SERVER AND WIRELESS NETWORK RESOURCE ALLOCATION STRATEGIES IN HETEROGENEOUS CLOUD DATA CENTERS

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER ENGINEERING

By

Cem Mergenci

August 2020

Server and Wireless Network Resource Allocation Strategies in Heterogeneous Cloud Data Centers
By Cem Mergenci
August 2020

We certify that we have read this dissertation and that in our opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

İbrahim Körpeoğlu (Advisor)

Özgür Ulusoy

Ezhan Karaşan

Ahmet Coşar

Ertan Onur

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

SERVER AND WIRELESS NETWORK RESOURCE ALLOCATION STRATEGIES IN HETEROGENEOUS CLOUD DATA CENTERS

Cem Mergenci
Ph.D. in Computer Engineering
Advisor: İbrahim Körpeoğlu
August 2020

Resource allocation is one of the most important challenges in operating a data center. We investigate allocation of two main types of resources: servers and network links.

Server resource allocation problem is the problem of how to allocate virtual machines (VMs) to physical machines (PMs). By modeling server resources (CPU, memory, storage, IO, etc.) as a multidimensional vector space, we present design criteria for metrics that measure the fitness of an allocation of VMs into PMs. We propose two novel metrics that conform to these design criteria. We also propose VM allocation methods that use these metrics to compare allocation alternatives when allocating a set of VMs into a set of PMs. We compare performances of our proposed metrics to the ones from the literature using vector bin packing with heterogeneous bins (VBPHB) benchmark. Results show that our methods find feasible solutions to a greater number of allocation problems than the others.

Network resource allocation problem is examined in hybrid wireless data centers. We propose a system model in which each top-of-the-rack (ToR) switch is equipped with two radios operating in 60-GHz band using 3-channel 802.11ad. Given traffic flows between servers, we allocate wireless links between ToR switches so that the traffic carried over the wireless network is maximized. We also present a method to randomly generate traffic based on a real data center traffic pattern. We evaluate the performance of our proposed traffic allocation methods using randomly generated traffic. Results show that our methods can offload significant amount of traffic from wired to wireless network, while achieving low latency, high throughput, and high bandwidth utilization.

Keywords: Resource Allocation, Cloud Computing, Vector Bin Packing, Wireless Data Center, 802.11ad, Data Center Traffic.

ÖZET

TÜRDEŞ OLMAYAN BULUT VERİ MERKEZLERİNDE SUNUCU VE KABLOSUZ AĞ KAYNAKLARI AYRIM İZLEMLERİ

Cem Mergenci
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: İbrahim Körpeoğlu
Ağustos 2020

Veri merkezi yönetiminde en önemli konulardan biri kaynak ayrımıdır. Kaynak ayrımı konusu, sunucu kaynakları ve ağ bağlantıları olmak üzere iki ana başlıkta incelenmektedir.

Sunucu kaynaklarının ayrımı problemi sanal makinelerin fiziksel makinelere atanması olarak ele alınmaktadır. Sunucu kaynaklarını (işlemci, hafıza, saklama, girdi/çıktı başarımı, vb.) çok boyutlu bir vektör uzayı olarak düşünerek, bir sanal makine ile bir fiziksel makinenin ne kadar uyumlu olduğunu belirten tasarım ölçütleri tanımlanmaktadır. Bu tasarım ölçütlerine uyan iki yeni ölçü önerilmektedir. Bu ölçüleri kullanarak farklı sanal makine yerleştirme seçeneklerini değerlendiren algoritmalar da sunulmaktadır. Bu algoritmalar bir sanal makine isteği kümesini bir fiziksel makine kümesine yerleştirmek için kullanılmaktadır. Önerdiğimiz ölçülerin başarımlarını yazındaki diğer ölçülerle karşılaştırmak için türdeş olmayan kutulara kutulama problemi için geliştirilmiş bir kıyaslama düzeni kullanılmaktadır. Kıyaslama sonuçları önerdiğimiz ölçülerin mevcut ölçülerden daha çok sayıda kutulama problemini çözdüğünü göstermektedir.

Ağ kaynaklarının ayrımı problemi melez veri merkezleri kapsamında ele alınmaktadır. Önerdiğimiz dizge kalıbına göre veri merkezinde her raf üstü ağ anahtarı, 60 GHz bandında çalışan, 3 kanallı 802.11ad iletişim kuralını işleten iki adet radyoya sahip olmaktadır. Sunucular arası trafik akış bilgisi verildiğinde, raf üstü ağ anahtarları arasındaki kablosuz bağlar belirlenerek kablosuz ağ trafiği en yüksek değere ulaştırılmaktadır. Gerçek bir veri merkezi trafik akışı bilgisinden yola çıkarak sunucular arası trafik akışını rastgele oluşturan bir yöntem de önerilmektedir. Başarım, rastgele oluşturulan birçok trafik akışı için ölçülmekte ve sonuçlara göre, önerilen yöntemler kablolu ağdan kablosuz ağa büyük miktarda trafik aktarırken aynı zamanda düşük iletişim gecikmesi, yüksek iletim hacmi ve yüksek bant genişliği kullanım oranı elde etmektedir.

Anahtar sözcükler: Kaynak Ayrımı, Bulut Bilişim, Vektör Kutulama Problemi, Kablosuz Veri Merkezi, 802.11ad, Veri Merkezi Trafiği.

Acknowledgement

Thanks to Scientific and Technological Research Council of Turkey (TÜBİTAK)[1] *and Department of Computer Engineering for financially supporting me during my graduate education.*

First, I would like to thank Prof. İbrahim Körpeoğlu. He has been more than an advisor — a teacher of wisdom. I am lucky that I had the privilege to learn from him. After all this time, he still has more to teach.

I am thankful to my PhD committee members Prof. Özgür Ulusoy and Prof. Ezhan Karaşan. They have been a source of accountability, challenge, guidance, and encouragement that I needed a lot during the process. I hope that they remember our committee meetings with a good taste in their mouth.

Prof. Ahmet Coşar and Prof. Ertan Onur have been very kind to accept being jury members. This thesis has become better thanks to their valuable feedback.

I began my PhD journey by following footsteps of Alper and Metin. Their mentorship has been very dear to my heart. Çağlar and Fırat has been my companions in this journey. Only they know deep in their hearts what we have been through.

Special thanks to my teammates with whom I took another journey alongside my PhD. Mert, Canol, Efe, Kubilay, Sébastien, and Yiğit set the bar so high that I do not think any other team would be able to surpass.

Eren deserves a special special thanks. He is definitely one of the most favorite seven friends of mine.

Last, but not least, I am grateful to my family for their sincere and endless support at every moment throughout my life and for providing me every opportunity they can, especially for my education. This thesis would be a dream without them.

[1]*I am thankful for the financial support I received from BİDEB 2211 National Graduate Scholarship Program, and ARDEB 1001 The Scientific and Technological Research Projects Funding Program with grants numbered 113E274 and 116E048.*

To the crazy ones who dare to wake up...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing provides access to seemingly infinite computing resources in an on-demand and pay-as-you-go basis. Setup time of these resources are within seconds or minutes. Payment is per minutes or hours of use. Compared to traditional hosting methods, where setup time is measured by days and payment is done per month, cloud computing offers rapid and easy scaling for applications without any upfront costs.

National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. NIST also defines three basic models of service for cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

IaaS gives control to users in configuring and managing computation, networking, storage and operating systems. IaaS is much flexible than PaaS. Users can run arbitrary software on the infrastructure, but have to deal with the associated complexities. The physical arrangement of the hardware is still hidden from users, therefore limiting architecture specific solutions. Amazon Elastic Compute

Cloud (EC2) [2], Microsoft Azure Virtual Machines [3], and Rackspace Cloud Servers [4] are among prominent IaaS offerings.

The challenge for IaaS provider is to allocate available resources to virtual machine (VM) instance requests and traffic flows. While allocating resources, there are several factors that can be considered such as utilization, energy consumption, load balancing, task allocation, and fairness. The fact that instances and traffic flows are created and terminated dynamically may require the resource allocation to be dynamic as well. Another important issue is to satisfy as many VM requests and traffic flows as possible for a given limited physical capacity.

In this thesis, we address two of the resource allocation problems in IaaS systems: (1) allocating VM requests to physical machines (PMs), and (2) allocating wireless network resources to data traffic in a hybrid wireless data center.

First, we examine how to allocate physical machine (PM) resources to VM requests on a data center consisting of heterogeneous cloud computing infrastructures. We begin by defining qualities of a good allocation by examining the nature of multi-dimensional resource allocation where PM resources are of different types, such as CPU, memory, disk, and network bandwidth. Based on the qualities of good allocations, we propose metrics that quantify an allocation state. Therefore, these metrics can be used to compare different allocation methods. More importantly, they can be used by allocation algorithms to evaluate allocation alternatives at a certain iteration of the algorithm or with every request arriving.

We also show how our metrics can be used in online and offline VM allocation schemes by proposing some heuristic algorithms utilizing the metrics. We assume that VMs do not migrate from one PM to another, hence the cost of migration is not considered in this thesis.

We evaluated the proposed metrics and methods through extensive simulation experiments. We studied how well our metrics can solve the VM placement problem. We also compared our methods with a number of existing methods.

Our evaluation results show that the metrics we propose accurately capture the resource utilization state and can be used as part of VM placement algorithms with high satisfaction ratios. In majority of the cases, our methods perform much better than the other existing methods in terms of number of VM placement problem instances that can be solved successfully.

Second, we examine how to assign data traffic flows to the wireless network in a hybrid wireless data center (WDC). The developments in unlicensed 60 GHz ISM band communication enabled use of wireless networking in data centers.

60 GHz band offers high-bandwidth line-of-sight wireless communication over short distances [5]. Line-of-sight requirement and short communication distance are handicaps for a general-purpose wireless communication protocol, but they become advantageous in a densely-packed data center network (DCN) by reducing the interference with nearby concurrent communications, therefore increasing throughput across the data center [6]. IEEE 802.11ad standardizes the use of 60 GHz band [7, 8].

There are two approaches to using wireless networking in data centers: completely wireless and hybrid. In completely wireless data centers (WDCs), all communication between servers is wireless — there is no wired communication [9–15]. A completely wireless data center has a very different physical organization than a traditional data center. In hybrid wireless data centers, wireless communication is used to assist wired network [16–21].

In this thesis, we focus on hybrid wireless data centers because they are more applicable in short-term than completely wireless data centers. Existing data centers could be equipped with wireless networking devices with little effort. Top-of-the-rack (ToR) switches are good candidates for radio placement, so that racks can communicate wirelessly as well. Existing studies focus on using wireless resources to increase capacity at the bottlenecks of the wired network [16, 18, 20]. Increasing bandwidth at a bottleneck can be achieved with single-hop wireless communication.

Bottlenecks, also called hotspots, may occur between $5-10$ switches in a network of 1500 servers running a Map-Reduce job [16]. Other analyses of data center traffic find that even though core switches carry a higher traffic load than edge switches, edge switches have higher link loss [22, 23]. Alleviating hotspots can be achieved by routing the traffic to a neighboring ToR switch and forwarding it to the bottlenecked one using wireless communication, therefore increasing bandwidth only at the bottleneck [20]. An alternative is to connect two sets of hot servers directly over wireless network, using multiple radios when necessary [18]. Existing studies employ single-hop wireless communication.

We examine the problem of offloading as much traffic as possible from wired to wireless network, so that hybrid data centers could be designed accordingly. We aim to analyze capabilities of a multi-hop wireless network by quantifying the amount of traffic carried, multi-hop path length, and throughput in a data center setting. The results can be used by data center designers to design a hybrid wired and wireless network that is more efficient to build, operate, maintain, and expand than traditional data center network designs.

We propose multi-hop routing algorithms that assign traffic flows to wireless links in a hybrid data center. Traffic flows are evaluated in the ascending order of wireless hop distance between their source and destination. Source-destination pairs that are nearby are assigned to wireless links before the ones that are more distant to each other. The basic premise is to create longer routes between distant nodes from shorter routes that connect closer ones. The required wireless link configuration to assign a new flow may conflict with the existing configuration. In that case, the traffic flowing over the conflicting links may need to be deallocated. A cost-benefit analysis determines the result. The wireless link configuration that carries more traffic is preferred. In other words, allocation is greedy with respect to traffic amount.

