.97 kWh) and PAVA+BWA (25.98 kWh) is lower than the Random. As those

VMs in the same application group are closely placed with PAVA mostly within the same edge network or within the same pod, the network traffics are consequently consolidated passing through fewer switches. Thus, the energy consumption is lowered resulting from the decreased number of active switches.

Nevertheless, the results show that the proposed algorithm at least will not increase the power consumption of the data center. In fact, it can help to reduce the operational cost by consolidating VMs into a smaller number of hosts while providing the required QoS for a critical application. In Wikipedia workload, the energy consumption is even reduced compared to the state-of-the-art baseline.

### 5.5.5 Analysis of algorithm complexity

We analyze the time complexity of the proposed algorithm. Firstly, the greedy bin-packing FFD algorithm (Algorithm 7) takes $O(|H| \log |H|)$ time to place one VM in a data center with $|H|$ number of available hosts. In order to place $|VM|$ number of VMs, the algorithm takes $O(|VM| \cdot |H| \log |H|)$ which is feasible for online dynamic VM placement.

PAVA is based on FFD algorithm with an extra computation for critical applications. For each VM in critical applications, PAVA needs an additional sorting time, $O(|H| \log |H|)$, to find a closely connected host from the previously placed VMs. Thus, given VMs in critical applications ($VM_c \in VM$), the overall complexity of PAVA including the additional computation for critical VMs along with the basic FFD is:

$$\begin{aligned} O(|VM_c| \cdot |H| \log |H|) + O(|VM| \cdot |H| \log |H|) \\ = O(|VM| \cdot |H| \log |H|) \end{aligned}$$

which is same as the time complexity of FFD algorithm.

The time complexity of BWA algorithm is $O(|S| \cdot |F_c|)$ where $|F_c|$ number of flows for critical applications are placed in a data center network consisting of $|S|$ number of switches. This is also a small addition compared to dynamic scheduling algorithm where flows are periodically re-routed by running routing algorithms to find the best route. However, the overhead of BWA may occur at the time of packet forwarding in

switches because of extra forwarding rules and queues configured for critical applications. Although we could not simulate this overhead in our evaluation environment, the extra overhead is negligible when the proportion of higher-priority flows is significantly smaller than the data center. The number of switches installing the extra queues will be minimized especially with PAVA where the priority VMs are placed in close hosts, thus network traffics are transmitted through the minimal number of network hops.

## 5.6 Summary

In this chapter, we presented a priority-based VM allocation and network traffic management scheme with bandwidth allocation and dynamic flow pathing mechanism. The algorithms are evaluated in a simulation environment with a large-scale fat-tree topology and multiple applications with a different priority. The results show that the proposed priority-aware VM allocation method actually places the critical application into the closer hosts so that it reduced both the energy consumption and the average response time for the critical application. The bandwidth allocation method is specifically effective in the overloaded network scenario where the higher-priority traffic is interfered by other applications. Our algorithm is outperformed the state-of-the-art approaches.

# Chapter 6

# Prototype System of Integrated Control Platform

*This chapter proposes SDCon, a practical platform developed on OpenStack and OpenDaylight to provide integrated manageability for computing and networking 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. We also propose a network topology aware VM placement algorithm for heterogeneous resource configuration (TOPO-Het) that consolidates 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 baseline. 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 managing both resources efficiently.*

## 6.1 Introduction

MANY management platforms have been introduced and developed for cloud computing and SDN, such as OpenStack, VMWare, and Xen Cloud Platform for clouds and OpenDaylight, Floodlight, NOX, ONOS, and Ryu for SDN. Although these individual platforms are matured in cloud computing and SDN individually, no software platform exists in practice for integration of clouds and SDN to jointly control both networking and computing devices for clouds. For example, OpenStack adopts

This chapter is derived from: Jungmin Son and Rajkumar Buyya, "SDCon: Integrated Control Platform for Software-Defined Clouds," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (accepted, in press).

OpenVSwitch, a software virtual switch compatible with SDN, in its own networking module to provide network virtualization of VMs, but OpenVSwitch is created only in compute nodes to establish virtual tunnels with other nodes. OpenStack does not provide any feature to control the network fabric of the cloud. 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 integrational controllability of both compute nodes and network switches.

In this chapter, we propose an integrated control platform, named *SDCon* (Software-Defined Clouds Controller), which can jointly manage both computing and networking resources in the real world for Software-Defined Clouds (SDC). 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 driver modules, but for simplicity of development, we adopted OpenStack and OpenDaylight as underlying software. To the best of our knowledge, this is the first attempt to integrate cloud's computing fabric and SDN's network fabric controllability into a combined SDC controller. In addition, we propose a topology-aware resource allocation algorithm in heterogeneous configuration.

The key **contributions** of this chapter are:

- the design of a practical SDC controller that integrates the controllability of network switches with the management of cloud resources;
- the implementation of the integrated SDC control system on a testbed with 8 compute nodes connected through 2-pod modified fat-tree network topology;
- a joint resource provisioning algorithm for heterogeneous resource configuration based on network topology;
- an evaluation of the joint resource provisioning algorithm on the testbed configured with the proposed SDC control platform.

This chapter is organized as follows: Section 6.2 provides relevant literature studied and developed for SDN integration platform for cloud computing. In Section 6.3, 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 6.4. The heterogeneity and network topology aware VM placement algorithm is proposed in Section 6.5 in addition to baseline algorithms. Section 6.6 provides the validation result of SDCon, and Section 6.7 shows the evaluation result of SDCon and the proposed algorithm by comparing with baselines. Finally, the chapter is summarized in Section 6.8.

## 6.2 Related Work

Increasing number of studies have investigated joint provisioning of networking and computing resource in clouds [27, 59, 60, 132, 135], in which experiments have been conducted either in simulation environment [60], or on their in-house customized empirical system [27, 59, 132, 135].

We explained SDN-enabled cloud computing simulator, CloudSimSDN, to enable large-scale experiment of SDN functionality in cloud computing in Chapter 3. 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 gaps between the simulation and real implementation.

Mininet [68] has gained a great popularity for SDN emulation to experiment practical OpenFlow controllers. Any SDN controllers supporting OpenFlow protocol can be used 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* 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.

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 [27].

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 test bed 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 [2, 72, 73]. This 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.

OpenStackEmu is proposed to integrate OpenStack with a large-scale network emulator named CORE (Common Open Research Emulator) [14]. The proposed system combines OpenStack platform with a network emulator to perform evaluations considering both cloud and networking. All network switches are 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.

Related work in the context of algorithms for joint resource provisioning will be discussed later in Section 6.5.

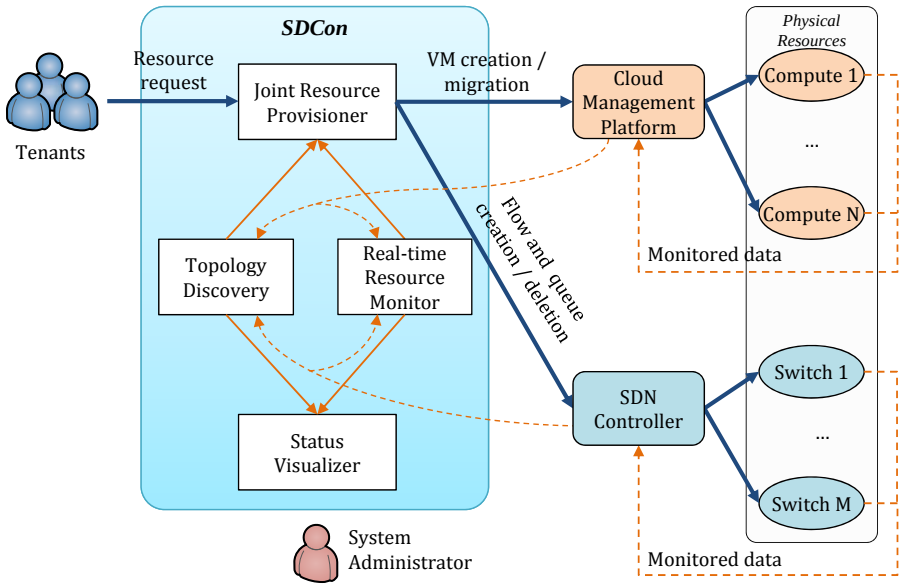



Figure 6.1: Design principle and control flows.

## 6.3 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. We explain the design principle and implementation details of SDCon in this section.

The conceptual design and the control flows between components are shown in Figure 6.1. 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 platform. 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 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, 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, networking requirements can be also specified in the resource request, such as requested bandwidth between VMs. In addition to allocating resource for tenants, Joint Resource Provisioner is also possible to optimize both computing and networking resources based on the optimization policy provided by the system administrator. 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. System administrators 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 specs 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. Both topology and monitored data provide all information necessary for resource provisioning and optimization.

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 this system in detail.

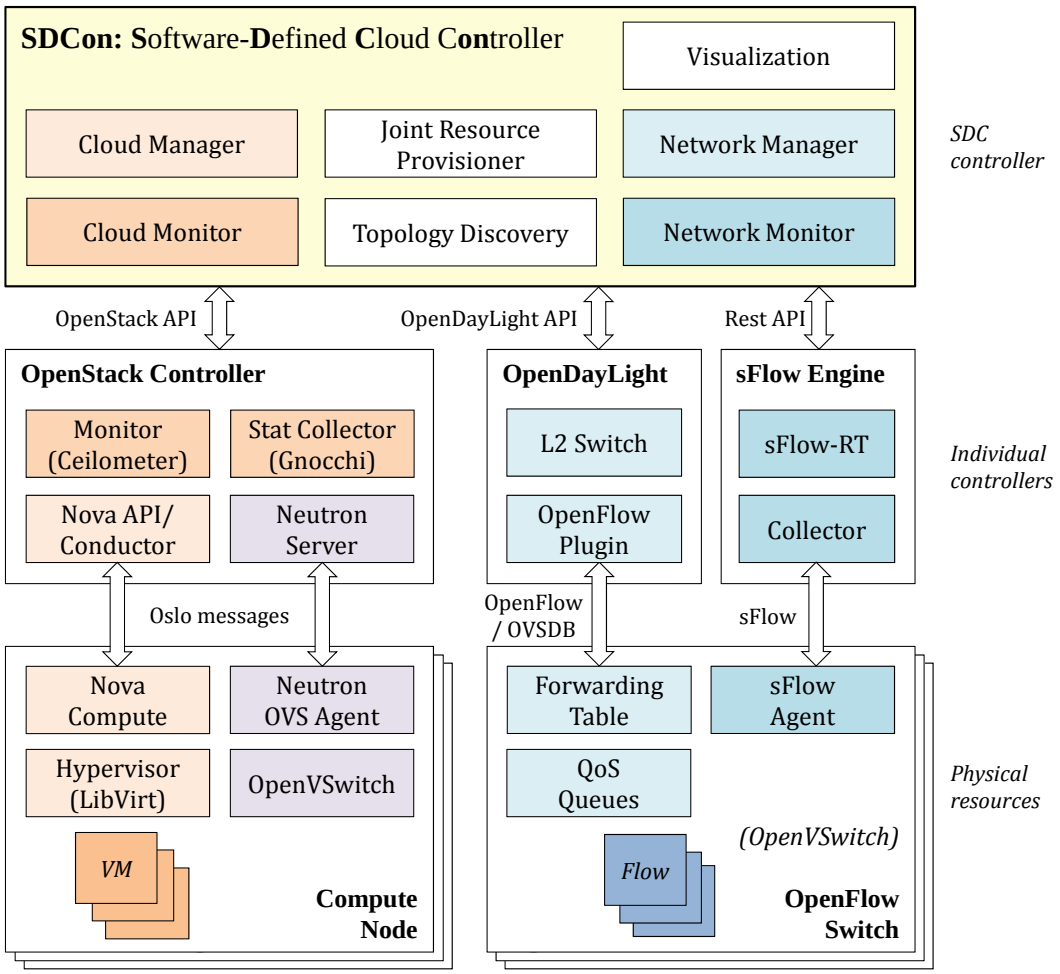



Figure 6.2: System architecture of SDCon and underlying software components.

## 6.4 System Implementation

We implemented the proposed system by utilizing OpenStack cloud management platform and OpenDaylight SDN controller to control computing and networking resources respectively. Figure 6.2 shows the implemented architecture of SDCon platform. The system is implemented with Python, on top of OpenStack Python SDK[1], Gnocchi Python API[2], OpenDaylight OpenFlow plugin[3] and OVSDB plugin[4], and sFlow-RT REST API[5].

[1] http://developer.openstack.org/sdks/python/openstacksdk/users/index.html
[2] http://gnocchi.xyz/gnocchiclient/api.html
[3] http://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:End_to_End_Flows
[4] http://docs.opendaylight.org/en/stable-carbon/user-guide/ovsdb-user-guide.html
[5] http://www.sflow-rt.com/reference.php

OpenStack platform is exploited to manage to compute nodes. Multiple compute nodes are running OpenStack Nova-compute component to provide a bridge between its 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, Neutron is in charge of managing virtual networks for VMs. Neutron-OVS-Agent running in each compute node is creating and deleting virtual tunnels between compute nodes by modifying OpenVSwitch, 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 Rest API.

For networking resources, OpenDaylight controls OpenFlow switches that connects 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 bandwidth allocation. Due to the OpenFlow's limited support for queues, queue management is provided through OpenDaylight's OpenVSwitch plug-in which can create and delete Hierarchical Token Bucket (HTB) queues 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 in the controller which gathers the data and provide the measurements through Rest API.

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

- allocating resources for VMs and flows with the information of an application deploying on these VMs and flows;

- monitoring both hosts and networking devices for their utilization, availability, and capacity with the knowledge of network topology;

- dynamic re-provisioning of both computing and networking resources based on the real-time monitored matrices;
- visualizing the monitored measurements and the operational status of SDC.

The following subsections provide the detailed explanation of each component in SDCon, data flows between components and sequence diagram of VM allocation with SDCon.

### 6.4.1 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 node to host the VM. 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 provides all other functions related to VMs and compute nodes, such as getting the IP address of a VM, VM types (Flavor), VM 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.

### 6.4.2 Cloud Monitor

Cloud Monitor is to retrieve a real-time measurement of computing resources monitored at compute nodes and VMs. We use OpenStack Ceilometer and Gnocchi components to measure and collect the monitored data. Ceilometer installed on each compute node measures CPU and memory utilization of all VMs and the host itself, and sends to Gnocchi component that collects and aggregates the data from every compute node. Our Cloud Monitor uses Gnocchi Python API to retrieve the collected data, such as CPU utilization of both VMs and compute nodes. This information can be leveraged in many studies based on the real-time resource utilization. It is also utilized by Power Consumption Estimator to calculate the estimated power consumption of compute nodes.

### 6.4.3 Network Manager

In SDCon, OpenDaylight SDN Controller is managed by Network Manager module to push OpenFlow rule 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 to alter the path for performance improvement or power efficiency. We also implement a default path routing module supporting the load-balancing routing in fat-tree topology as suggested in the original paper [4]. Because the default L2 switch module in OpenDaylight only supports STP for multi-path topology, we cannot utilize the benefit of multiple paths with the default module. Thus, we need to implement our own routing module to set the default path between compute nodes based on the source address of the host. Our default path module installs flow rules on each switch to forward the packet to different ports based on the source host if there are multiple paths. With the default path module, packets are forwarded to the pre-configured path based on the source host 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. For a priority flow which needs a higher priority over other traffics, we can allocate a specific bandwidth by configuring 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 (Hierarchy Token Bucket). 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 flows share the same link. 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 the evaluation section.

### 6.4.4 Network Monitor

Similar to Cloud Monitor, Network Monitor pulls a real-time network status from switches and compute nodes. We leverage sFlow protocol that sends sampled packets from sFlow agents in switches to the collector in the controller. 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, sFlow provides more detailed information such as the source and destination address of the flow and encapsulated packets for tunneling protocols.

### 6.4.5 Topology Discovery

Network topology information is retrieved from OpenDaylight through Network Manager. With the support of host tracker in OpenDaylight, SDCon can acknowledge host and switch devices on the network along with ports and links. Topology Discovery module generates a graph structure internally to store the topology information. In addition to the network topology, 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 module 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 at the start of the platform and updated only when there is an update on the topology.

### 6.4.6 Joint Resource Provisioner

The main control logic of SDCon resides in Joint Resource Provisioner module. When VM requests are submitted by 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 provided for VM provisioning, as well as a network flow scheduling algorithm for network provisioning if the default path set by Network Manager is not sufficient.

Once the decision is made at the control logic, Joint Resource Provisioner exploits Cloud Manager to create VMs on the selected hosts and Network Manager to push new flow entries. For bandwidth allocation, QoS rules and queues can be created in switches by modifying OpenVSwitch through Network Manager. When a migration policy is specified, it can also schedule the migration process by sending migration command to OpenStack through Cloud Manager.

### 6.4.7 Power Consumption Estimator

For energy-related experiments, we additionally implement Power Consumption Estimator that calculates the power consumption of the data center from 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. By default, we implement the model derived from Pelly et al. [95] for compute nodes and from Wang et al. [125] for switches. Please note that the estimator is an alternative of commercialized PDUs due to the limited budget and resources. Although hardware PDU equipment can measure and manage the power consumption of hosts and switches more accurately, it is not feasible to procure such equipment at this time.

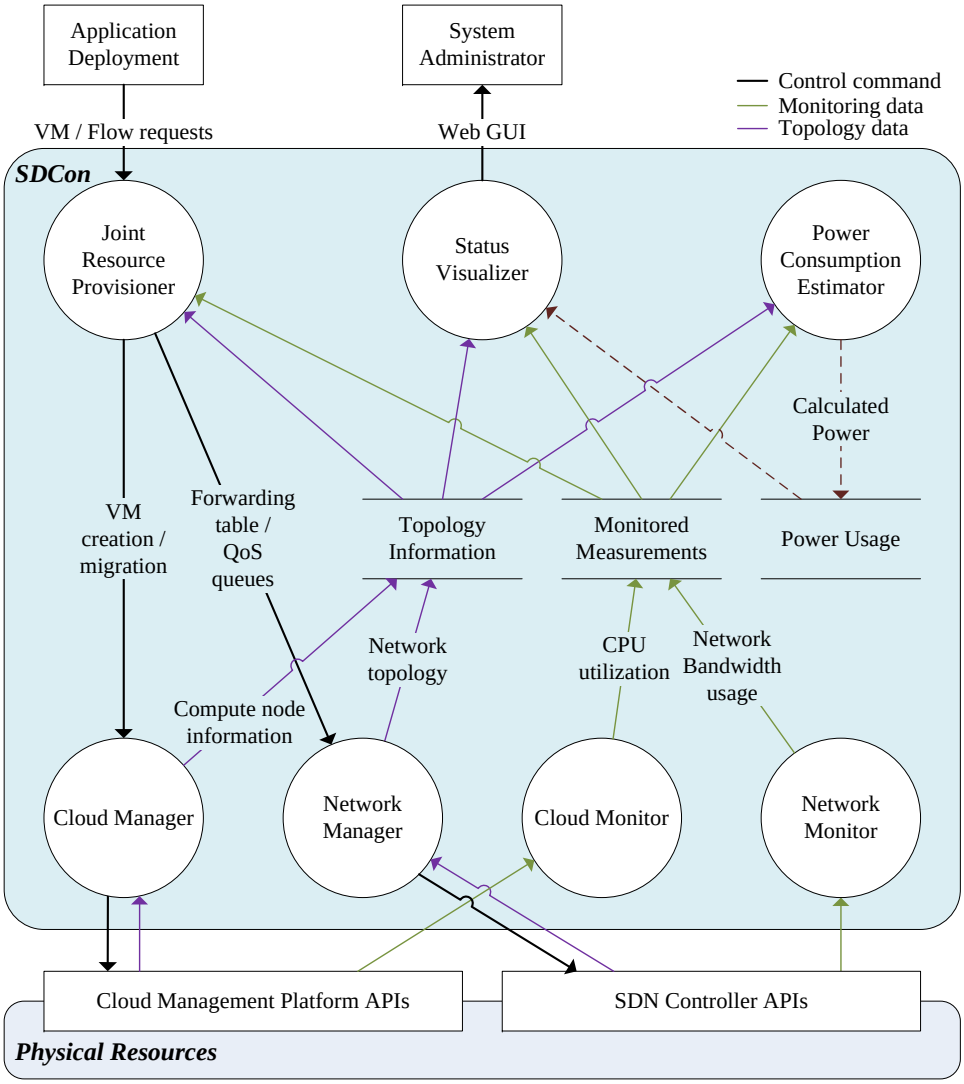



Figure 6.3: Data flow diagram of SDCon components.

### 6.4.8 Status Visualizer

Visualization module 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 and compute utilization with different colors. We use D3 JavaScript library[6] to visualize the data onto web interface. By refreshing the page every second, the Visualizer continuously updates the monitored measurements in real time.

[6] https://d3js.org/

### 6.4.9 Data Flow and Sequence Diagram

sFigure 6.3 depicts the detailed 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 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. All the information is visualized with Status Visualizer through a web application.