Our proposed algorithms are run periodically, to assign traffic according to changing traffic needs of the data center. At the beginning of each period, an external traffic estimator outputs expected traffic exchange between ToR switches during that period. Our algorithms take this estimate as input, and output a configuration of wireless links. Reconfiguring wireless links costs bandwidth, because the traffic that has no route to its destination in the new configuration needs to be dropped. In addition, broadcasting the new configuration and making sure that all nodes has finished configuration takes time, during which the wireless network cannot be used efficiently. Therefore, we aim to maximize the amount of traffic carried for a given configuration. The traffic that is not assigned to wireless network, flows over the wired network as usual. Our extensive simulation results show how much traffic can be offloaded to wireless network, so that data center networks could be designed accordingly.

We summarize contributions of this thesis as follows. For server resource allocation:

- We define two design principles to assess the quality of an allocation so that allocation alternatives could be compared in a consistent manner.
- Using these design principles, we propose two novel metrics that quantify the quality of an allocation.
- The metrics we propose are parametric so that they could be adapted to the specific environment in which they will be used.
- Our metrics are suitable to be used with different allocation methods. We demonstrate this in the thesis by proposing resource allocation methods that use our metrics.
- We evaluate the performance of our proposed methods using a benchmark. Results show that our methods perform better than the ones in the literature.

For wireless network resource allocation:

- We define a practical system model for hybrid wireless data centers.
- We propose multi-hop routing algorithms that utilize the wireless infrastructure to its potential.

- We propose a method to randomly generate data center traffic based on a real data center traffic pattern.

- We verify and evaluate our proposed methods with simulations.

1.1 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 gives an overview of existing VM allocation and WDC work in the literature. Chapter 3 defines multidimensional server resource allocation problem and presents our proposed metrics. Chapter 4 proposes resource allocation heuristic methods that use our proposed metrics, and discusses experiments and results. Chapter 5 defines hybrid wireless data center system model, its rationale, and proposes methods for assignment of traffic flows to multi-hop wireless routes. Chapter 6 gives our proposed random traffic generation method, presents simulation experiments, and discusses results. Finally, Chapter 7 concludes the thesis and discusses future work.

Chapter 2

Related Work

In this chapter, we present related work in the literature and compare it to our thesis. We present the studies related to virtual machine allocation in Chapter 2.1 and the studies related to hybrid wireless data center networking in Chapter 2.2.

2.1 Server Resource Allocation

We categorize server resource allocation literature into four main categories: network-aware, energy-aware, service level agreement (SLA) based, and utilization-focused methods.

2.1.1 Network-aware

A traffic-aware VM placement strategy is presented in [24]. Authors formulate an optimization problem and prove its hardness. They provide a heuristic algorithm that solves the problem efficiently for large problem sizes. The algorithm first clusters the set of VMs and PMs. Then it matches VM clusters with PM clusters, finally assigning individual VMs to PMs. Experiments evaluate the efficiency of the allocation algorithm for various data center network architectures. A similar study is presented in [25], where communication dependencies and network topology is incorporated as cost metrics into migration decision.

Another network based allocation method is presented in [26]. The study considers a multiple data center environment and proposes algorithms that minimizes latency between VMs of the same request hosted at different data centers, as well as algorithms that minimize inter-data center traffic and inter-rack traffic within a data center. The data center selection problem under presented model is shown to be NP-hard, and a 2-approximation algorithm is described. Datacenter network is assumed to have a tree topology. Machine selection aims to allocate VMs so that height of the communication tree is minimum.

FairCloud [27] defines three cloud network service requirements: min-guarantee, high utilization, and network proportionality. Min-guarantee states that a cloud user should be able to get a guaranteed minimum bandwidth. High utilization requires available bandwidth to be used when needed. Network proportionality means that bandwidth should be distributed among cloud users proportionally to their payments. The study defines trade-offs between these requirements and presents allocation algorithms.

To the best of our knowledge none of the allocation methods that focus on network resources consider multidimensionality of the PM resources. They assume a certain VM capacity for PMs, neglecting request and workload requirements of different VMs. In our thesis, we consider network resources just like others, such as CPU or memory.

2.1.2 Energy-aware

A genetic algorithm (GA) based task scheduling optimization for both makespan and energy consumption metrics is proposed in [28]. Because there are two objectives, a solution is not unique, but is a set of Pareto points. The user is able to choose the right amount of trade-off between makespan and energy consumption among those points. The study is based on energy-conscious scheduling (ECS) heuristic proposed in [29], which is a greedy scheduling algorithm considering makespan and energy consumption. Proposed method uses GA to find Pareto optimal points among solutions to ECS instances. Multiple evolutionary algorithms are run in parallel. Asynchronous migrations of solutions between parallel-running instances enable exploring a larger solution space.

A concise survey of energy aware studies in grid and cloud computing is presented in [30]. Authors also define general principles in energy-conscious cloud management. The study defines algorithms for initial placement and dynamic migration of VMs to decrease energy consumption due to CPU utilization. Different from [30], our thesis focuses on multidimensional resources and management of heterogeneous workloads. These aspects of energy aware resource allocation are stated as open challenges in the referenced paper.

Whereas many energy-aware studies focus on CPU power consumption, [31] also consider energy consumption by network resources while maximizing load-balancing and resource utilization. Their proposed method is an extension to Knee Point-Driven Evolutionary Algorithm (KnEA) [32], which is a many-objective optimization problem.

In our thesis, we do not use an energy model to reduce energy consumption. However, as a consequence of packing VMs into fewer PMs we address energy consumption concerns indirectly.

2.1.3 SLA-based

MorphoSys [33] describes a colocation model for SLA-based services. SLA model captures periodic resource requirements of requests. The study uses first fit and best fit heuristics for resource allocation in a homogenous environment. Two cases are considered for allocations: Workload Assignment Service allocate resources to requests, while Workload Repacking Service migrates VMs such that resources are used more efficiently. Different repacking and migration policies are also discussed. The system finds alternative allocations by transforming SLAs into equivalent forms, in case they do not fit into any of the PMs in their original forms.

An artificial neural network approach to predict workload for two different types of applications (high performance computing and Web applications) and allocate VMs accordingly is presented in [34]. Resource usage is monitored live so that any violations of service level agreements because of errors in prediction could be resolved by migrating VMs to a better PM. The study focuses on homogeneous cloud infrastructures and uses only one resource dimension, CPU.

2.1.4 Utilization-focused

A two-step resource allocation process, as inspired by Eucalyptus cloud platform [35], is proposed in [36]. First, a cluster is selected within the cloud, then a node is selected within the cluster. Combinations of three cluster selection and six node selection heuristics are compared. Heuristics consist of the ones used in Eucalyptus, and those inspired by online bin packing literature. Experiment results show that cluster selection yields statistically significant difference in the average fraction of VMs obtained. Node selection methods, on the other hand, do not produce significant difference.

An end-to-end load balancing strategy between machine and storage virtualization for data centers is described in [37]. Proposed VectorDot algorithm extends Toyoda heuristic [38] for multidimensional single knapsacks to dynamic case of multiple knapsacks. VMs above a certain load threshold are migrated to PMs, so that the dot product of resource vector of the VM and those of required network resources is minimized, therefore achieving a lower cost migration.

Quantifying load imbalance on PMs and the overall system is discussed in [39]. Load imbalance of an individual server is defined to be a weighted sum of the imbalances for each resource type. Load imbalance of the system is defined as coefficient of variation in load distribution. The study also describes a greedy algorithm to balance load among servers. When the load of a server exceeds a threshold, VMs hosted on that server are chosen as migration candidates. The system imbalance values resulting from the migration of every candidate VM to every PM is calculated. The migration that achieves least imbalance is applied. As shown in experiment results presented in [40], and reproduced in our thesis, a metric of weighted sum of dimensions is inferior to others.

Sandpiper [41] is a monitoring and profiling framework to detect and remove hotspots by migrating VMs. The gray-box approach cooperates with VMs in determining workloads, while the black-box method does not require an integration with VMs. The study uses the inverse of available resource volume as a metric of multi-dimensional load. However, this approach has problems as explained in [42].

Anomalies of methods in existing VM placement literature is presented in [42]. Based on these anomalies, they define properties of a good VM allocation and propose algorithms for static VM placement and dynamic VM placement with load balancing or server consolidation goals. Authors define properties of a good allocation as follows: it should capture the shape of the allocation, it should use total remaining capacity as well as remaining capacity of individual resources, it should consider overall utilization as well as utilization of individual dimensions. Our proposed methods obey these rules. Their proposed method uses planar resource hexagon that consists of triangles representing different resource utilization categories. VMs are allocated to PMs in complementary resource triangles (CRT), so that overall utilization is tried to be balanced.

A similar idea of placing complementary VMs in the same PM is used in [43]. Complementary VMs are chosen by profiling the performance requirements in terms of resources and time. VMs that use the same resource at different times, or those that use different resources at the same time are considered complementary. VMs are allocated to PMs for which the remaining capacity weighted by weights assigned to resource dimensions is minimum.

Rather than explicitly finding VMs with complementary resource requirements, we focus on designing a good fitness function, minimization or maximization of which will have the same effect.

Other methods model VM placement as a bin packing problem. For single dimensional bin packing there are efficient heuristics to approximate the optimal solution. *First-fit decreasing* [44] is a popular and very simple such heuristic. It sorts the items to be placed in decreasing order. Beginning from the largest item, it places them in the first bin they fit, opening a new bin if none of the existing ones are suitable. This heuristic does not use more than 11/9 OPT + 1 bins, where OPT defines the value of an optimal solution.

Generalizing the first-fit decreasing heuristic to vector packing case is not trivial. Different methods are presented in [45]. The dot product metric that we used in our experiments is proposed in [46].

The performances of greedy, LP-based, genetic, and vector packing algorithms are compared in [47]. Even though the authors only consider clouds with homogeneous resources, they conclude that vector packing approaches are superior to others.

Vector bin packing with heterogeneous bins (VBPHB) problem is formally defined in [40]. A weighted sum of dimensions of vectors are proposed as a method of ordering multi-dimensional items and bins. By using different weights and calculating the weighted sum statically or dynamically, various measures are defined. These measures are used in combination with item- and bin-centric allocation heuristics as well as balancing-focused ones. Authors present a benchmark that consists of five classes of problem instances with different item and bin properties. Combinations of proposed measures and heuristics are evaluated on this benchmark. Authors also apply their theoretical work to a real-world machine reassignment problem. We compare our novel methods with the ones presented in [40]. Results show that our methods perform much better in the majority of the cases.

A survey of VM placement schemes are presented in [48]. According to their taxonomy, our method can be considered as a resource-aware bin packing-based method for heterogeneous environments. According to another survey of resource provision algorithms in cloud infrastructures [49], our approach classifies as a bin packing method for server selection with objectives of node cost minimization, energy efficiency, and utility maximization.

There is a need for a VM allocation method for heterogeneous cloud infrastructures, because clouds rarely consist of homogeneous resources. As cloud computing develops, ever more types of computing resources are offered to customers, such as SSD storage, GPUs, application specific integrated circuits (ASICs); therefore, the allocation method should support an arbitrary number of dimensions. We know that simple weighted sum of resource dimensions is not good enough. We improve upon these points.

2.2 Hybrid Wireless Data Center Networking

An analysis of data center traffic between ToR pairs is presented in [16]. Authors argue that only a few ToR pairs exchange very high amount of traffic at a given time, therefore it is an overkill to design an oversubscribed wired network. Rather, wireless communication can be used on demand to increase the capacity at congested points. The idea of allocating more resources at necessary points is called flyways. Flyways are applied to real data center environment in [20]. Authors measure capabilities of 60 GHz communication in a data center environment. Based on the results, they propose a system that uses one radio per ToR switch to offer additional bandwidth from one of the neighboring racks in case of network congestion. Authors also propose several methods to determine which wireless flyways to establish based on the traffic demand in the network.

We use a similar deployment of radios as in [16] and [20], except we propose two radios per ToR switch as presented in Chapter 5.1.

The same problem of alleviating hotspot in the wired network with a different method in [18] and [19]. Rather than using one radio per ToR switch, a set of servers are grouped into so called Wireless Transmission Units (WTUs). Authors note that although a WTU may correspond to a rack in certain data center architectures, the idea of WTU generalizes to other architectures as well. The problem is to schedule wireless links between WTUs according to a utility value based on the distance between nodes and traffic demand. Min-max and best-effort scheduling methods are proposed as solutions. Results show that both methods perform similarly when the traffic distribution is unbalanced across the network. When the traffic is uniform, min-max performs better than best-effort scheduling.