To provide an in-depth understanding on the working process of SDCon, we also depict sequence diagram of deploying VMs and flows with SDcon (see Figure 6.4). The diagram shows the process to get VM and host information from OpenStack API and network topology information from OpenDaylight. 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 provide the procedure of updating a flow path based on the real-time bandwidth usage of each link, as an example of SDCon usage. 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.

## 6.5 Joint Resource Provisioning in Heterogeneous Clouds

In this section, we propose a joint computing and networking optimization algorithm for heterogeneous resource configuration. 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 [27, 135]. This is an effective assumption in large-scale public cloud considering the identical hard-

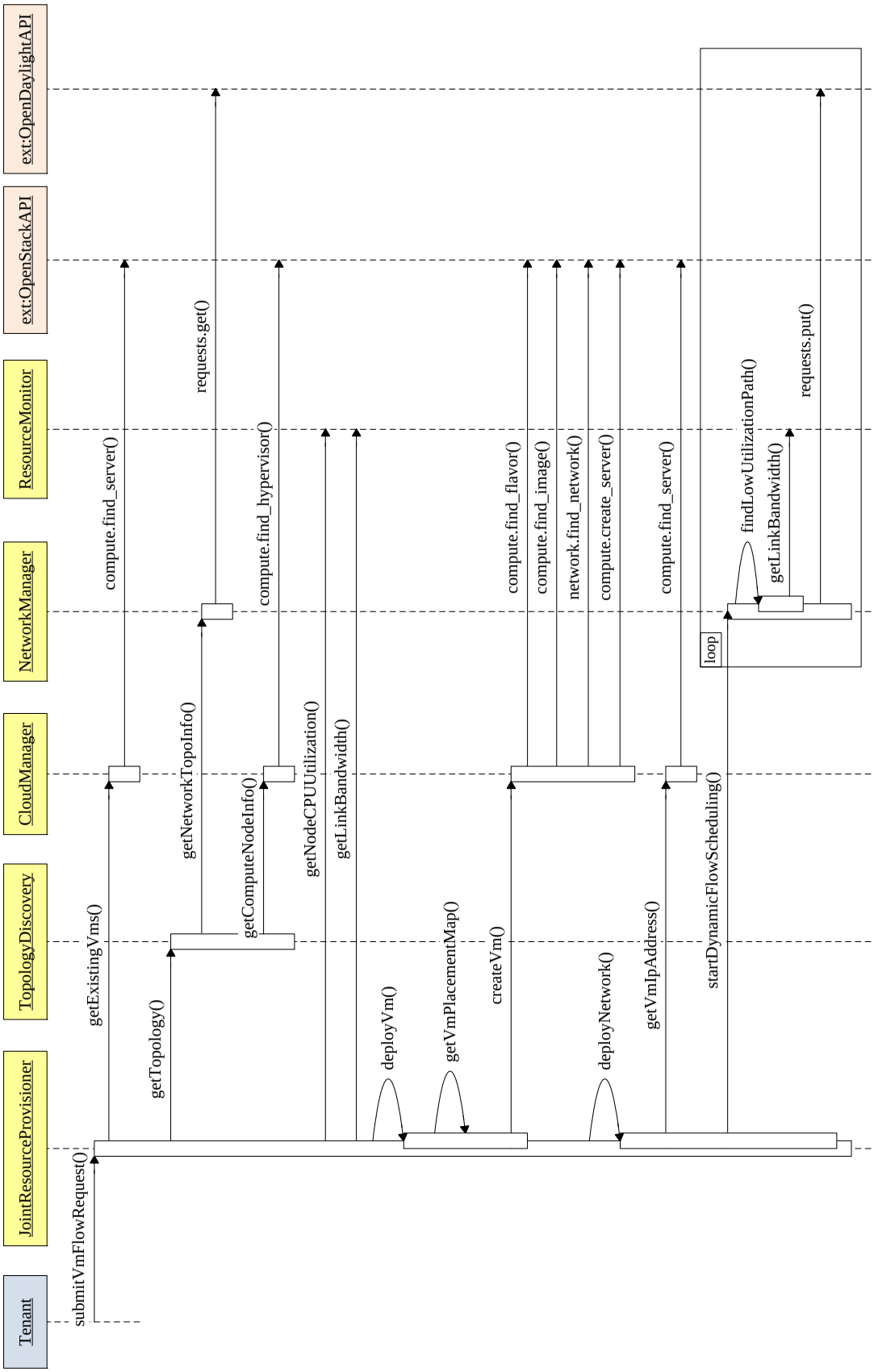



Figure 6.4: Sequence diagram to deploy VMs and flows with SDCon.

ware within the same rack which is procured at the same time. However, as the data center grows and needs to upgrade or purchase new hardware, the configuration 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 the power consumption level of each host. When VMs can be consolidated into the smaller number of physical hosts, the unused hosts can be powered off to save energy consumption of the data center. In a homogeneous model, reducing the number of turned-on hosts leads to the reduction of energy consumption of the entire data center. However, in heterogeneous configuration, we have to consider the different capacity and power consumption level of each physical machine. If VMs are consolidated into the less power-efficient host which consumes more power than the total of multiple energy-efficient hosts, consolidation can actually increase the power consumption of the data center. Thus, the power consumption level of different host types must be taken into an account for power optimization in heterogeneous clouds.

Many studies considering resource heterogeneity in clouds have focused on the level of brokers for various providers or for 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 [115]. Recently, a renewable-aware load balancing algorithm across geologically distributed data centers was proposed to increase the sustainability of data centers [114]. This algorithm selects a data center operated by more renewable power sources (e.g., data center sourced by a solar power plant in a clear day) to reduce the carbon footprint and places more VMs on those data centers. Also, some researchers studied the resource optimization within a data center considering heterogeneity to reduce the energy consumption of the data center [12]. The on-line deterministic algorithm consolidates VMs dynamically into the smaller number of compute nodes and switches

off the idle nodes to save electricity. 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, and none has studied the joint optimization considering both computing and networking resources in heterogeneous infrastructure in a data center.

Also, a VM management method considering network topology was proposed by Cziva et al. [27] 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.

### 6.5.1 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 [61]. In this chapter, 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 hosts 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 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 bandwidth. 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 can not only provide more bandwidth for VMs but also reduce the amount of traffic in network infrastructure. However, if VMs with no inter-VM traffics are placed in the same node, it consumes more networking resources and reduces per-VM bandwidth because more VMs share the same interface in the node.

Algorithm 9 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 hosts which can collectively provide the requested resources for VMs. In the beginning, VMs are grouped based on the network connectivity. If a VM needs to communicate with another VM, they are put into the same group. VMs in each group are sorted in the order of required resource from high to low to make sure a large VM is placed before smaller ones.

The algorithm first 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, the connected nodes under the same edge switch, or the nodes within the same pod. It also finds the candidate host group, sorted by the available bandwidth calculated from the number of VMs running in the entire host group, with the information of network topology. If there are more VMs running in 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 in this step where 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, a VM in the group is tried to place in a host within the higher priority host group. Note that the

**Algorithm 9** 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 ← 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 ← empty list for available host groups;
9:     VM_conn ← topo.getConnectedVMs(vmg);
10:    if VM_conn is not empty then
11:        H_host ← topo.findHostRunningVMs(VM_conn);
12:        H_edge ← topo.getHostGroupSameEdge(H_host);
13:        H_pod ← topo.getHostGroupSamePod(H_host);
14:        HG.append(H_host, H_edge, H_pod);
15:    end if
16:    HG_host ← topo.findAvailableHost(vmg);
17:    sort(HG_host, key=hg.capacity, order=desc);
18:    HG_edge ← topo.findAvailableHostGroupEdge(vmg);
19:    sort(HG_edge, key=hg.capacity, order=desc);
20:    HG_pod ← 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 ← 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
```

host list within a group is sorted by the number of free CPU cores before placing VM, 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.

### 6.5.2 Baseline algorithms

Our heterogeneity-aware resource allocation algorithm is evaluated in comparison 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 [11, 121]. VMs are sorted in descending order of the size of the required resource, and compute nodes are sorted in ascending order of available resources. Since it chooses the most full node first, the algorithm consolidates VMs into a smaller number of compute nodes which can benefit the power consumption and network traffic consolidation. Our algorithm (TOPO-Het) also adopts the core idea of FFD in order to consolidate VMs in addition to the consideration of network topology and resource heterogeneity.

Further, we implement two network management methods, BWA and DF, in combination with FFD to show the effectiveness of SDCon. In BWA (Bandwidth Allocation), SDCon allocates the requested network bandwidth for traffic flows between VMs using the aforementioned method in Section 6.6.2. 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 [32, 119, 134]. 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, SDCon selects the less congested path and updates forwarding tables on switches along the selected path through SDN controller.

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.

## 6.6 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

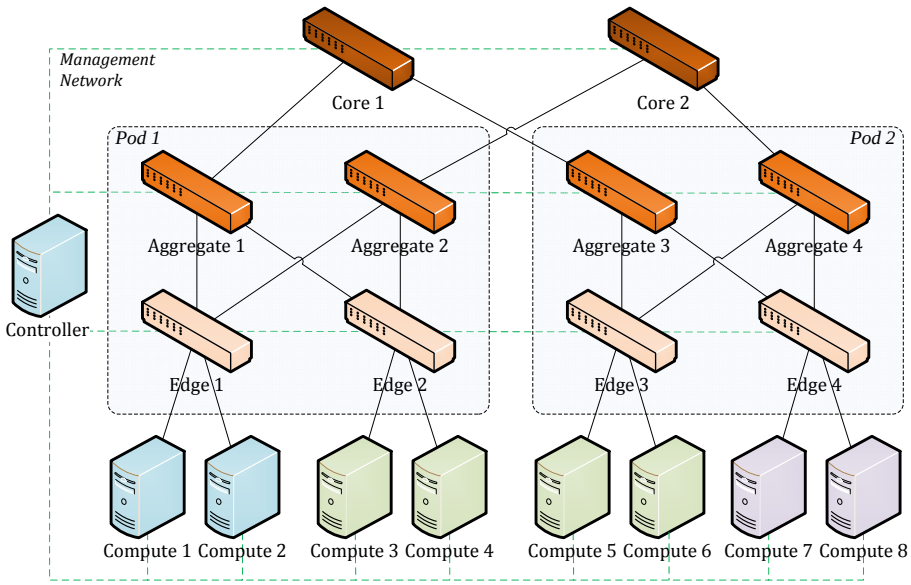



Figure 6.5: Testbed configuration.

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 which we will discuss in Section 6.7.

### 6.6.1 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 and 10 OpenFlow switches. Figure 6.5 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 propose [4]. 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 USB 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 [116] that Raspberry-Pi was used for compute nodes in clouds. In our testbed, we

Table 6.1: Hardware specification of controller and compute nodes.

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

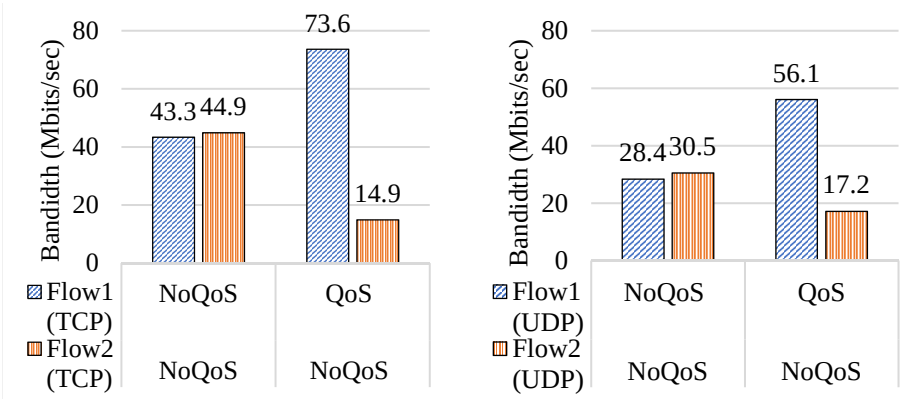
use Raspberry-Pi to build switches, not compute nodes, as we have enough servers for compute nodes whereas we cannot procure OpenFlow switches.

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. Note that we configured a separate management network for controlling and API communications between switches, compute nodes, and the controller.

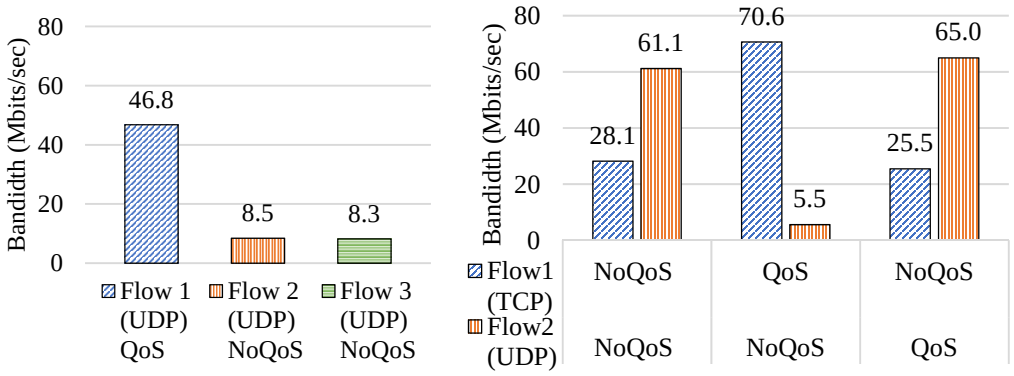
Each compute nodes are running OpenStack Nova-Compute component to provide computing resources to VMs as well as OpenStack Neutron-OVS Agent for virtual network configuration for the hosted VMs. OpenStack components in compute nodes are connected to the OpenStack server running in the controller node 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 6.1 shows the different hardware configuration of each computer.

### 6.6.2 Bandwidth Allocation with QoS Settings

As described in Section 6.4, 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 iperf3 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 iperf3 simultaneously on two or three different VMs. Also, various combinations of TCP and




(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.

Figure 6.6: Bandwidth measurement of multiple flows sharing the same network resource.

UDP protocols are used to compare the impact of our bandwidth allocation mechanism. 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 specified the minimum bandwidth of 70 Mbps for QoS flows and 10 Mbps for other flows.

Figure 6.6 shows the measured bandwidth of different flows sharing the same network path. When iperf3 was used in TCP mode (Figure 6.6a), 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. Figure 6.6b shows the measured bandwidth for two UDP flows. In UDP mode, we run iperf3 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/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 (Figure 6.6c), 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. 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 adjusting the transmission rate to avoid network congestion. 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 QoS queues in switches forwarded packets of Flow 1 at 70 Mbps as specified in the configuration, whereas Flow 2's packets are 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. 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.

## 6.7 Performance Evaluation

We evaluate the proposed system and the algorithm with a real-world application and workload: Wikipedia application. Wikipedia publishes its web application, named MediaWiki, and all database dump files online[7] so that anyone can replicate Wikipedia ap-

[7] `https://dumps.wikimedia.org/`

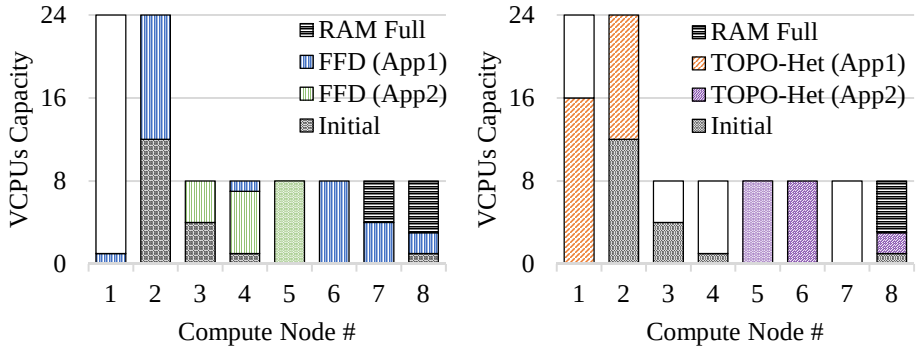
plication in their own environment with various configuration. Testing with Wikipedia application has been more straightforward with the introduction of WikiBench [117], a software to deploy Wikipedia database dumps and send web requests to MediaWiki servers based on the historical traffic trace. These experiments are also conducted on the same testbed described in the previous section, comparing our proposed algorithm with baseline algorithms (discussed in Section 6.5.2).

### 6.7.1 Application Settings and Workloads

For empirical evaluations, we deploy Wikipedia application consisting of multiple VMs across our testbed using SDCon to measure the response time. Three-layer web application architecture is exploited to configure the Wikipedia: client, web, and database server. 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 is configured with Apache2 server to host MediaWiki application written in PHP language. Database VM runs MySQL which retrieves Wikipedia database rebuilt by the dump.

We set two applications with a different number of VMs to check the impact of the proposed algorithm on the scale of the application. Application 1 is deployed with 4 client VMs, 4 web server VMs, and 1 database VM, whereas Application 2 consists of two clients, two web servers, and one database VM. Although they are separate applications without any network traffic, we conduct the experiment on both applications simultaneous to see the impact of different applications on the network resources of a data center. Each VM is configured with a predefined VM Type shown in Table 6.2. In order to evaluate the case when some VMs of the application are already deployed, we placed the database VM of App 2 in Compute 5 node before starting the experiment. Also, VM images are prepared with proper software settings to deploy the application quickly and easily.

In addition to VM configurations, network flow settings are defined for VM traffics (Table 6.3). 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




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

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

Figure 6.7: Used and available CPU cores of compute nodes after VM placement in different algorithm.

physical link in our testbed is 100 Mbps, a web server still can use the entire bandwidth when it needs to send data to both database and client.

### 6.7.2 Analysis of VM Allocation Decision

We submit the VM configurations and flow definitions for two applications to SDCon and runs the VM allocation algorithms. Figure 6.7 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 host some VMs. 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, for both App 1 and App 2, VMs are scattered across the whole 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 in different network

Table 6.2: VM types for Wikipedia application deployment.

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

pods. Note that Compute 7 and 8 cannot host anymore VMs because of RAM limitation, even though there are free CPUs.

In comparison, TOPO-Het considers network topology, and the VMs for the same application are consolidated within the same pod or edge hosts. 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, VMs for App 1 are all 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.

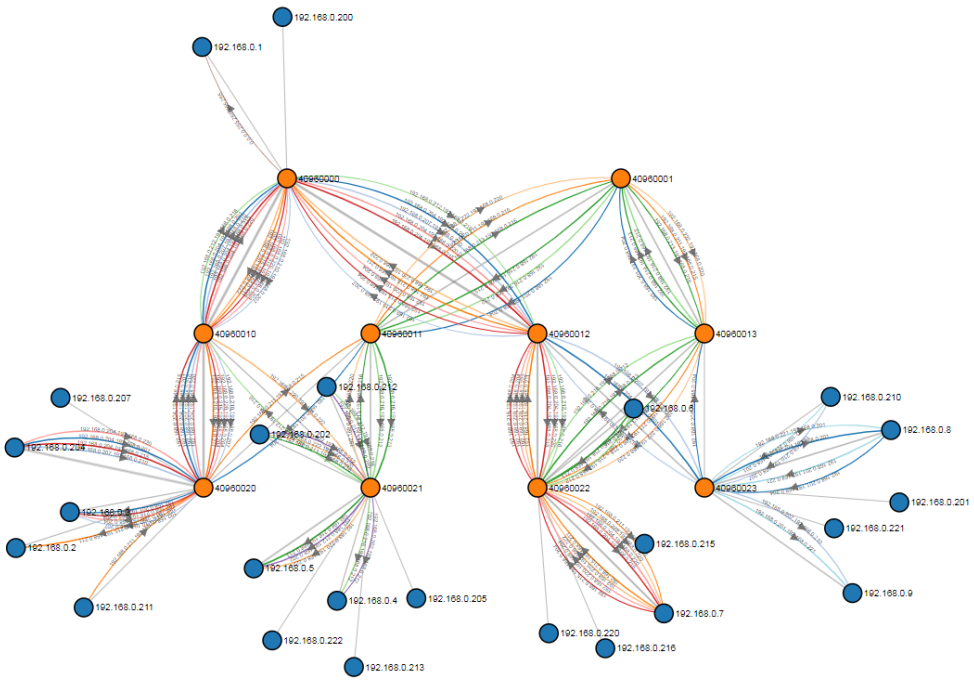
The network traffic visualized by SDCon shows the effectiveness of TOPO-Het in network resources (Figure 6.8). When VMs are placed with FFD that is lack of the information of network topology, the network traffics for both applications flood across all the network links in the testbed. In contrast, VMs placed with TOPO-Het are consolidated under the same pod or edge switches which reduce overall network traffic significantly.

### 6.7.3 Analysis of Response Time

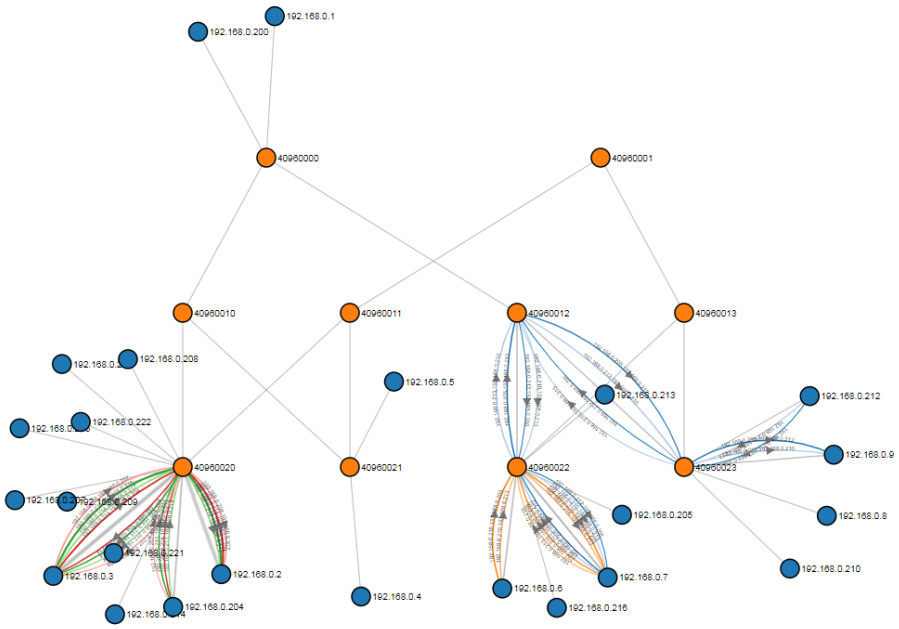
Figure 6.9 and Figure 6.10 show the average response time of Wikipedia applications measured with WikiBench program from App 1 and App2 respectively. Web requests are sent for 30 minutes with 10% of the original trace from Wikipedia. Each client injects the workload individually from a local file to avoid the latency between client VMs. For clear comparison, we further separate the result based on the type of request. DB access is the request that needs a database access to retrieve the requested contents, whereas static file is to transfer files from the web server without access to the database.

Table 6.3: Network flow settings for VM traffic.

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

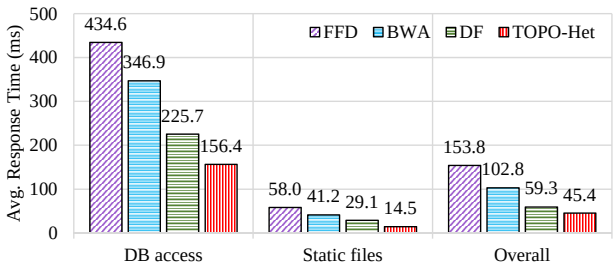



(a) Network traffic from Wikipedia application VMs placed with FFD algorithm.

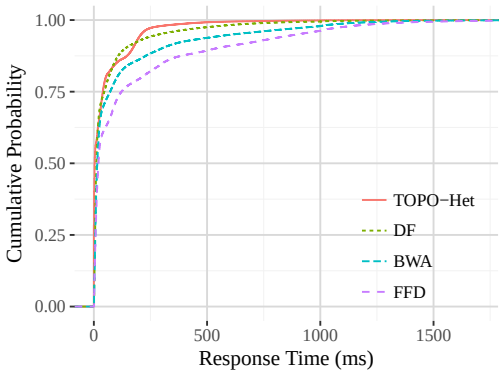



(b) Network traffic from Wikipedia application VMs placed with TOPO-Het algorithm.

Figure 6.8: Network traffic after Wikipedia application deployment visualized with Status Visualizer.




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

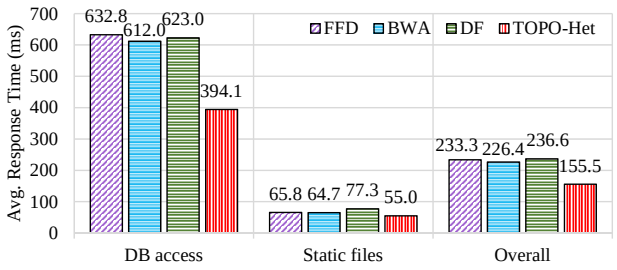



(b) CDF graph of requests' response time of App 1.

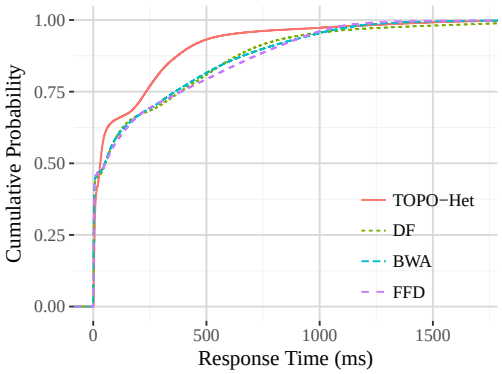
Figure 6.9: Average and CDF of response time measured with Wikipedia workload from App 1.

For the result from App 1 (see Figure 6.9), 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 entire data center which leads to increase 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 also 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 (Figure 6.9b) also shows that TOPO-Het outperforms compared to baselines.

A similar result can be observed in Figure 6.9a measured from App 2 which was running simultaneously with App 1. Compared to App 1, the overall average response times