Wireless networking equipment is deployed differently in [21] and [50]. Because 60 GHz is restricted to line-of-sight communication, a signal reflecting surface is installed above the antennas so that they do not block the line of sight between other racks. Nodes are no longer restricted to communicating to their immediate neighbors as in [16,18], and [19], therefore the system achieves better performance.

Wireless networking in DCNs is also used for other purposes. Wireless communication as a facilities network in the data center in [51]. Authors argue that wireless network is more suitable for the control plane of Software Defined Networking [52, 53] and management tasks of cloud provider, rather than enhancing capabilities of the data plane. Another study [54] addresses the concern between control and data plane separation for a multiple-input multiple-output (MIMO) wireless DCN setting. Authors propose to replace wires between rows of racks with a MIMO wireless crossbar. Another take on MIMO wireless data center networking is presented in [55]. Carrying multicast traffic in wireless DCNs is examined in [56]. Another multicast traffic management approach that uses multiple channels is presented in [57].

Surveys of using wireless communication in DCNs can be found in [58] and [59].

In this thesis, our motivation is to discover the potential of wireless infrastructure when it is used with multi-hop routing. Studies [21] and [50] acknowledge that multi-hop communication could be used as an alternative to their method.

They consider the decreased throughput because of half-duplex communication as a potential drawback. We address this issue by using one-way communication in wireless links, so that full bandwidth of the channel could be used. Because the data center traffic displays an asymmetric traffic pattern, as presented in Chapter 6.1, the lack of the reverse communication direction is not important. The other drawback they present is the latency introduced by multi-hop forwarding, nevertheless they do not quantify it. In this thesis, we show that the latency could be kept as low as few hops even in large networks while carrying a big portion of the traffic wirelessly.

We note, however, that communicating over a reflective surface and over multiple hops are not mutually exclusive methods. To the contrary, these methods complement each other. By using a reflective surface, each hop could reach farther physical distances in the data center floor, therefore the benefits of multi-hop communication would be augmented.

Chapter 3

Metrics for Allocation of Server Resources

We first investigate the problem of allocating server resources (CPU, memory, storage, IO, etc.) to virtual machine requests.

3.1 Multidimensional Resource Allocation Problem

Allocating resources to virtual machines can be considered from different perspectives and at different levels of complexities.

According to the way requests are processed, resource allocation can be performed online (incrementally) or offline (in batches). Online methods consider each VM request one-by-one as they arrive. A request is allocated to the best PM according to the current state of the system resources. Offline algorithms process a set of requests and resources are allocated accordingly. In the ideal case, an offline method knows all requests before allocation starts.

In practice, request information is not available to the system beforehand, because VM requests arrive dynamically. Therefore, we consider offline allocation as batch allocation where resource requests arrive in sets. These sets may consist of dependent or independent requests. Dependent requests consist of VMs that communicate as part of an operation. VMs that do not collaborate are considered independent. Online and offline algorithms are interchangeable. Requests can be buffered to be processed in batches, or batch requests could be allocated individually. Online algorithms are simple in expression but may not achieve the same resource efficiency as offline algorithms. Offline algorithms can be more sophisticated and efficient, but delaying requests to form a batch may not be practical.

Virtualization technologies enable VMs to migrate between PMs. Although migration does not interrupt VM operation, it causes delay and overhead. Migration costs bandwidth, processing and IO operations to the cloud operator. Therefore, some data centers do not use migration. In those data centers VMs will be *static*, running on the same PM until termination. As VMs are created and terminated, the resources can be fragmented because different VMs may have different resource requirements. When VMs are *dynamic*, they can be migrated to more suitable PMs to improve allocation efficiency. However, the cost of migration should be accounted into the decision to migrate. More migrations than necessary would decrease overall performance and consume network resources.

We define resource allocation problem as a multi-objective optimization problem. Resource allocation aims to improve energy efficiency, resource utilization, load balancing, and server consolidation. NP-hard nature of the problem makes it infeasible to search the problem space efficiently to find an optimal solution. A utility function that combines different objectives into a single value is usually necessary to order alternative solutions. Heuristic or metaheuristic methods can be used to find practically good enough results. Heuristic algorithms run faster and therefore could be preferable for real time tasks.

In this thesis, we focus on request-oriented static VM allocation. We model the problem in a similar manner to the well-known bin packing problem, which is an NP-hard combinatorial optimization problem. In a bin packing problem, there are n items, i, of different sizes, w_i. These items are placed into at most m bins, j, with a capacity, C. The objective is to minimize the number of bins used while placing the items. The bin packing problem is formalized with the following integer linear program:

$$\text{minimize} \sum_{j=1}^{m} y_j \tag{3.1}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_{ij} \leq C y_j, \quad 1 \leq j \leq m \tag{3.2}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad 1 \leq i \leq n \tag{3.3}$$

where binary decision variable x_{ij} defines whether item i is placed in bin j, and y_j defines whether bin j is used.

In server resource allocation problem, VMs correspond to items and PMs correspond to bins. The original bin packing model requires some modifications in server resource allocation context. First of all, properties of server resources cannot be captured with a single weight value. Resources are multidimensional, best defined as a vector of values. Secondly, data centers are heterogeneous environments. Hardware resources have different capacities. We modify Equation 3.2 as in Equation 3.4 to satisfy these additional requirements:

$$\sum_{i=1}^{n} w_{ik} x_{ij} \leq C_{jk} y_j, \quad 1 \leq j \leq m, \ 1 \leq k \leq d \tag{3.4}$$

where d is the number of resource types (dimensions). This version of the problem is called *vector bin packing with heterogeneous bins* (VBPHB) [40], which is an extension of the *vector bin packing*, or simply *vector packing* problem, defined with the constraint given in Equation 3.5 instead of Equation 3.4:

$$\sum_{i=1}^{n} w_{ik} x_{ij} \leq C y_j, \quad 1 \leq j \leq m, \ 1 \leq k \leq d \tag{3.5}$$

The difference is how bin capacities are defined. In vector bin packing, each bin has the same capacity, C, in all resource dimensions. Vector bin packing formulation cannot directly model PMs, because CPU, memory, and storage resources of a physical machine cannot be represented by a single value. Therefore, a vector bin packing-based solution should propose a method to represent capacities of all dimensions by a single capacity value. One option is to use a weighted sum of the capacities of all dimensions [45]. However, using a weighted sum poses other problems, such as how to choose weights.

Rather than trying to adapt the problem to be modeled as a vector bin packing problem, we prefer using the well-suited VBPHB formulation, in which each bin j has a different capacity in dimension k, C_{jk}. VBPHB is reducible to vector bin packing [40], therefore which formulation is used is a matter of practical concern rather than a theoretical one.

In this thesis, we present our own vector packing approach based on *fitness* functions, i.e., *utilization metrics*. A fitness function maps each point of a multi-dimensional resource utilization space to a single numeric value, so that different points in space can be compared easily.

3.2 Proposed Utilization Metrics

In this section, we propose utilization metrics, i.e., fitness functions. These metrics quantify how good an allocation is. Chapter 4 explains how these metrics can be used as part of an allocation method. We begin with preliminary definitions. Then we introduce our proposed design guideline for utilization metrics. We propose our metrics and introduce some other from the literature.

3.2.1 Definitions

We consider a vector-based resource allocation model based on the following definitions from [42]:

- *Normalized Resource Cube* (NRC): A unit cube representing resources of a PM. Each dimension of the cube represents a resource type such as CPU, memory, disk space, I/O performance. All other normalized vectors are located inside the unit cube. When a PM or VM has d different resources, we consider the problem as d-dimensional resource allocation problem.
- *Total Capacity Vector* (TCV): The vector along the principal diagonal of an NRC. It is equal to the vector $\mathbf{1^d}$.
- *Resource Requirement Vector* (RRV): The amount of resources that are required by a VM request. RRVs are not normalized when considered by themselves. However, in the context of an NRC, they are assumed to be normalized by the scaling factor of the NRC.
- *Resource Utilization Vector* (RUV): The amount of utilized resources inside an NRC. RUV is the sum of RRVs of VMs that are hosted in a PM.
- *Remaining Capacity Vector* (RCV): The amount of available resources inside an NRC. ($\mathbf{RCV} = \mathbf{TCV} - \mathbf{RUV}$)
- *Resource Imbalance Vector* (RIV): The difference between RUV and its projection onto TCV.

Figure 3.1 pictures an NRC with its vectors. The number of dimensions of the vector space is equal to the number of resource types. In this case, a three dimensional space is depicted with resources x, y, and z. Note that adding an RRV can only increase the utilization. Therefore, the RUV always approaches towards full utilization planes $x = 1, y = 1, z = 1$. The best utilization is when $\mathbf{RUV} = \mathbf{TCV}$, in other words, when the sum of allocated RRVs is $(1, 1, 1)$.

3.2.2 Design Criteria

It is clear that achieving the full utilization point is preferable over suboptimal allocations. However, it is not easy to compare two arbitrary suboptimal allocations. Figure 3.2 illustrates a two dimensional vector space (a square) with

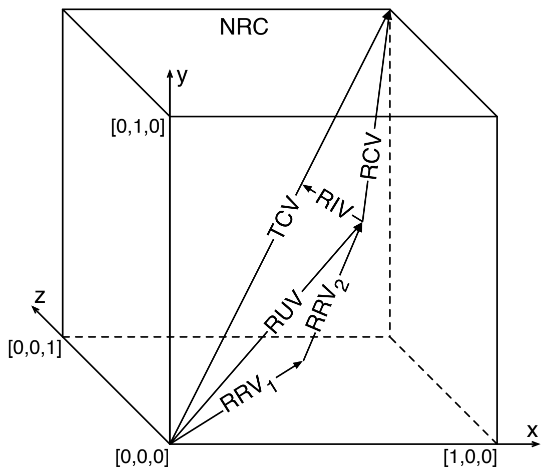


Figure 3.1: Normalized Resource Cube (NRC) in vector model for resource, request, and allocation representation.

two utilization vectors. Two RUVs may consist of two different sets of RRVs, which may share identical subsets. When allocating resources, we need to decide whether RUV_1 or RUV_2 is a better allocation. Comparing RUVs is the same as comparing corresponding RCVs. We prefer an RCV-based comparison, because focusing on the availability of resources seems semantically more apt than focusing on the used amount.

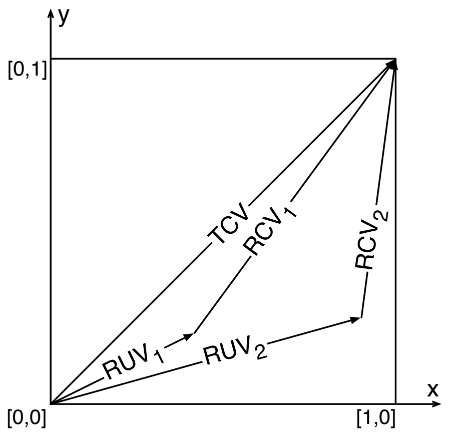
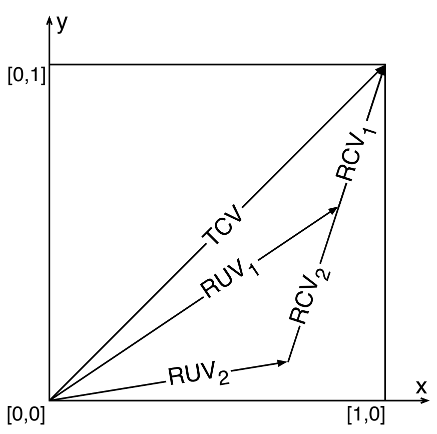


Figure 3.2: Vector comparison.