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




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

Figure 6.10: Average and CDF of response time measured with Wikipedia workload from App 2.

are 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. Another reason is that VMs for App 2 are more scattered across the data center into smaller compute nodes as observed in Figure 6.7 leading to increasing the network latency.

We can still observe 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 was 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 Figure 6.10a, the time difference for static file requests are not as large as the difference for DB access requests between FFD and TOPO-Het. This implies the network traffic between the database and web servers is improved by TOPO-Het which allocates web servers into the compute node under the same edge

as the database VM.

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 by allocating VMs to nearby compute nodes aware of network topology, although the level of improvement is differentiated based on the application configuration, the workload and the current status of the infrastructure.

### 6.7.4 Analysis of Power Consumption

We also measured the power consumption of the data center using SDCon. Figure 6.11 depicts the power usage per time unit measured 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 connected ports for switches. 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 of SDN with cloud management, measuring power usage, and real-time resource monitoring, empirical studies on SDN-enabled clouds are feasible with SDCon platform.

## 6.8 Summary

In this chapter, we introduced SDCon[8], 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

[8]Source code available at: `github.com/cloudslab/sdcon`

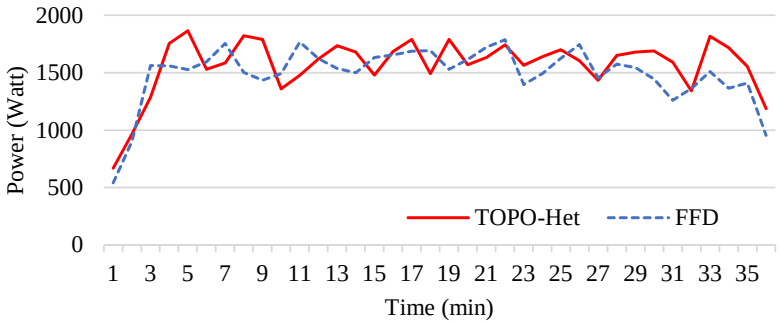



Figure 6.11: Overall power usage of the test bed estimated from CPU and network utilization.

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.

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 traffics into closely connected compute nodes in order to decrease the network traffic as well as to improve the network performance. The algorithm is evaluated on our testbed controlled by SDCon platform and compared with the measurement from different baselines. The result shows that the proposed algorithm outperforms the compared state-of-the-art baselines. The effectiveness of SDCon's controllability for networking resources is also shown through the validation and evaluation experiments, including bandwidth allocation with QoS configuration and dynamic flow scheduling.

# Chapter 7
# Conclusions and Future Directions

*This chapter concludes the presented research work in this thesis and highlights the key contributions. It also discusses open challenges and future directions in SDN-enabled cloud computing, which we could not have investigated in the thesis.*

## 7.1 Conclusions and Discussion

THE introduction of SDN has brought up many opportunities when it is applied in cloud computing environment. SDN enabled dynamic adaptation and reconfiguration with its discretion of control plane from the forwarding device controlled by the centralized controller. The SDN controller manages the network flows dynamically and individually with the global view of the entire network. Its dynamic controllability has led to significant benefits in cloud data centers where in essence the requirements and the utilization change dynamically on demand. The openness of programmable and customizable network control logic in SDN has also fostered the innovation on cloud DCN management schemes which can be easily developed and evaluated in open-source platforms.

With the adoption of SDN in clouds, *integrated resource provisioning of compute and network resources* has become feasible and effective to manage both resources simultaneously and dynamically. However, the massive scale of compute and network resources in a cloud data center makes joint resource provisioning a complex and challenging problem. This thesis investigated the joint resource provisioning methodologies to allocate

This chapter is partially derived from: Jungmin Son and Rajkumar Buyya, "A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing," *ACM Computing Surveys*, vol.51, no.3, article 59, 2018.

sufficient resources for application performance and SLA while considering the energy efficiency of a cloud data center. In detail, the background of core technologies enabling SDN-enabled cloud data centers, and the detailed objectives in accordance with the research problem has been delineated in Chapter 1.

Chapter 2 presented a taxonomy and comprehensive literature survey in the area of SDN-enabled cloud computing for energy efficiency, performance, virtualization, and security. It also depicted the state-of-the-art SDN-cloud architectures, simulation tools, and empirical platforms for performance evaluation. The analysis of current research has assisted in finding gaps and identifying open research challenges that have clarified the direction of this thesis. Chapter 2 also defined the terms and definitions in SDN-enabled cloud computing in order to clarify the ambiguous terms proposed in different works.

In Chapter 3, we modeled SDN components in a cloud data center to abstract the functionality and behavior of SDN controller and switches. Based on the model, a discrete-event simulation framework has been proposed and implemented in the extension of CloudSim, with the extra capability of managing SDN switches with a programmable network control logic of the abstracted controller. Physical hosts and switches were modeled in conjunction with virtual resources which can be provisioned in the abstracted physical resources. User requests were also abstracted for the convenience of running reproducible experiments. To measure the network bandwidth for comparison, Iperf tool was exploited in various network configurations on Mininet environment. The framework was validated through a series of experiments and compared with the result from a practical system.

Upon the developed simulation framework, we proposed two novel heuristics for joint compute and network resource provisioning in Chapter 4 and Chapter 5. A dynamic overbooking algorithm was proposed in Chapter 4 based on the correlation of computing and networking resource utilization. The algorithm analyzes the connectivity of VMs for initial VM placement, then consolidates or distributes VMs based on the selected placement method with overbooking. After initial placement, the utilization level of processing cores and network bandwidth is periodically monitored to calculate the correlation coefficient between VMs. A highly utilized VM in an overloaded and overbooked host is migrated to the either more correlated or less correlated host. A VM is

migrated to more correlated host in order to reduce the inter-VM network traffic, which is effective for network-intensive applications. On the other hand, a migration to less correlated host is effective for CPU-intensive workloads because running less correlated VMs in the same host can avoid reaching the peak CPU utilization at the same time in an overbooked host. The performance evaluation conducted with real-world traces showed that the effectiveness of the proposed algorithm varies depending on the characteristic of the workload.

Chapter 5 considered the priority of applications on resource allocation and provision more resources to the higher priority application to fulfill its QoS requirement. The priority-aware VM allocation (PAVA) algorithm selects compute nodes with more capacity and closer proximity for the priority application and utilizes the low capacity nodes for low priority applications. With PAVA, VMs for a priority application were consolidated in closely connected hosts which led to improving the response time and reducing energy. Our network bandwidth allocation method (BWA) showed that the provisioned network bandwidth is increased for a priority application as the network resource was dedicated to the usage of the priority application even in the network congestion.

Chapter 6 implemented an empirical control platform, named SDCon, for integrated resource provisioning of both cloud and network resources in the SDN-enabled data center based on OpenStack and OpenDaylight. We presented the design concept and control flows of the platform. The implemented components and their functionalities were detailed in accordance with the underlying OpenStack and OpenDaylight platforms. The proposed control framework is capable of provisioning VMs on OpenStack compute nodes and network flows with OpenVSwitch switches controlled by OpenDaylight. It also supports resource monitoring for both compute and network devices, which are exploited for power usage estimation. The visualization component provides a web-based GUI for data center administrators to be able to oversee the current status of the physical and virtual resources.

In addition, Chapter 6 proposed a topology-aware VM placement algorithm in a heterogeneous cloud data center. The algorithm considers the data center network topology consisting of heterogeneous compute nodes with different hardware spec. With the knowledge of the network connectivity, VMs are collectively placed in a host group with

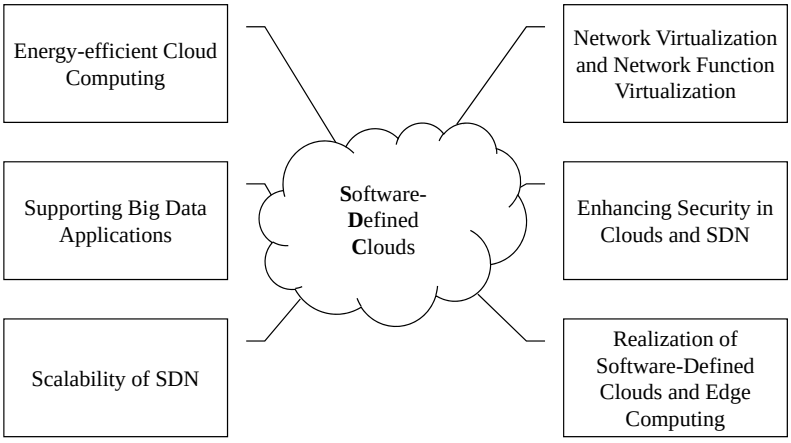



Figure 7.1: Future directions.

sufficient capacity that can be provisioned for the selected VMs. The algorithm consolidates the set of VMs with network traffic in a closely connected compute nodes. We implemented the algorithm on SDCon platform to conduct benchmark experiment with WikiBench, a realistic benchmarking tool using Wikipedia traces. The result showed that the proposed algorithm reduced the network traffic across the data center and consequently improved the network performance and response time, compared with the state-of-the-art baseline. The experiment also showed the effectiveness of SDCon, which was able to allocate a specific network bandwidth for selected VMs, reroute per-flow path dynamically, and estimate power consumption from compute nodes and network switches.

## 7.2 Future Directions

Although the investigated methodologies in this thesis contributes to the advancement of SDN-enabled cloud computing and joint resource provisioning, there are still several aspects that need to be explored comprehensively. Figure 7.1 presents the overview of future directions we discuss in this section.

### 7.2.1 Energy-efficient Cloud Computing

Cloud data centers consume an enormous amount of electricity that has been increasing rapidly. The annual amount of energy consumption of U.S. data centers was estimated at 91 billion kilowatt-hours in 2013 which is enough to power all households in NYC for

two years. It is even expected to be increased to approximately 140 billion kilowatt-hours in 2020 which will cost US$13 billion annually [84]. Thus, improving energy-efficiency in cloud computing has become an utmost research problem in academia and industries. Although many researchers have already contributed to reducing the energy consumption in cloud computing, most of them try to optimize computing resources or network resources separately.

Joint optimization of computing and networking resources for reducing power consumption is one of the less explored aspects. Joint optimization considering both computing and network resources simultaneously is rare due to the lack of supportive technology which can control both at the same time. With the recent evolution of SDN, joint optimization is feasible in technology and can be investigated further to save more energy and provide better QoS. Service Level Agreement (SLA) must also be fulfilled in the power optimization to guarantee profit maximization for cloud providers.

### 7.2.2 Supporting Big Data Applications

Cloud computing becomes a fundamental infrastructure for Big Data applications with its massive resource requirements for computation jobs and network transmission. Utilizing SDN-enabled cloud infrastructure can bring vital benefits to Big Data applications to reduce network bottlenecks during the application processing. For example, aggregation of the mapped data from mappers to the reducer can cause a network burst in a short time when the mapping processes are completed at the similar time. SDN can be exploited for network-aware scheduling of Big Data workloads in Map-reduce platform such as Hadoop. Network bandwidth is considered using SDN for Hadoop job scheduling to guarantee data locality and optimize task assignment among available workers [102]. In addition to the computation processing time of the workload, network limitation between workers can also be considered when scheduling and distributing jobs.

Researchers also investigate SDN usage for large data transfers in distributed file access system and for content delivery cache network. SDN can be exploited to collect network information and find the optimal path between the closest data source and the consumer in distributed file system [66]. For a content delivery network, distributed

cache nodes can be considered as network switches in SDN paradigm which are controlled by cache network controller that optimizes data transfers between cache nodes by updating the popularity statistics of contents measured at each cache node [26]. As few researchers have been studied how to utilize SDN in cloud data centers for Big Data applications, more studies are necessary in this regard.

### 7.2.3 Scalability of SDN

Scalability of SDN controller is a critical concern for SDN deployment due to its centralized logic of the architecture and the separation of the control plane from forwarding plane. As the controller gathers all network information and manages every switch in the entire network, it is challenging to make the controller scalable and prevent the single-point of failure at the controller. In clouds, this pitfall can be a significant issue due to the size and complexity of DCN, and the expected SLA for the provider. The SDN controller for the cloud DCN can easily become a single-point of failure as controllers in DCN can be overloaded due to the tremendous number of switches. The problem can be tackled by distributing controllers or proactively installing the flow rules on switches [130].

A single controller design is simple and easy for maintenance with a global network view, but it is considered less scalable compare to other topological architecture of SDN controllers [63]. SDN controllers can be distributed with a flat or hierarchical architecture via the west-east bound APIs. The west-east bound APIs can be utilized to communicate between the controllers so that the controllers can be scaled by exchanging network information and policies through the APIs. Empowering the performance of controller machines is also exploited to increase SDN scalability which leads the controllers to handle more packet flows and reduce overhead. Multi-core parallel processing and routing optimization are proposed to improve I/O performance and reduce memory overhead [63].

### 7.2.4 Network Virtualization and NFV

There are also gaps in realizing network virtualization in clouds which can be more feasible with the integration of SDN. Although the traditional networking technologies such as VLAN and VPN have been widely adopted in industry, integrating SDN with the tra-

ditional approaches can bring even better performance and more flexibility to clouds. As DCN has a complex topology with a large number of devices connected through various switches, network virtualization in traditional DCN was difficult to achieve due to the complexity of network configuration and the lack of adaptability. With SDN, flows between VMs can be fine-controlled more flexibly and dynamically adapting to the changing traffic. Switches can be dynamically configured in real-time based on the decision of the controller which can oversee the entire network. Thus, network virtualization for network SLA guarantee is possibly investigated and introduced in practice with more extensive research in SDN-enabled cloud data centers.

Recently, NFV has acquired significant attention in both cloud computing and networking research communities. By virtualizing network functions which used to be provided by expensive and tightly coupled hardware, providers can reduce the cost of purchasing and upgrading the hardware as the network functions can run on generic hardware like running VM on generic hosts. In networking aspects, how to virtualize those network functions is one of the utmost research topics whereas resource allocation and provisioning for NFV have been studied in cloud computing aspects. Both aspects should be further explored in order to enhance and implement NFV.

### 7.2.5 Enhancing Security and Reliability in Clouds and SDN

For public cloud providers, security of the data center and their data is a crucial factor of the business that must be fulfilled. While security in cloud data centers has been explored extensively with diverse approaches, security of SDN usage, in particular, still requires being investigated to provide insurance for providers applying SDN in their data centers. Current security issues include encrypting SDN control packets between a switch and the controller, protecting the SDN controller from an abnormal access, as well as protecting packets passing in the virtualized network. Also, researchers put more effort on investigating the vulnerability of SDN itself from DDoS attack [55] and the security issue of the stateful SDN data planes [28]. As a massive number of different tenants share same network medium in the cloud, it is crucial to ensure the virtualized network to be secure and isolated from other tenants. In the unexpected event of resource failures, cloud system must be able to overcome the failure by providing fault-tolerant algorithms to en-

hance the reliability of clouds [57]. For SDN, it is also critical to assure the reliability and availability of SDN controllers and switches to maintain the proper operation of network system under unexpected threats and failures [128]. Therefore, it is important to investigate these aspects of improving security and reliability of clouds and SDN to build fully automated and fault-tolerant Software-Defined Clouds.

### 7.2.6 Realization of Software-Defined Clouds and Edge Computing

Although the fundamental architecture of autonomous Software-Defined Clouds (SDC) is proposed in several works [16,56], the implementation of SDC is still in the early stage due to the lack of virtualization technologies for cloud resources and the integration of these techniques in clouds. Leasing computing and storage resources have been already implemented in commercial cloud platforms thanks to the recent enhancement in virtualization technologies, but further investigation in network slicing and dynamic management is needed to support fully automated SDC. In order to support network slicing resource management, SDN is a core technology that can bring autonomic network function and application deployment in clouds.

Furthermore, SDN can be functioned as a key networking technology for edge computing where the current cloud computing concept is extended to the edge of the network. In edge computing, time-critical and recurring workloads can be processed at the edge nodes close to end-users without transferring to the central cloud infrastructures so that the application responsiveness and the networking efficiency can be improved [40]. It is heavily driven by emerging IoT applications where the low-profile devices equipped with sensors collectively generate a massive amount of data continuously. With the emergence of edge computing, the large volume of data can be processed at the edge of the network without transmission to the central cloud infrastructure. The optimal decision has to be made in edge-cloud environment whether migrating data to edge nodes or hosting in the central cloud [6]. Recently researchers also have started exploring virtualized network function (VNF) placement for edge computing to enable auto-scaling and placing the VNFs across the edges and the clouds. SDN can be utilized for WAN optimization to manage the traffic between the edge and cloud on top of network slicing and VNF placement [50].

## 7.3 Software Availability

CloudSimSDN presented in Chapter 3 and SDCon presented in Chapter 6 are open-sourced and freely available to download on CLOUDS-lab GitHub website:

- CloudSimSDN: `github.com/cloudslab/cloudsimsdn`
- SDCon: `github.com/cloudslab/sdcon`

# Bibliography

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*. New York, NY, USA: ACM, 2010, pp. 338–347. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/1815961.1816004

[2] 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," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 11, p. e3172, 2017. [Online]. Available: http://dx.doi.org/10.1002/ett.3172

[3] A. V. Akella and K. Xiong, "Quality of Service (QoS)-guaranteed network resource allocation via software defined networking (SDN)," in *Proceedings of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. Washington, DC, USA: IEEE Computer Society, 2014, pp. 7–13. [Online]. Available: http://dx.doi.org/10.1109/DASC.2014.11

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. New York, NY, USA: ACM, 2008, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402967

[5] H. Amarasinghe, A. Jarray, and A. Karmouch, "Fault-tolerant IaaS management for networked cloud infrastructure with SDN," in *Proceedings of the 2017 IEEE International Conference on Communications*, May 2017, pp. 1–7.

[6] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Rajan, "Optimal decision making for big data processing at edge-cloud environment: An sdn perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778–789, Feb 2018.

[7] M. Azizian, S. Cherkaoui, and A. S. Hafd, "Vehicle software updates distribution with SDN and cloud computing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 74–79, 2017.

[8] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 54–62, July 2013.

[9] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an SDN platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, February 2013.

[10] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[11] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.04.017

[12] 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 and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.1867