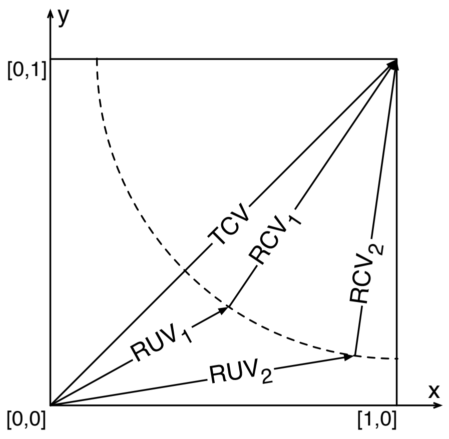
We would like to achieve the best utilization. A naive approach is to prefer the allocation alternative with smaller Euclidean distance between RUV and TCV, which corresponds to $\|\mathrm{RCV}\|$, the magnitude of RCV. According to this metric, $\|\mathrm{RCV}_\mathbf{2}\|$ is smaller and therefore is a better allocation than $\|\mathrm{RCV}_\mathbf{1}\|$ in Figure 3.2.

However, RUV_2 is very close to fully utilizing resource x, while leaving resource y relatively underutilized. On the other hand, RUV_1 utilizes both resource types in a more balanced manner. We conclude that the angle between TCV and RCV is an important factor in comparison.

Simply prioritizing either the angle or the magnitude would not work. For two RCVs with the same angle, the one with smaller magnitude is closer to full utilization, therefore is better (Figure 3.3a). For two RCVs with the same magnitude, the one with the smaller angle yields better balance of resource types, therefore is better (Figure 3.3b). Hence, our design criteria is: *a good fitness function should distinguish allocations based on both angle and the magnitude of the RCV.*



(a) RCV_1 is better.



(b) RCV_1 is better.

Figure 3.3: Comparison of two RCVs with the same angle or magnitude.

3.2.3 Our Metrics

According to our proposed design criteria, we propose two parametrized metrics: TRfit and UCfit.

3.2.3.1 Our First Metric: TRfit

A fitness function f of RCV determines the suitability of an allocation. A fitness function defines a total order in the allocation vector space, so that allocation alternatives can be compared. We are interested in a fitness function that achieves an efficient allocation. For such purpose, we propose the fitness function defined in Equation 3.6. It is called *TRfit*, short for TCV-RCV fitness. It considers total capacity vector and remaining capacity vector. It has one parameter, α.

$$f_{\text{TRfit}}(\mathbf{RCV}) = \frac{\|\mathbf{RCV}\|}{\cos^{-1}\left(\frac{1}{\sqrt{d}}\right) - \theta_{\mathbf{TCV}}(\mathbf{RCV}) + \alpha} \tag{3.6}$$

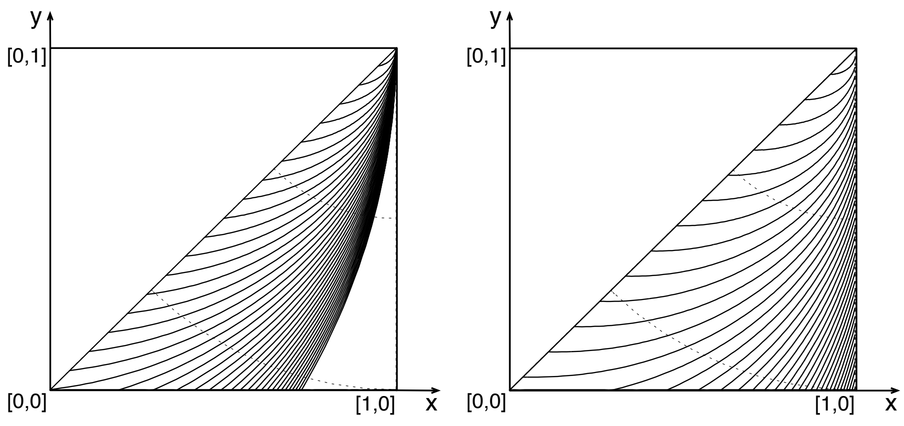
where $\theta_{\mathbf{TCV}}(\mathbf{RCV})$ is the angle between TCV and RCV, calculated by Equation 3.7:

$$\begin{aligned}\theta_{\mathbf{TCV}}(\mathbf{RCV}) &= \cos^{-1}\left(\frac{\mathbf{TCV}\cdot\mathbf{RCV}}{\|\mathbf{TCV}\|\|\mathbf{RCV}\|}\right) \\ &= \cos^{-1}\left(\frac{1}{\sqrt{d}}\cdot\frac{\sum_{i=1}^{d}\mathbf{RCV}[i]}{\sum_{i=1}^{d}(\mathbf{RCV}[i])^2}\right)\end{aligned} \tag{3.7}$$

α is a parameter that results in fitness functions with different properties, and d is the number of resource types (number of dimensions).

Figure 3.4a and Figure 3.4b depict isometric curves of fitness functions with $\alpha = 0$ and $\alpha = \pi/12$ respectively. RCVs on the same curve have the same fitness value. Figures show that our fitness function obeys the design guide described above in Chapter 3.2.2. Among the two RCVs with the same angle to the TCV, the one with less magnitude has better fitness value. Among two RCVs with the same magnitude (the ones that are on the dashed lines), the one with smaller angle to the TCV has better fitness value. In our definition of fitness function, *lower* values denote *better* fitness.

An important property of this function is the fact that the same fitness value achieved by allocating a new RRV leads to increasingly higher utilization in all dimensions. In this sense, α is the parameter that defines the amount of increase in utilization. For $\alpha = 0$, additional allocations with the same fitness value lead to full utilization point. For $\alpha = \pi/12$, the highest achievable utilization for a given



(a) $\alpha = 0$ (b) $\alpha = \pi/12$

Figure 3.4: TRfit fitness function with different parameters.

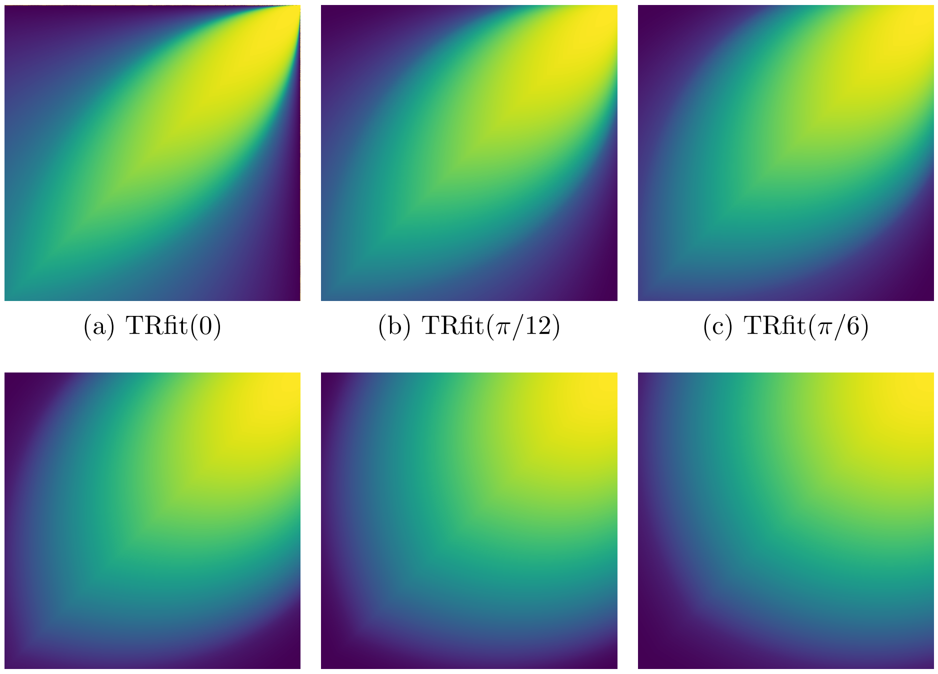
fitness value is suboptimal. Figure 3.5 visualizes TRfit in a heat map for various values of α. A heat map renders the multidimensional resource space in different colors, each corresponding to a different fitness value. Smaller (better fitness) values are represented by lighter colors, while higher (worse fitness) values are represented by darker colors. The points of the same color have the same fitness value.

3.2.3.2 Our Second Metric: UCfit

Many fitness functions conform to our proposed design criteria. We present another fitness function that is an alternative to TRfit in Equation 3.8. We call it BasicUCfit, short for utilization-capacity fitness. It considers both resource utilization and remaining capacity. It is, again, a function of RCV. BasicUCfit combines the Euclidean distance to full utilization point, $\|\text{RCV}\|$, and the angle between RUV and RCV, θ.

$$f_{\text{BasicUCfit}}(\mathbf{RCV}) = \|\mathbf{RCV}\| \cdot \sin\big(\theta_{\mathbf{RUV}}(\mathbf{RCV})\big) \tag{3.8}$$

Figure 3.6a visualizes euclidean distance to full utilization point. Figure 3.6b visualizes sine of the angle between RUV and RCV. Multiplication of both, yields

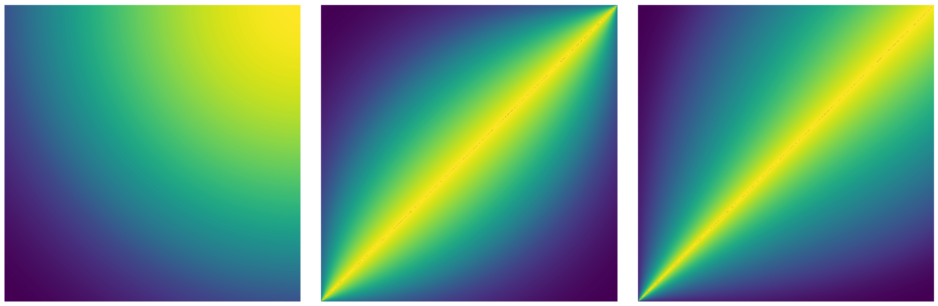


(a) TRfit(0) (b) TRfit($\pi/12$) (c) TRfit($\pi/6$)

(d) TRfit($\pi/3$) (e) TRfit($2\pi/3$) (f) TRfit($4\pi/3$)

Figure 3.5: Visualization of TRfit for different parameter values.

BasicUCfit and is visualized in Figure 3.6c. Euclidean distance component contributes to the fitness value more around the full utilization point. Fitness value of points farther from the full utilization point are dominated by the sine component.



(a) $\|\text{RCV}\|$ (b) $\sin(\theta)$ (c) $\|\text{RCV}\| \cdot \sin(\theta)$

Figure 3.6: BasicUCfit heat map.

It would be better if contributions of each component of the BasicUCfit function could be adjusted. Equation 3.9 introduces its parametrized version, *UCfit*.

It has three parameters a, b, c. Parameter a adjusts the contribution of the Euclidean distance component, and b adjust the contribution of the sine term. An important difference of UCfit from BasicUCfit is the normalization of the Euclidean distance with the magnitude of TCV. Normalizing the Euclidean distance brings its range to $[0, 1]$, the same range as the sine component, so that the values of parameters a and b are meaningful relative to each other.

Parameter c is an offset to the sine component. Points on the TCV have a fitness value of 0, because their sine component is 0 when the angle between RUV and RCV is 180 degrees. For any non-zero value of c, points on the TCV have non-zero fitness values. In addition, different points on TCV have different fitness values because they are at different distances to the full utilization point.

$$f_{\text{UCfit}}(\mathbf{RCV}) = \left(\frac{\|\mathbf{RCV}\|}{\|\mathbf{TCV}\|}\right)^{a} \cdot \Big(\sin\big(\theta_{\mathbf{RUV}}(\mathbf{RCV})\big) + c\Big)^{b} \tag{3.9}$$

Figure 3.7 shows heat maps for various a and b values of UCfit. Parameter c is zero in those figures.

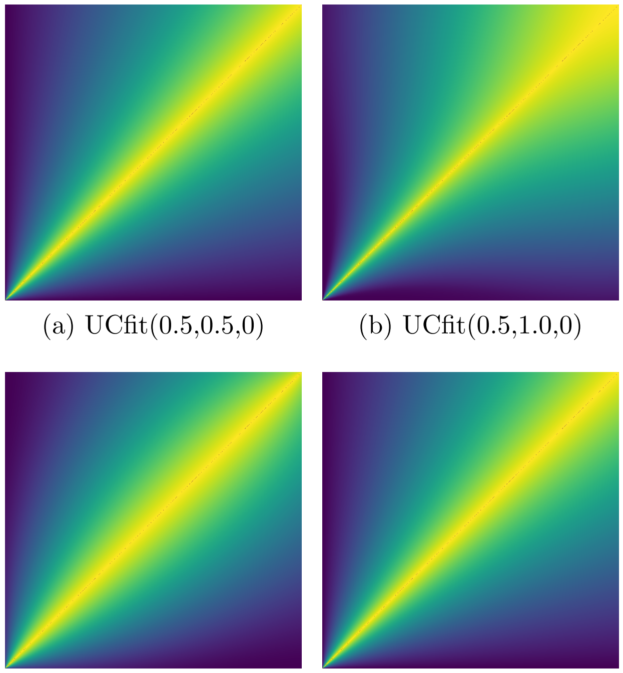
The angle θ is always between 90 and 180 degrees for all vector space dimensions. Therefore, UCfit applies equivalently well to all resource dimensions.