[13] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[14] C. H. Benet, R. Nasim, K. A. Noghani, and A. Kassler, "OpenStackEmu - a cloud testbed combining network emulation with OpenStack and SDN," in *Proceedings*

*of the 2017 14th IEEE Annual Consumer Communications Networking Conference*, Jan 2017, pp. 566–568.

[15] R. Bonafiglia, G. Castellano, I. Cerrato, and F. Risso, "End-to-end service orchestration across sdn and cloud computing domains," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–6.

[16] R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon, "Software-defined cloud computing: Architectural elements and open challenges," in *Proceedings of the 3rd International Conference on Advances in Computing, Communications and Informatics*, IEEE. IEEE Press, 2014.

[17] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[19] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, and A. Manzalini, "Sdn for dynamic nfv deployment," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 89–95, October 2016.

[20] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Communications of the ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/2661061.2661063

[21] W. Cerroni, M. Gharbaoui, B. Martini, A. Campi, P. Castoldi, and F. Callegati, "Cross-layer resource orchestration for cloud service delivery: A seamless sdn approach," *Computer Networks*, vol. 87, no. Supplement C, pp. 16 – 32, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615001644

[22] A. Chowdhary, S. Pisharody, A. Alshamrani, and D. Huang, "Dynamic game based security framework in SDN-enabled cloud networking environments,"

in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. New York, NY, USA: ACM, 2017, pp. 53–58. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/3040992.3040998

[23] D. Citron and A. Zlotnick, "Testing large-scale cloud management," *IBM Journal of Research and Development*, vol. 55, no. 6, Nov. 2011.

[24] C. Clos, "A study of non-blocking switching networks," *The Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, March 1953.

[25] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Network*, vol. 30, no. 1, pp. 58–65, January 2016.

[26] Y. Cui, J. Song, M. Li, Q. Ren, Y. Zhang, and X. Cai, "Sdn-based big data caching in isp networks," *IEEE Transactions on Big Data*, 2017.

[27] R. Cziva, S. Jouët, D. Stapleton, F. P. Tso, and D. P. Pezaros, "SDN-based virtual machine management for cloud data centers," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 212–225, June 2016.

[28] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1701–1725, thirdquarter 2017.

[29] M. D. de Assunção, R. Carpa, L. Lefévre, and O. Glýck, "On designing SDN services for energy-aware traffic engineering," in *Proceedings of the 11th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, 1 2017.

[30] F. R. de Souza, C. C. Miers, A. Fiorese, and G. P. Koslovski, "QoS-aware virtual infrastructures allocation on sdn-based clouds," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. Piscataway, NJ, USA: IEEE Press, 2017, pp. 120–129. [Online]. Available: https://doi-org.ezp.lib.unimelb.edu.au/10.1109/CCGRID.2017.57

[31] X. Du, Z. Lv, J. Wu, C. Wu, and S. Chen, "PDSDN: A policy-driven SDN controller improving scheme for multi-tenant cloud datacenter environments," in *Proceedings of the 2016 IEEE International Conference on Services Computing (SCC)*, June 2016, pp. 387–394.

[32] 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 *Proceedings of the 2012 Asia-Pacific Signal Information Processing Association Annual Summit and Conference*, Dec 2012, pp. 1–8.

[33] D. Ersoz, M. S. Yousif, and C. R. Das, "Characterizing network traffic in a cluster-based, multi-tier data center," in *Proceedings of the 27th International Conference on Distributed Computing Systems*. IEEE, 2007, pp. 59–59.

[34] F. Esposito, I. Matta, and V. Ishakian, "Slice embedding solutions for distributed service architectures," *ACM Computing Surveys*, vol. 46, no. 1, pp. 6:1–6:29, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2522968.2522974

[35] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, no. 1, pp. 179 – 196, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612003301

[36] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2559899.2560327

[37] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," *Emerging Research in Cloud Distributed Computing Systems*, vol. 42, 2015.

[38] S. Fichera, M. Gharbaoui, P. Castoldi, B. Martini, and A. Manzalini, "On experimenting 5g: Testbed set-up for sdn orchestration across network cloud and iot domains," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–6.

[39] A. Fischer, F. B. Botero, M. Till Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[40] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, Sep. 2015. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/2831347.2831354

[41] S. K. Garg and R. Buyya, "NetworkCloudSim: Modelling parallel applications in cloud simulations," in *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 105–113. [Online]. Available: http://dx.doi.org/10.1109/UCC.2011.24

[42] M. Gharbaoui, B. Martini, D. Adami, S. Giordano, and P. Castoldi, "Cloud and network orchestration in sdn data centers: Design principles and performance evaluation," *Computer Networks*, vol. 108, no. Supplement C, pp. 279 – 295, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128616302821

[43] K. Govindarajan, K. C. Meng, H. Ong, W. M. Tat, S. Sivanand, and L. S. Leong, "Realizing the quality of service (QoS) in software-defined networking (SDN) based cloud infrastructure," in *Proceedings of the 2nd International Conference on Information and Communication Technology (ICoICT)*, May 2014, pp. 505–510.

[44] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, July 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[45] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data*

*Communication*. New York, NY, USA: ACM, 2009, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592577

[46] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*. New York, NY, USA: ACM, 2008, pp. 75–86. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402968

[47] P. Habibi, M. Mokhtari, and M. Sabaei, "QRVE: QoS-aware routing and energy-efficient vm placement for software-defined datacenter networks," in *Proceedings of the 8th International Symposium on Telecommunications*, Sept 2016, pp. 533–539.

[48] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2010, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855728

[49] C. Hopps, "Analysis of an equal-cost multi-path algorithm," Internet Requests for Comments, RFC Editor, RFC 2992, November 2000. [Online]. Available: https://tools.ietf.org/html/rfc2992

[50] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing-a key technology towards 5g," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[51] iPerf, "iPerf: The ultimate speed test tool for tcp, udp and sctp," 2017. [Online]. Available: https://iperf.fr//

[52] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Proceedings of the 2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 81–86.

[53] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, November 2013.

[54] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. New York, NY, USA: ACM, 2013, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486019

[55] S. Jantila and K. Chaipah, "A security analysis of a hybrid mechanism to defend ddos attacks in sdn," *Procedia Computer Science*, vol. 86, no. Supplement C, pp. 437 – 440, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916304082

[56] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "Software defined cloud: Survey, system and evaluation," *Future Generation Computer Systems*, vol. 58, pp. 56 – 74, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15003283

[57] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of Parallel and Distributed Computing*, vol. 72, no. 10, pp. 1318 – 1331, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731512001517

[58] K. Jeong and R. Figueiredo, "Self-configuring software-defined overlay bypass for seamless inter- and intra-cloud virtual networking," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. New York, NY, USA: ACM, 2016, pp. 153–164. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/2907294.2907318

[59] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *Proceedings of the 2012 IEEE INFOCOM*, March 2012, pp. 2876–2880.

[60] 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 *Proceedings of the 2013 IEEE 27th International Symposium on Parallel Distributed Processing*, May 2013, pp. 623–634.

[61] J. Kang and S. Park, "Algorithms for the variable sized bin packing problem," *European Journal of Operational Research*, vol. 147, no. 2, pp. 365 – 372, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221702002473

[62] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges," in *Proceedings of the 3rd IEEE International Advance Computing Conference*, Feb 2013, pp. 963–969.

[63] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Computer Networks*, vol. 112, pp. 279 – 293, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861630411X

[64] Y. Kim, S. Kang, C. Cho, and S. Pahk, "Sdn-based orchestration for interworking cloud and transport networks," in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct 2016, pp. 303–307.

[65] D. Kliazovich, P. Bouvry, and S. U. Khan, "Greencloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, Dec 2012. [Online]. Available: https://doi.org/10.1007/s11227-010-0504-1

[66] S. Koulouzis, A. S. Belloum, M. T. Bubak, Z. Zhao, M. Živković, and C. T. de Laat, "Sdn-aware federation of distributed data," *Future Generation Computer Systems*, vol. 56, no. Supplement C, pp. 64 – 76, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1500312X

[67] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and stochastic netshare: Predictable bandwidth allocation for data centers," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 5–11, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2317307.2317309

[68] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[69] A. Leivadeas, M. Falkner, I. Lambadaris, and G. Kesidis, "Optimal virtualized network function allocation for an SDN enabled cloud," *Computer Standards & Interfaces*, vol. 54, no. P4, pp. 266–278, Nov. 2017. [Online]. Available: https://doi.org/10.1016/j.csi.2017.01.001

[70] M. Licciardello, M. Fiorani, M. Furdek, P. Monti, C. Raffaelli, and L. Wosinska, "Performance evaluation of abstraction models for orchestration of distributed data center networks," in *2017 19th International Conference on Transparent Optical Networks (ICTON)*, July 2017, pp. 1–4.

[71] S.-C. Lin, P. Wang, and M. Luo, "Jointly optimized QoS-aware virtualization and routing in software defined networks," *Computer Networks*, vol. 96, pp. 69 – 78, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912861500256X

[72] 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 *Proceedings of the 2016 IEEE International Conference on Communications*, May 2016, pp. 1–6.

[73] 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 *2014 European Conference on Networks and Communications (EuCNC)*, June 2014, pp. 1–5.