3.3 Other Metrics

In this section, we present some existing metrics from the literature. We will use these in our performance evaluation.

3.3.1 RIV Metric

Resource imbalance vector, RIV, can be used as a metric as well [42]. As defined in Chapter 3.2.1, it is the difference between RUV and its projection onto TCV. Similar to our proposed metrics, it can be expressed as a function of RCV. We


(a) UCfit(0.5,0.5,0) (b) UCfit(0.5,1.0,0)

(c) UCfit(1.0,0.5,0) (d) UCfit(1.0,1.0,0)

Figure 3.7: Visualization of UCfit for different parameter values.

define RIV metric as the magnitude of RIV, ‖RIV‖. Figure 3.8a gives the heat map for RIV metric.

3.3.2 SandFit Metric

Sandpiper [41] uses a metric based on the inverse of the available volume of resources. We refer to this metric as Sandpiper. It is a function of RCV, but its value is inversely proportional to RCV, as seen in its heat map given in Figure 3.8b.

3.3.3 R Metric

R is the Euclidian distance to full utilization point. It is a naive metric that can be used to maximize resource utilization. Formally, it is equal to ‖RCV‖.

Figure 3.8c shows its heat map.

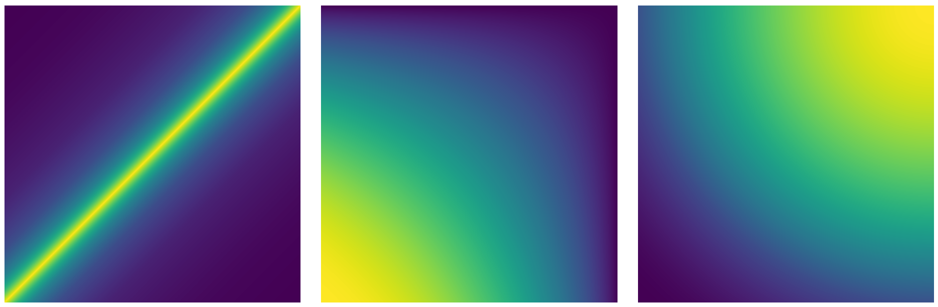

(a) RIV (b) Sandpiper (c) R

Figure 3.8: Heat maps for other metrics.

3.4 Summary

As a cloud computing service, Infrastructure as a Service (IaaS) operators provide public access to their infrastructures. Cloud customers use cloud resources via virtual machines, on which they can run arbitrary software. One of the challenges faced by providers in this type of cloud computing is efficient allocation of physical resources to VM requests so that the number of customers whose requests are satisfied can be increased.

In this chapter, we provide metrics that measure the goodness of an allocation of VMs into physical machines. These metrics can be used by an allocation algorithm to satisfy consumer VM requests in as many cases as possible. We propose two novel and generic resource utilization metrics, called TRfit and UCfit. We present other metrics from the literature and visualize all metrics in a 2D resource space. As opposed to the metrics from the literature, our metrics are parametric. Each different selection of a parameter enables a new sub-metric that can be more suited for particular cases.

Chapter 4

Algorithms for Allocation of Server Resources

In this chapter, we present our VM allocation algorithms that use our metrics proposed in Chapter 3.

4.1 Allocation Algorithms Based on Our Metrics

Our first method, presented in Algorithm 1, is an *online* algorithm that allocates a given VM instance request to a suitable PM among infrastructure resources. Main strategy is to allocate the VM to a PM that is switched on. If the VM does not fit into any of the switched-on PMs, a sleeping PM is woken up to allocate the VM. It may be the case that the request exceeds capacity of resources available on the infrastructure (including switched-off physical machines), in which case the request is denied.

Capacity constraints are ensured in Line 4. A PM could host a VM if and only if RRV of VM is less than or equal to RCV of PM (i.e., demand is less than remaining capacity for all dimensions). This is denoted by $\preceq$ symbol and formally defined in Equation 4.1:

$$\mathbf{u} \preceq \mathbf{v} \triangleq \mathbf{u_i} \leq \mathbf{v_i}, \forall i = 1 \ldots d \tag{4.1}$$

Similarly, $\prec$ symbol is defined in Equation 4.2:

$$\begin{gathered} \mathbf{u} \prec \mathbf{v} \triangleq \mathbf{u_i} \leq \mathbf{v_i}, \forall i = 1 \ldots d \text{ and} \\ \mathbf{u_j} < \mathbf{v_j}, \exists j = 1 \ldots d \end{gathered} \tag{4.2}$$

Allocation vectors of PMs (RUV, RIV, RCV, and TCV) are normalized, they range from $\mathbf{0^d}$ to $\mathbf{1^d}$, whereas RRVs of VMs are not. In order to be able to compare RCV and RRV, we scale RRV of VM by the normalization factor of PM. This operation is denoted with a subscript of the normalization factor on scaled vector, such as $\mathrm{RRV_{PM}(VM)}$.

When a sleeping PM is determined to host the request, Switch-On method is called to bring machine into operation. Basically, it removes the given PM from the set of sleeping machines, Off[Resources], and adds it to the set of machines in operation, On[Resources].

Online allocation runs in $O(m)$ time, where m is the number of resources.

We can also use our metrics in *offline* allocation methods that have a batch of VMs to place into a set of PMs. Next we describe an offline method using our metrics. But before describing it in detail, we introduce an auxiliary procedure, Algorithm 2, which allocates as much VMs as possible from a given set of requests onto a given PM. At each iteration, R contains requests that could fit into remaining space of the PM. This invariant is satisfied by the Lines 2 and 8. Among the requests in R the best fitting VM is allocated, until either all of the requests are allocated or there are no requests left that could fit into remaining space.

Arg min operation in Line 4 can be implemented by scanning the set of requests and keeping the running minimum of calculated fitness function and the VM for which it is minimum. This takes $O(n)$ time, where n is the number of requests.

Algorithm 1 Online VM allocation

```
ONLINE-ALLOCATE(Resources, VM)
1: bestOn ← $nil$
2: bestOff ← $nil$
3: for all PM ∈ Resources do
4:     if $\text{RRV}_{\text{PM}}(\text{VM}) \preceq \text{RCV}[\text{PM}]$ then
5:         rcv ← $\text{RCV}(\text{RUV}[\text{PM}] + \text{RRV}_{\text{PM}}(\text{VM}))$
6:         if On[PM] and $f(\text{rcv}) < f(\text{bestOnRCV})$ then
7:             bestOnRCV ← rcv
8:             bestOn ← PM
9:         else if Off[PM] and $f(\text{rcv}) < f(\text{bestOffRCV})$ then
10:            bestOffRCV ← rcv
11:            bestOff ← PM
12:        end if
13:    end if
14: end for
15: if bestOn $\neq$ $nil$ then
16:     RUV[bestOn] ← $\text{RUV}[\text{bestOn}] + \text{RRV}_{\text{bestOn}}(\text{VM})$
17: else if bestOff $\neq$ $nil$ then
18:     RUV[bestOff] ← $\text{RRV}_{\text{bestOff}}(\text{VM})$
19:     SWITCH-ON(Resources, bestOff)
20: end if
21: return $nil$
```

Similarly, eliminating VMs in Lines 2, 7, and 8 could be achieved with a single pass over R, therefore taking $O(n)$ time. Line 10 could be performed by iterating over allocated items and removing them from requests in $O(kn)$ time, where k is the number of requests that could be allocated to the PM. The while loop iterates k times allocating a single VM in each iteration; therefore, total time complexity of the algorithm is $O(n + kn)$. In case of an allocation, kn term dominates. In the worst case, all of the requests could be allocated, which takes $O(n^2)$ time.

Algorithm 3 is our offline method that allocates a set of VM requests to a set of physical machines. It chooses a suitable PM for a given VM. Our offline algorithm uses complementary strategy, choosing suitable VMs for each PM. The offline algorithm begins by allocating requests to PMs that are currently switched on. Switched-on machines are sorted in descending order of their fitness value, so that imbalanced PMs are given priority to choose the best fitting VM for them.

Algorithm 2 VM allocation to a given PM

```
FillPM(PM, Requests)
1: Allocated ← $\emptyset$
2: $R \leftarrow \{\mathrm{VM} \in \mathrm{Requests} \,|\, \mathrm{RRV}_{\mathrm{PM}}(\mathrm{VM}) \preceq \mathrm{RCV}[\mathrm{PM}]\}$
3: while $R \neq \emptyset$ do
4:     $\mathrm{BestVM} \leftarrow \arg\min_{\mathrm{VM} \in R} f(\mathrm{RCV}(\mathrm{RUV}[\mathrm{PM}] + \mathrm{RRV}_{\mathrm{PM}}(\mathrm{VM})))$
5:     $\mathrm{RUV}[\mathrm{PM}] \leftarrow \mathrm{RUV}[\mathrm{PM}] + \mathrm{RRV}_{\mathrm{PM}}(\mathrm{BestVM})$
6:     $\mathrm{Allocated} \leftarrow \mathrm{Allocated} \cup \{\mathrm{BestVM}\}$
7:     $R \leftarrow R - \{\mathrm{BestVM}\}$
       ▷ Remove VMs requesting larger than remaining capacity
8:     $R \leftarrow R - \{\mathrm{VM} \in R \,|\, \mathrm{RCV}[\mathrm{PM}] \prec \mathrm{RRV}_{\mathrm{PM}}(\mathrm{VM})\}$
9: end while
10: Requests ← Requests − Allocated
11: return Allocated
```

Remaining requests are allocated to sleeping/switched-off machines.

Sum of RRVs of remaining requests are calculated so that PMs with resource ratios similar to that of overall requests get allocated first. Resource ratio similarity is calculated by the fitness function. Normally, RCVs of different PMs are not comparable, because they are normalized by different factors. Therefore, we renormalize them with the sum of RRVs to be able to compare. Beginning from the best fitting PM to worst fitting one, offline algorithm uses Algorithm 2 to allocate VMs. Note that for practical reasons, it would be wise to return from the procedure immediately when all of the requests are satisfied. The algorithm terminates when there are no requests left to allocate, or when remaining requests would not fit into any of the PMs because of capacity constraints.

It is important to define the data structures of resources, On[Resources] and Off[Resources], properly. On[Resources] is the set of PMs that are switched on. We use a simple list to implement this set. We could use a simple list to also keep the set of sleeping PMs, Off[Resources], but that would cause redundancy; instead, we use a counted set. Switched off PMs of the same type can be represented by a single element, since all of them have the same amount of each resource. There are many fewer distinct types of PMs than there are individual PMs, therefore it would be better to keep the list of PM types along with the number of individual machines of that type. This approach would not be suitable for On[Resources], because individual PMs would have different allocations resulting in different residual capacities even though they belong to the same PM type. Therefore, they cannot be practically aggregated as a single element.

Algorithm 3 uses counted set operations while processing Off[Resources]. Sorting in Line 5 is done with respect to elements (PM types) regardless of associated counts (individual machines). For loop in Line 6 iterates over individual machines. Loop variable PM is set to PMs corresponding to each count of each element in R. Note that R is modified inside the loop, therefore the loop should be considered to get the first element at each iteration rather than passing over a static list.

When a PM gets allocated some requests, count of its type is decreased by one. The iteration continues with next instance of the same type, assuming the count has not reached zero. When count of an element reaches zero, it is removed from the set. If a PM could not be allocated any requests, none of the instances of the same type could be. Therefore, corresponding PM type element is removed from the counted set, and iteration continues with an instance of next PM type.

Modifications on R is formally defined as a set difference operation. For counted sets, set difference operator returns a set in which the associated counts of elements in the first set is decreased by the associated counts of corresponding elements in the second set. Elements that have a count of zero are assumed to be removed from the set. In Line 8 only a single instance of a PM type is differentiated from R, therefore causing the count of the PM type decrease by one. In Line 11 the count of the PM type is set to zero by differentiating the set of all instances of a certain type. Therefore, the type of the PM is removed from R. Counted set is denoted by a subscripted n on closing set brace.

Allocating requests to On[Resources] takes $O(t \log t + tn + k_{\text{on}} n)$ time, where $t = |\,\text{On[Resources]}|$ and k_{on} is the number of requests that could be allocated to On[Resources]. $t \log t$ term is due to sorting. tn term comes from iterating all requests for each PM. Time complexity of allocating k_{on} requests is independent of the number of PMs to which they are allocated, therefore it takes $O(k_{\text{on}}n)$ time.

Line 4 takes linear time in n. Line 5 takes $O(s \log s)$ time, where s is the number of PM types.

Allocating requests to Off[Resources] takes $O(sn + k_{\text{off}}(n + s))$ time, where k_{off} is the number of requests that could be allocated to Off[Resources]. Allocating k_{off} resources takes $O(k_{\text{off}}n)$ time. For each allocation, PM type is removed from R and Off[Resources] in $O(s)$ time. When no resources could be allocated to a PM type, the for loop iterates s times scanning the list of requests for a suitable VM; hence the term sn.

Overall time complexity of Algorithm 3 is $O(t \log t + tn + k_{\text{on}}n + s \log s + sn + k_{\text{off}}(n + s))$. Ordering the terms semantically, we obtain $O(t \log t + s \log s + n(t + s) + n(k_{\text{on}} + k_{\text{off}}) + sk_{\text{off}})$. To simplify the terms, we can consider a common configuration of the cloud where $s << t \leq m$, in which case the time complexity becomes $O(m \log m + n(m + k))$, where $k = k_{\text{on}} + k_{\text{off}}$.

Algorithm 3 Offline VM allocation.

```
Offline-Allocate(Resources, Requests)
 1: for all PM ∈ $\textsc{Sort}_{>f(\text{RCV})}$(On[Resources]) do
 2:     FillPM(PM, Requests)
 3: end for
 4: Total ← $\sum_{\text{VM} \in \text{Requests}} \text{RRV}[\text{VM}]$
 5: $R$ ← $\textsc{Sort}_{<f(\text{RCV}_{\text{Total}})}$(Off[Resources])
 6: for all PM ∈ $R$ do
 7:     if FillPM(PM, Requests) ≠ ∅ then
 8:         $R \leftarrow R - \{\text{PM}\}_n$
 9:         Switch-On(Resources, PM)
10:     else                                   ▷ PM cannot host any of the VMs
11:         $R \leftarrow R - \{r \in R \mid r \equiv \text{PM}\}_n$
12:     end if
13: end for
```

An alternative offline allocation algorithm is presented in Algorithm 4. It shares the outline of Algorithm 3, except it calculates the sum of remaining requests at each iteration and chooses the best fitting PM accordingly. Although costlier than the previous version, it handles high variance request sets better, since remaining requests would result in a different overall resource balance after each allocation to a PM. In terms of time complexity, $s \log s$ term is replaced by s^2 term because of the arg min calculation in each iteration of the while loop. The other difference, Line 13, takes $O(k_{\text{off}} n)$ time in the aggregate, therefore doesn't affect the time complexity because $k_{\text{off}} n$ term is already accounted for. Time complexity of the whole algorithm is, therefore, $O(t \log t + s^2 + n(t+s) + n(k_{\text{on}} + k_{\text{off}}) + sk_{\text{off}})$. Common case still runs in $O(m \log m + n(m+k))$, assuming that $s^2 < m \log m$, which is a safe assumption for large-scale cloud infrastructures.

Algorithm 4 Offline VM allocation alternative.

```
OFFLINE-ALLOCATE(Resources, Requests)
 1: for all PM ∈ $\text{SORT}_{>f(\text{RCV})}$(On[Resources]) do
 2:     FILLPM(PM, Requests)
 3: end for
 4: Allocated ← $\emptyset$
 5: $R$ ← Off[Resources]
 6: Total ← $\sum_{\text{VM} \in \text{Requests}} \text{RRV}[\text{VM}]$
 7: while $R \neq \emptyset$ or Total $\neq 0$ do
 8:     BestPM ← $\arg\min_{\text{PM} \in R} f(\text{RCV}_{\text{Total}}[\text{PM}])$
 9:     Allocated ← FILLPM(BestPM, Requests)
10:     if Allocated $\neq \emptyset$ then
11:         $R \leftarrow R - \{\text{BestPM}\}_n$
12:         SWITCH-ON(Resources, BestPM)
13:         Total ← Total $- \sum_{\text{VM} \in \text{Allocated}} \text{RRV}[\text{VM}]$
14:     else                                ▷ BestPM cannot host any of the VMs
15:         $R \leftarrow R - \{\text{PM} \in R \mid \text{PM} \equiv \text{BestPM}\}_n$
16:     end if
17: end while
```

If all PMs are turned on and we have all VM requests in hand to place, we can look to each possible VM-PM pair while placing VMs into PMs to find out the best metric value. More specifically, we can consider VMs one by one and for each VM we can calculate the fitness value of placing that VM to each of the PMs and select the best fitting PM, and place the VM there. Then we can continue with the next VM in the list. The pseudocode of the method is shown in Algorithm 5. This is the algorithm that we use to compare various metrics in the evaluation section of this chapter. We use this algorithm in our evaluations to do a fair comparison with other methods in literature [40].

Fitness values are initialized in $O(mn)$ time. The while loop iterates k times by definition, because minimum of fitness values is ∞ when there are no more requests that could be allocated to any PM. Each iteration takes $O(mn)$ time due to arg min operation in Line 12 which considers all VM-PM pairs. Therefore, the whole algorithm runs in $O(kmn)$ time.

Algorithm 5 Offline VM allocation considering all pairs.

```
Allocate(Resources, Requests)
1: for all PM ∈ Resources do
2:     for all VM ∈ Requests do
3:         if $\mathrm{RRV}_{\mathrm{PM}}(\mathrm{VM}) \preceq \mathrm{RCV}[\mathrm{PM}]$ then
4:             $\mathrm{rcv} \leftarrow \mathrm{RCV}(\mathrm{RUV}[\mathrm{PM}] + \mathrm{RRV}_{\mathrm{PM}}(\mathrm{VM})))$
5:             $\mathrm{fitness}[\mathrm{PM},\mathrm{VM}] \leftarrow f(rcv)$
6:         else
7:             $\mathrm{fitness}[\mathrm{PM},\mathrm{VM}] \leftarrow \infty$
8:         end if
9:     end for
10: end for
11: while $\min\{\mathrm{fitness}[\mathrm{PM},\mathrm{VM}] \mid \forall\,\mathrm{PM} \in Res, \forall\,\mathrm{VM} \in Req\} \neq \infty$ do
12:     $\mathrm{BestPM}, \mathrm{BestVM} \leftarrow \operatorname*{arg\,min}_{\mathrm{PM}\in Res, \mathrm{VM}\in Req} \mathrm{fitness}[\mathrm{PM},\mathrm{VM}]$
13:     $\mathrm{RUV}[\mathrm{BestPM}] \leftarrow \mathrm{RUV}[\mathrm{BestPM}] + \mathrm{RRV}_{\mathrm{BestPM}}(\mathrm{BestVM})$
14:     $\mathrm{Requests} \leftarrow \mathrm{Requests} - \{\mathrm{BestVM}\}$
15:     for all VM ∈ Requests do
16:         if $\mathrm{RRV}_{\mathrm{BestPM}}(\mathrm{VM}) \preceq \mathrm{RCV}[\mathrm{BestPM}]$ then
17:             $\mathrm{rcv} \leftarrow \mathrm{RCV}(\mathrm{RUV}[\mathrm{BestPM}] + \mathrm{RRV}_{\mathrm{BestPM}}(\mathrm{VM}))$
18:             $\mathrm{fitness}[\mathrm{BestPM},\mathrm{VM}] \leftarrow f(rcv)$
19:         else
20:             $\mathrm{fitness}[\mathrm{BestPM},\mathrm{VM}] \leftarrow \infty$
21:         end if
22:     end for
23: end while
```

4.2 Simulation Experiments and Evaluation

To measure the performance of the proposed resource allocation metrics and methods, we developed two custom simulation environments.

The first one is implemented in Java and measures the effect of heterogeneity in cloud resources to allocation performance. The design is simple with few components. *PhysicalMachine* and *VirtualMachine* classes are subclasses of *AllocationResource* which is a model of vector operations. *Cloud* class consists of allocation algorithms and random generation of PM resources and VM requests. Many instances of *Cloud* are generated and allocation results are gathered by *Simulation* class. The code runs on any system with Java 5 or above.

The second simulation environment we developed aims to measure the performance of allocation methods we proposed and compare them with the ones from literature. It is based on open-source Python code provided by [60]. The codebase consists of four main files. *container.py* file models a problem *Instance* that consists of *Item*s and *Bin*s. *generator.py* file generates problem *Instance*s according to different set of specifications and parameters. *measures.py* file contains allocation metrics which are used by the allocation algorithms defined in *heuristics.py* file. Finally, *benchmark.py* file runs a simulation consisting of many different problems.

We extended the existing code by implementing our fitness functions and integration code with the rest of the simulation environment in *measures.py* file. We also implemented *trace.py* file that keeps a trace of the simulation so that results are persisted and could be analyzed by importing them into *pandas* [61]. We modularized *benchmark.py* file by introducing a *config.py* file that keeps a set of parameters and measures for which a simulation was run. Using these configuration files, we were able to implement *parallel.py* file that runs many simulations with different parameters in parallel, taking advantage of many-core architectures. The code runs on a Linux system with python 2.7 and *numpy* [62].

4.2.1 Effect of Resource Heterogeneity to Allocation Performance

Before evaluating our metrics and comparing them with others, we wanted to see the effect of heterogeneity. For this experiment we considered number of physical machine types as the metric of heterogeneity. When the cloud infrastructure consists of a single type of physical machine, it has a homogenous architecture. The more types of different physical machines make it more heterogeneous. Physical machine types are specified by the number of resource dimensions and minimum and maximum capacities for each dimension. The capacity of a resource in a dimension is drawn uniform randomly between the minimum and maximum capacity for that dimension. The number of physical machines available in the cloud for each type is also drawn uniform randomly from a specified interval. We examined the effect of number of machine types to request satisfaction ratio, i.e., the ratio of VMs that are requested and placed, to all VMs that are requested. In order to minimize the effect of randomness to simulation results, an average of 200 runs is presented.

Figure 4.1 shows that request satisfaction increases as heterogeneity increases. Having a more heterogeneous infrastructure enables virtual machines of different resource demand compositions to find suitable resources, therefore increasing the request satisfaction ratio. As can be seen, request satisfaction ratio does not increase monotonically as the number of types of PMs increases. Peaks and lows of request satisfaction ratio correlate with those of cloud capacity in dimension 1, which is the resource constrained capacity that dominates the satisfaction ratio for this scenario.

4.2.2 Performance of Proposed Methods

Next we present our main results: the results of performance and comparison experiments. In these experiments, we want to compare various metrics and algorithms in terms of how well VMs are placed. For this we consider the following

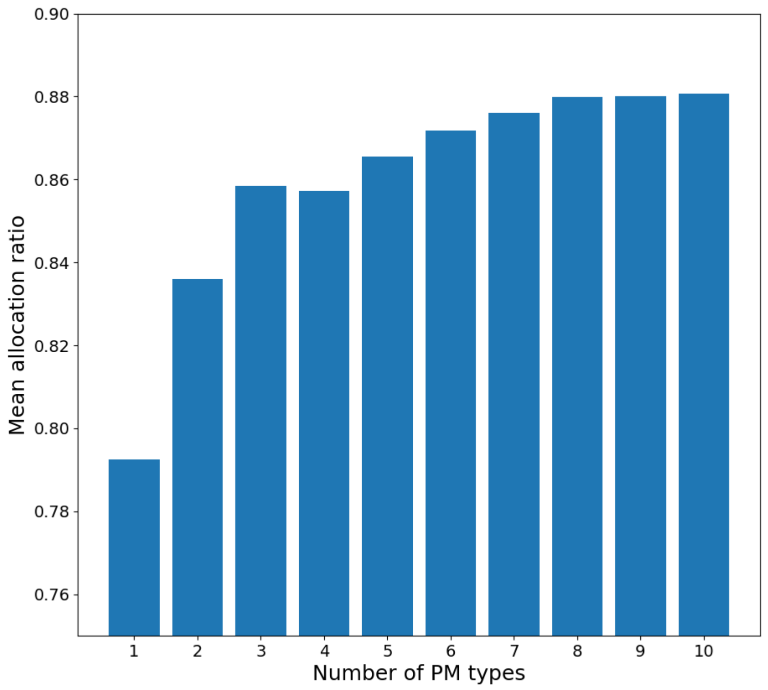


Figure 4.1: Effect of heterogeneity on allocation ratio.

evaluation strategy. We generate feasible problem instances. A problem instance has a set of VMs (items) with d dimensional resource requirements to be placed into a set of PMs (bins) with d dimensional resource capacities. In a feasible problem instance all VMs can be placed into PMs. Various resource dimensions (d) and number of bins (PMs) are considered. A problem instance can be for example: place 300 items with some random request requirements into 50 bins with some random capacities. We generate the items to make an instance feasible. If placed in the generation order, all items will be placed. But while placing the items, the algorithm does not know the feasible order (which is very hard to enumerate).