[74] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Computing Surveys*, vol. 47, no. 2, pp. 33:1–33:36, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656204

[75] A. Mayoral, R. Munoz, R. Vilalta, R. Casellas, R. Martínez, and V. López, "Need for a transport api in 5g for global orchestration of cloud and networks through a virtualized infrastructure manager and planner," *J. Opt. Commun. Netw.*, vol. 9, no. 1, pp. A55–A62, Jan 2017. [Online]. Available: http://jocn.osa.org/abstract.cfm?URI=jocn-9-1-A55

[76] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, and R. Martínez, "SDN orchestration architectures and their integration with cloud computing applications," *Optical Switching and Networking*, vol. 26, pp. 2 – 13, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1573427716000047

[77] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, R. Martínez, M. S. Moreolo, J. M. Fabrega, A. Aguado, S. Yan, D. Simeonidou, J. M. Gran, V. López, P. Kaczmarek, R. Szwedowski, T. Szyrkowiec, A. Autenrieth, N. Yoshikane, T. Tsuritani, I. Morita, M. Shiraiwa, N. Wada, M. Nishihara, T. Tanaka, T. Takahara, J. C. Rasmussen, Y. Yoshida, and K. ichi Kitayama, "Control orchestration protocol: Unified transport api for distributed cloud and network orchestration," *J. Opt. Commun. Netw.*, vol. 9, no. 2, pp. A216–A222, Feb 2017. [Online]. Available: http://jocn.osa.org/abstract.cfm?URI=jocn-9-2-A216

[78] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[79] M. Mechtri, I. Houidi, W. Louati, and D. Zeghlache, "SDN for inter cloud networking," in *Proceedings of the 2013 IEEE SDN for Future Networks and Services*, Nov 2013, pp. 1–7.

[80] V. Medina and J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys*, vol. 46, no. 3, pp. 30:1–30:33, Jan. 2014. [Online]. Available: http://doi.acm.org/10.1145/2492705

[81] P. M. Mell and T. Grance, "SP 800-145. the NIST definition of cloud computing," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

[82] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[83] K. Mills, J. Filliben, and C. Dabrowski, "Comparing vm-placement algorithms for on-demand clouds," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, Nov 2011, pp. 91–98.

[84] Natural Resources Defense Council, "America's data centers consuming and wasting growing amounts of energy," 2014. [Online]. Available: http://www.nrdc.org/energy/data-center-efficiency-assessment.asp

[85] ns2, "The network simulator ns-2," http://www.isi.edu/nsnam/ns/, 1995–2011.

[86] ns3, "The network simulator ns-3," http://www.nsnam.org/, 2011–2015.

[87] NTT Corporation, "RYU the network operating system (NOS)," 2017. [Online]. Available: https://osrg.github.io/ryu/

[88] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, Mar. 2012.

[89] ONF Solution Brief, "OpenFlow-enabled SDN and network functions virtualization," *Open Network Foundation*, 2014. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/05/sb-sdn-nvf-solution.pdf

[90] Open Network Foundation, "Software-defined networking: The new norm for networks," https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2012.

[91] Open Networking Lab, "Introducing ONOS - a SDN network operating system for service providers," 2014. [Online]. Available: http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf

[92] OpenDaylight, "OpenDaylight: Open source SDN platform," 2017. [Online]. Available: https://www.opendaylight.org/

[93] OpenStack Foundation, "Open source software for creating private and public clouds." 2017. [Online]. Available: https://www.openstack.org/

[94] E. Pakbaznia and M. Pedram, "Minimizing data center cooling and server power costs," in *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*. New York, NY, USA: ACM, 2009, pp. 145–150. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/1594233.1594268

[95] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Workshop on Energy-Efficient Design (WEED)*, 2009.

[96] I. Petri, M. Zou, A. R. Zamani, J. Diaz-Montes, O. Rana, and M. Parashar, "Integrating software defined networks within a cloud federation," in *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2015, pp. 179–188.

[97] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed sdn-based cloud environments," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[98] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM, 2012, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342396

[99] POX, "Pox," 2013. [Online]. Available: http://www.noxrepo.org/

[100] Project Floodlight, "Floodlight openflow controller," 2017. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[101] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. New York, NY, USA: ACM, 2013, pp. 27–38.

[102] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with sdn in hadoop: A new trend for big data," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2337–2344, Dec 2017.

[103] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12331-3_2

[104] A. C. Risdianto, J.-S. Shin, and J. W. Kim, "Deployment and evaluation of software-defined inter-connections for multi-domain federated SDN-cloud," in *Proceedings of the 11th International Conference on Future Internet Technologies*. New York, NY, USA: ACM, 2016, pp. 118–121. [Online]. Available: http://doi.acm.org.ezp.lib.unimelb.edu.au/10.1145/2935663.2935683

[105] D. Samant and U. Bellur, "Handling boot storms in virtualized data centers—a survey," *ACM Computing Surveys*, vol. 49, no. 1, pp. 16:1–16:36, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2932709

[106] D. J. Schumacher and W. C. Beckman, "Data center cooling system," April 2002, US Patent 6,374,627.

[107] S. Seetharam, P. Calyam, and T. Beyene, "ADON: Application-driven overlay network-as-a-service for data-intensive science," in *Proceedings of the 2014 IEEE 3rd International Conference on Cloud Networking*, Oct 2014, pp. 313–319.

[108] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM, 2012, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342359

[109] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2011, pp. 309–322. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972489

[110] S. Spadaro, A. Pagès, F. Agraz, R. Montero, and J. Perelló, "Resource orchestration in sdn-based future optical data centres," in *2016 International Conference on Optical Network Design and Modeling (ONDM)*, May 2016, pp. 1–6.

[111] I. Sriram, "SPECI, a simulation tool exploring cloud-scale data centres," in *Proceedings of the First International Conference on Cloud Computing Technology and Science (CloudCom)*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5931, pp. 381–392.

[112] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos, "Datacenter in a box: Test your SDN cloud-datacenter controller at home," in *Proceedings of the 2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 99–104.

[113] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys*, vol. 47, no. 1, pp. 7:1–7:47, May 2014. [Online]. Available: http://doi.acm.org/10.1145/2593512

[114] A. N. Toosi, C. Qu, M. D. de Assunção, and R. Buyya, "Renewable-aware geographical load balancing of web applications for sustainable data centers," *Journal of Network and Computer Applications*, vol. 83, no. Supplement C, pp. 155 – 168, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517300590

[115] 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 Computer Systems*, vol. 28, no. 2, pp. 358 – 367, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11001373

[116] 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 *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, July 2013, pp. 108–112.

[117] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Elsevier Computer Networks*, vol. 53, no. 11, pp. 1830–1845, July 2009.

[118] A. Vahdat, D. Clark, and J. Rexford, "A purpose-built global network: Google's move to SDN," *Queue*, vol. 13, no. 8, pp. 100:100–100:125, Oct. 2015. [Online]. Available: http://doi.acm.org/10.1145/2838344.2856460

[119] R. Wang, S. Mangiante, A. Davy, L. Shi, and B. Jennings, "QoS-aware multipathing in datacenters using effective bandwidth estimation and SDN," in *Proceedings of the 12th International Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 342–347.

[120] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. Berkeley, CA, USA: USENIX Association, 2011, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972422.1972438

[121] 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 *Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop*, Nov 2014, pp. 100–108.

[122] R. Wang, J. A. Wickboldt, R. P. Esteves, L. Shi, B. Jennings, and L. Z. Granville, "Using empirical estimates of effective bandwidth in network-aware placement of virtual machines in datacenters," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 267–280, June 2016.

[123] S.-H. Wang, P. P. W. Huang, C. H. P. Wen, and L. C. Wang, "EQVMP: Energy-efficient and qos-aware virtual machine placement for software defined datacenter networks," in *The International Conference on Information Networking 2014 (ICOIN2014)*, Feb 2014, pp. 220–225.

[124] X. Wang, X. Wang, K. Zheng, Y. Yao, and Q. Cao, "Correlation-aware traffic consolidation for power optimization of data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 992–1006, April 2016.

[125] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "CARPO: Correlation-aware power optimization in data center networks," in *Proceedings of the 2012 IEEE INFOCOM*, March 2012, pp. 1125–1133.

[126] W. Wendong, Q. Qinglei, G. Xiangyang, H. Yannan, and Q. Xirong, "Autonomic QoS management mechanism in software defined network," *Communications, China*, vol. 11, no. 7, pp. 13–23, July 2014.

[127] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. C. M. Lau, "Orchestrating bulk data transfers across geo-distributed datacenters," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 112–125, Jan 2017.

[128] A. Xie, X. Wang, W. Wang, and S. Lu, "Designing a disaster-resilient network with software defined networking," in *Proceedings of the IEEE 22nd International Symposium of Quality of Service (IWQoS)*, May 2014, pp. 135–140.

[129] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.

[130] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, February 2013.

[131] Z. Zhang, Y. Deng, G. Min, J. Xie, and S. Huang, "ExCCC-DCN: A highly scalable, cost-effective and energy-efficient data center structure," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1046–1060, April 2017.

[132] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proceedings of the 2014 IEEE INFOCOM*, April 2014, pp. 2598–2606.

[133] K. Zheng and X. Wang, "Dynamic control of flow completion time for power efficiency of data center networks," in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems*, June 2017, pp. 340–350.

[134] K. Zheng, X. Wang, and J. Liu, "DISCO: Distributed traffic flow consolidation for power efficient data center network," in *Proceedings of the 16th International IFIP TC6 Networking Conference, Networking*. IFIP Open Digital Library, 2017. [Online]. Available: http://dl.ifip.org/db/conf/networking/networking2017/1570334940.pdf

[135] K. Zheng, W. Zheng, L. Li, and X. Wang, "PowerNetS: Coordinating data center network with servers and cooling for power optimization," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 661–675, Sept 2017.