Consider a toy-example cloud of 2 PMs with 2 resource dimensions. PMs have capacities (RCVs) of $\binom{7}{7}$ and $\binom{5}{6}$. 3 VM requests are to be allocated with resource requirements (RRVs) of $\binom{4}{3}$, $\binom{2}{4}$, and $\binom{5}{5}$. VMs 1 and 2 could be allocated to PM 1, and VM 3 could be allocated to PM 2. Therefore, the problem is feasible.

We consider 5 different problem instance classes, as defined by [40]. These are: 1) random uniform, where bin capacities are chosen independently and in a uniform random manner (we call it as *uniform* in our result tables); 2) random uniform with rare resources, where bin capacities are again chosen randomly but one resource dimension is scarce (we call it as *uniform-rare*); 3) correlated capacities, where dimension capacities of a bin are correlated but resource requests are not (we call it as *correlated-false*); 4) correlated capacities and requirements, where both bin dimension capacities and resource dimension requirements are correlated (we call it as *correlated-true*); 5) similar items and bins, where resource requests are similar to bin capacities (we call it as *similar*).

Depending on the metric, not all VMs can be placed. If at least one VM in a problem instance cannot be placed, we consider that instance as unsolved (unsuccessful). If all VMs in a problem instance are placed, we consider that as solved (success). We count, for each algorithm, the number of instances solved. The more, the better. For example, if use of a metric A enables 4200 of 5000 instances to be solved, we consider it better than a metric B that enables only 3700 of the same instances to be solved.

Consider the toy-example cloud defined above. Using UCfit(2, 1, 0.2) we first calculate fitness values for all VM-PM pairs. Among 6 pairs VM 3 $\binom{5}{5}$ - PM 2 $\binom{5}{6}$ pair has the best fitness value of 0.013, therefore VM 3 is allocated to PM 2. PM 2 now has residual capacity (RCV) $\binom{0}{1}$, which is less than RRVs of VM 1 or 2. Next, VM 1 $\binom{4}{3}$ is placed into PM 1 $\binom{7}{7}$ because its fitness value, 0.12, is better than that of VM 2 $\binom{2}{4}$, 0.25. Finally, VM 2 is placed into remaining capacity of PM 1 $\binom{3}{4}$. As a result, UCfit(2, 1, 0.2) is able to find a feasible solution to this problem instance.

Using dot product (dp) metric, for example, results in a different allocation. dp metric calculates dot products of RCV and RRV, and considers higher values to fit better. Among all VM-PM pairs, VM 3 $\binom{5}{5}$ and PM 1 $\binom{7}{7}$ has the best fitness value of 70. Once VM 3 is placed into PM 1, RCV of PM 1 becomes $\binom{2}{2}$ which is not large enough to hold any of the remaining VMs. Next, VM 1 $\binom{4}{3}$ is allocated to PM 2 $\binom{5}{5}$, because its dot product, 35, is better than that of VM 2 $\binom{2}{4}$, 30. Residual capacity (RCV) of PM 2 is now $\binom{1}{3}$, therefore VM 2 is left unallocated. dp metric is considered to be unsuccessful because it fails to find a feasible allocation.

We compare our metrics, TRfit and UCfit, with some other metrics that are defined in other studies and by [40]. Table 4.1 shows the first set of results. The number of resource dimensions d is varied between 2 and 8. There are 30 physical machines (bin count is 30). There are 5 problem classes. For each class 100 instances are generated. Hence, for each resource dimension count d there are a total of 500 instances. Therefore the maximum success count can be 500 for a given resource dimension count, meaning that all 500 problem instances are solved. Total number of problems is 3500.

As can be seen in Table 4.1, our methods TRfit and UCfit perform much better than various other methods proposed in literature and by [40]. While our two methods with various parameters achieve a total success count of at least 1485, 42.4% of the problems for all resource dimensions, a lot of other methods are providing a total success count below 1000 (28.5%). Our best result is obtained by UCfit with 1711 problems (48.8%) solved successfully, whereas the closest performing other method, dp, provides 1433 feasible solutions (40.9%). Our metric improves over performance of dp by 19.3%. There are metrics with very poor performance (less than 2.0%), which proves that choosing a good heuristic is important.

In the subsequent experiments we compared our parametric methods TRfit and UCfit with the best performing three methods from [40], dp, dp_normC, and dp_normR, and three other methods from literature, r, riv, and sandpiper. 6 different bin counts are considered, 10, 20, 30, 40, 50, and 100, each having 9 different resource dimensions: $2-10$. For each bin count and dimension count pair, we generate 100 problem instances for each instance class. Therefore, a total of 5400 instances are generated for each class, and the total number of instances is 27,000 across all classes.

Table 4.2 shows the performance of the selected best performing methods and our methods with different parameters for various problem classes. When the bin capacities at various dimensions are correlated, heuristics perform the best

Table 4.1: Number of problem instances solved (i.e., success count) by various methods for various number of resource dimensions (d). Each column gives the results for a different d value. There are 30 physical machines. Success count can be at most 500.

# bins	30	30	30	30	30	30	30	
# resources	2	3	4	5	6	7	8	
Method								Total
UCfit-2-1-02	480	384	289	193	123	121	121	1711
UCfit-2-1-025	475	392	287	191	129	111	117	1702
UCfit-2-15-02	465	368	289	191	130	119	123	1685
UCfit-3-2-01	469	378	282	197	125	119	115	1685
UCfit-2-1-03	477	386	289	194	118	110	105	1679
UCfit-3-1-01	476	386	293	183	119	111	111	1679
UCfit-2-1-01	463	374	279	189	124	120	128	1677
TRfit-pi-3	453	414	231	181	122	103	100	1604
TRfit-pi-2	470	399	228	177	119	100	100	1593
TRfit-pi-6	436	385	252	179	125	105	100	1582
TRfit-pi-12	410	348	256	160	109	102	100	1485
dp	450	366	186	129	103	99	100	1433
dp_normC	425	312	196	116	100	100	100	1349
bc_dyn_1/C	417	281	142	102	100	100	100	1242
bc_dyn_R/C	414	272	150	103	100	100	100	1239
R/C	410	270	145	104	100	100	100	1229
bc_dyn_1/R	403	269	132	100	100	100	100	1204
1/C	385	273	142	103	100	100	100	1203
1/R	378	263	141	103	100	100	100	1185
ic_dyn_R/C	350	254	152	104	100	100	100	1160
ic_dyn_1/C	344	245	144	103	100	100	100	1136
ic_dyn_1/R	323	232	141	105	100	100	100	1101
sbb_st_R/C	368	190	108	100	100	100	100	1066
sbb_st_1/R	363	193	108	100	100	100	100	1064
sbb_dyn_R/C	370	181	107	100	100	100	100	1058
sbb_st_1/C	366	184	107	100	100	100	100	1057
sbb_dyn_1/C	331	186	104	100	100	100	100	1021
sbb_dyn_1/R	318	186	108	100	100	100	100	1012
bb_dyn_1/R	221	128	101	100	100	100	100	850
bb_dyn_1/C	215	130	102	100	100	100	100	847
bb_st_1/R	214	125	100	100	100	100	100	839
bb_st_R/C	212	122	100	100	100	100	100	834
bb_st_1/C	207	125	101	100	100	100	100	833
bb_dyn_R/C	197	128	101	100	100	100	100	826
sandpiper	101	100	100	100	100	100	100	701
shuff1	141	81	85	71	65	68	68	579
nothing	152	78	73	75	64	67	66	575
dp_normR	173	100	36	15	30	46	56	456
sbb_nothing	63	40	33	31	16	20	17	220
sbb_shuff1	58	35	28	29	24	13	22	209
riv	92	34	6	1	0	1	6	140
bc_shuff	26	14	8	5	3	3	4	63
sbb_shuff	12	2	6	4	2	1	1	28
ic_shuff	8	1	0	0	0	0	0	9
bb_shuff1	4	1	0	1	0	0	1	7
bb_nothing	1	0	0	0	1	0	0	2
bb_shuff	0	0	0	0	0	0	2	2

Table 4.2: Success count of various methods for various instance classes. Each column is for a different instance class. Success count of a method for a class can be at most 5400: 9 different d values and 6 different bin count values are used. That means a total of 54 different configurations considered. For each configuration, there are 100 instances.

	Instance Class					
Method	cor-False	cor-True	similar	unif	unif-rare	Total
UCfit-2-1-02	2175	5395	3030	1240	700	12,540
UCfit-2-1-03	2198	5394	2790	1307	718	12,407
UCfit-2-1-01	2115	5394	3095	1158	629	12,391
UCfit-3-1	2088	5387	3021	1190	609	12,295
UCfit-2-2-02	2142	5394	3009	1091	610	12,246
UCfit-3-1-02	2192	5395	2219	1329	758	11,893
TRfit-pi-4	2595	5396	1443	1426	603	11,463
UCfit-2-3-02	1846	5395	2444	1081	621	11,387
UCfit-2-1	1802	5292	2885	884	443	11,306
TRfit-pi-2	2480	5393	1072	1477	803	11,225
TRfit-3pi-4	2365	5391	869	1394	842	10,861
UCfit-1-1-02	1330	5370	2691	768	460	10,619
dp	1695	5372	1142	1441	595	10,245
dp_normC	1919	5388	817	1325	319	9768
r	2003	5387	405	1035	667	9497
TRfit-0	593	5390	2341	647	111	9082
sandpiper	47	5369	5	3	7	5431
dp_normR	402	219	3027	122	177	3947
UCfit-1-1	100	387	2161	119	175	2942
riv	0	1	1573	0	38	1612

compared to other problem classes. When overall success count is considered, our UCfit metric performs the best. In total, it finds a feasible solution to 12,540 problem instances (46.4% of all problems). Our TRfit metric performs close to UCfit, where the best TRfit value is 11,463 (42.4%). Performance of other metrics from the literature is worse. The closest one, dp, solves 10,240 instances (37.9%), Compared to dp, UCfit performs 22.4% better. Riv performs the worst among the compared algorithms, satisfying just 1612 instances (5.9%), which is only 12.8% of our best result.

Table 4.3 and Table 4.4 compare various methods for different number of resource dimensions (d). In Table 4.3 we see results for correlated capacities only (correlated-False class). The first thing that we notice is that when d increases, the placement becomes more difficult and the number of satisfied problem instances decreases dramatically for all metrics. For $d > 8$, problems become practically unsolvable with the proposed heuristics. TRfit achieves highest scores compared to other metrics, solving 2595 of 6000 problems (43.2%). Closest performing method from literature, r, is able to solve 2003 problems (33.3%). TRfit achieves 29.5% better performance than r. The difference between two methods are lower for easier problems. Both methods achieve above 95% performance for $d < 4$. The difference becomes larger for harder problems with $d > 4$, where TRfit solves 24.8% of the problems while r solves just 9.8%.

In Table 4.4 we see results for instances with uniform random capacities (uniform class). First of all, compared to correlated capacities, success rate of the methods are in general lower for uniform instance class. It is easier to place VMs into instances with correlated capacities, than with uniform random capacities. It is also interesting to note that when d gets larger, sometimes the success count can increase as well. For example, for various versions of UCfit heuristic, success count is more when d is 10 compared to when d is 5. This shows that depending on the capacities, having more dimensional resources does not always mean less success rate. The performance difference between TRfit and closest performing method from literature, dp, is significantly lower compared to correlated capacities class. TRfit solves 1477 of 6000 problems (24.6%) and dp solves 1441 (24.0%).

Table 4.5 through Table 4.9 evaluate the performance of various methods for different classes and bin counts. Instances with correlated capacities and requirements (correlated-True class) enable the best success rate for the methods. Because both capacities and requirements are correlated, it is easier to find good placements for VMs compared to other problem classes where it is harder to distinguish between good and bad placements. Uniform random capacities with rare resources (uniform-rare class), on the other hand, is the most difficult to solve instance class.

Table 4.5 gives the success rate of the methods for various bin counts for the

Table 4.3: Success counts of methods for various number of resource types (d) for correlated capacities (correlated-false) instance class. d is varied between 2 and 10. Each column shows the results for a different d value. There are 6 different bin counts considered (10, 20, 30, 40, 50, 100). The success count can be at most 600.

	# resources (d)									
Method	2	3	4	5	6	7	8	9	10	Total
TRfit-pi-4	599	576	524	427	279	134	55	1	0	2595
TRfit-pi-2	599	589	536	415	232	102	6	1	0	2480
TRfit-3pi-4	600	590	530	392	184	68	1	0	0	2365
UCfit-2-1-03	594	567	502	335	151	48	0	0	1	2198
UCfit-3-1-02	595	581	504	329	145	38	0	0	0	2192
UCfit-2-1-02	593	559	490	335	152	46	0	0	0	2175
UCfit-2-2-02	587	545	474	322	151	61	2	0	0	2142
UCfit-2-1-01	585	549	477	320	136	48	0	0	0	2115
UCfit-3-1	580	553	481	306	138	28	1	0	1	2088
r	599	572	476	266	85	5	0	0	0	2003
dp_normC	595	562	469	217	75	1	0	0	0	1919
UCfit-2-3-02	590	521	394	231	91	18	1	0	0	1846
UCfit-2-1	554	486	398	244	102	18	0	0	0	1802
dp	590	517	367	169	48	3	1	0	0	1695
UCfit-1-1-02	556	403	244	101	23	3	0	0	0	1330
TRfit-0	315	176	67	30	5	0	0	0	0	593
dp_normR	271	113	17	1	0	0	0	0	0	402
UCfit-1-1	94	5	1	0	0	0	0	0	0	100
sandpiper	39	7	1	0	0	0	0	0	0	47
riv	0	0	0	0	0	0	0	0	0	0

Table 4.4: Success counts of methods for different d values and for uniform instance class. Success count can be at most 600.

	# resources (d)									
Method	2	3	4	5	6	7	8	9	10	Total
TRfit-pi-2	598	519	255	52	1	2	12	13	25	1477
dp	590	508	231	42	1	7	11	20	31	1441
TRfit-pi-4	592	491	237	52	3	5	10	11	25	1426
TRfit-3pi-4	597	508	210	22	1	4	13	13	26	1394
UCfit-3-1-02	589	482	197	6	1	2	12	14	26	1329
dp_normC	586	468	201	16	2	1	5	15	31	1325
UCfit-2-1-03	582	456	197	20	1	2	11	12	26	1307
UCfit-2-1-02	575	423	174	18	1	1	11	12	25	1240
UCfit-3-1	553	423	152	9	1	2	11	13	26	1190
UCfit-2-1-01	564	397	142	4	1	0	10	14	26	1158
UCfit-2-2-02	570	372	101	2	1	1	8	13	23	1091
UCfit-2-3-02	575	386	86	1	2	2	5	9	15	1081
r	580	375	39	1	0	2	10	10	18	1035
UCfit-2-1	488	289	57	0	1	1	9	13	26	884
UCfit-1-1-02	521	200	13	0	1	2	6	6	19	768
TRfit-0	478	138	8	0	1	0	4	3	15	647
dp_normR	103	7	0	0	0	0	1	0	11	122
UCfit-1-1	84	18	0	0	1	0	4	3	9	119
sandpiper	3	0	0	0	0	0	0	0	0	3
riv	0	0	0	0	0	0	0	0	0	0

Table 4.5: Success counts of methods for different *bin count* values and for *correlated-false* instance class (that means only dimension capacities are correlated not the dimension requirements). Success count can be at most 900.

	# bins						
Method	10	20	30	40	50	100	Total
TRfit-pi-4	236	333	410	463	504	649	2595
TRfit-pi-2	254	328	394	428	481	595	2480
TRfit-3pi-4	254	319	372	401	452	567	2365
UCfit-2-1-03	224	285	352	388	417	532	2198
UCfit-3-1-02	230	294	344	386	412	526	2192
UCfit-2-1-02	223	286	340	382	416	528	2175
UCfit-2-2-02	191	269	341	385	417	539	2142
UCfit-2-1-01	212	275	332	369	405	522	2115
UCfit-3-1	206	267	331	368	406	510	2088
r	222	271	321	349	371	469	2003
dp_normC	206	257	297	336	360	463	1919
UCfit-2-3-02	195	239	295	321	362	434	1846
UCfit-2-1	141	212	284	323	368	474	1802
dp	219	255	275	302	310	334	1695
UCfit-1-1-02	110	157	197	237	257	372	1330
TRfit-0	101	108	107	102	99	76	593
dp_normR	44	38	46	57	71	146	402
UCfit-1-1	25	15	10	17	13	20	100
sandpiper	38	8	1	0	0	0	47
riv	0	0	0	0	0	0	0

class of instances where only resource capacities but not requirements are correlated (correlated-False class). Note that the problem gets easier as the number of bins increases. The best performing method is TRfit with $\alpha = \pi/4$. It solves 2595 of 5400 problems (48.0%). Other TRfit metrics with different parameters are at the top as well, except TRfit(0). Then comes UCfit. The metrics r and dp rank around the middle, by solving 37.0% and 31.3% of the problems respectively. TRfit-pi-4 improves over the performance of r by 29.5%. The metrics sandpiper and riv are at the bottom. sandpiper is able to solve less than 1% of the problems, where riv fails to solve any.

Table 4.6 gives the success rate of the methods for various bin counts for class of instances with correlated capacities and requirements (correlated-True). This

Table 4.6: Success counts of methods for different *bin count* values and for *correlated-true* instance class (both dimension capacities and requirements are correlated). Success count can be at most 900.

	# bins						
Method	10	20	30	40	50	100	Total
TRfit-pi-4	896	900	900	900	900	900	5396
UCfit-2-1-02	895	900	900	900	900	900	5395
UCfit-2-3-02	895	900	900	900	900	900	5395
UCfit-3-1-02	895	900	900	900	900	900	5395
UCfit-2-1-01	894	900	900	900	900	900	5394
UCfit-2-1-03	894	900	900	900	900	900	5394
UCfit-2-2-02	894	900	900	900	900	900	5394
TRfit-pi-2	893	900	900	900	900	900	5393
TRfit-3pi-4	892	899	900	900	900	900	5391
TRfit-0	890	900	900	900	900	900	5390
dp_normC	889	899	900	900	900	900	5388
r	888	899	900	900	900	900	5387
UCfit-3-1	889	898	900	900	900	900	5387
dp	873	900	899	900	900	900	5372
UCfit-1-1-02	885	894	899	896	897	899	5370
sandpiper	875	895	899	900	900	900	5369
UCfit-2-1	842	876	887	894	894	899	5292
UCfit-1-1	125	72	59	53	48	30	387
dp_normR	97	54	31	23	10	4	219
riv	1	0	0	0	0	0	1

is the easiest class of all instance classes. Almost all methods perform above 98.0%.

Table 4.7 gives the success count of the methods for various bin counts for the class of instances that have similar capacities (similar). The problems become harder to solve as the number of bins increases. This time UCfit method performs the best, solving 3095 of 5400 problems (57.3%). The metric dp_normR is very close with 56.0% performance. This is the only class of problems for which dp_normR shows reasonable performance. It performs poorly in all other cases. riv has the same performance characteristic with dp_normR, though it has worse performance, 29.1%. The metric r performs poorly, 7.5%, and sandpiper is the worst.

Table 4.7: Success counts of methods for different *bin count* values and for *similar* instance class. Success count can be at most 900.

	# bins						
Method	10	20	30	40	50	100	Total
UCfit-2-1-01	733	571	466	425	431	469	3095
UCfit-2-1-02	708	553	463	414	414	478	3030
dp_normR	779	614	518	437	378	301	3027
UCfit-3-1	714	553	459	403	413	479	3021
UCfit-2-2-02	729	548	462	425	401	444	3009
UCfit-2-1	703	539	438	398	377	430	2885
UCfit-2-1-03	671	496	399	386	391	447	2790
UCfit-1-1-02	681	502	397	365	353	393	2691
UCfit-2-3-02	604	417	350	338	339	396	2444
TRfit-0	571	385	332	333	334	386	2341
UCfit-3-1-02	553	353	307	301	321	384	2219
UCfit-1-1	608	383	297	281	272	320	2161
riv	548	291	192	179	170	193	1573
TRfit-pi-4	239	202	218	236	250	298	1443
dp	160	157	186	200	202	237	1142
TRfit-pi-2	129	149	182	190	195	227	1072
TRfit-3pi-4	93	116	140	150	169	201	869
dp_normC	105	113	123	135	146	195	817
r	35	50	65	74	82	99	405
sandpiper	5	0	0	0	0	0	5

Table 4.8: Success counts of methods for different *bin count* values and for *uniform* instance class. Success count can be at most 900.

	# bins						
Method	10	20	30	40	50	100	Total
TRfit-pi-2	201	188	213	247	280	348	1477
dp	210	184	210	236	263	338	1441
TRfit-pi-4	194	176	208	241	263	344	1426
TRfit-3pi-4	205	176	197	235	261	320	1394
UCfit-3-1-02	182	172	198	229	243	305	1329
dp_normC	176	161	199	227	248	314	1325
UCfit-2-1-03	172	156	192	217	250	320	1307
UCfit-2-1-02	154	140	181	215	233	317	1240
UCfit-3-1	146	138	171	206	225	304	1190
UCfit-2-1-01	147	132	169	198	216	296	1158
UCfit-2-2-02	148	123	159	181	207	273	1091
UCfit-2-3-02	142	132	156	184	201	266	1081
r	145	128	158	174	200	230	1035
UCfit-2-1	110	85	117	156	172	244	884
UCfit-1-1-02	93	80	107	132	153	203	768
TRfit-0	81	80	94	111	124	157	647
dp_normR	32	14	11	11	7	47	122
UCfit-1-1	35	14	10	11	15	34	119
sandpiper	3	0	0	0	0	0	3
riv	0	0	0	0	0	0	0

Table 4.8 gives the success count of the methods for various bin counts when bin capacities are drawn randomly from a uniform distribution (uniform). The problems become easier to solve as the number of bins increases. TRfit-pi-2 and dp are performing the best, with 27.3% and 26.6% performance respectively. In general, TRfit is ahead of UCfit for this class of instances. The best performing TRfit improves over the performance of the best performing UCfit by 11.1%. sandpiper and riv have the worst performance.

Table 4.9 gives the success count of the methods for various bin counts when bins are generated in the same manner as in uniform class, but the capacity of the last dimension is set to zero with certain probability indicating, a rare resource in the cloud (uniform-rare). Note that this is the most difficult of all problem classes. The problems become slightly harder to solve for intermediate number of

Table 4.9: Success counts of methods for different *bin count* values and for *uniform-rare* instance class. Success count can be at most 900.

	# bins						
Method	10	20	30	40	50	100	Total
TRfit-3pi-4	228	115	114	109	116	160	842
TRfit-pi-2	229	114	104	95	101	160	803
UCfit-3-1-02	220	103	98	98	100	139	758
UCfit-2-1-03	205	94	86	93	103	137	718
UCfit-2-1-02	205	90	93	91	93	128	700
r	183	89	92	91	94	118	667
UCfit-2-1-01	187	82	82	81	87	110	629
UCfit-2-3-02	159	80	83	86	93	120	621
UCfit-2-2-02	169	82	78	81	84	116	610
UCfit-3-1	182	75	72	76	82	122	609
TRfit-pi-4	219	91	73	58	65	97	603
dp	196	79	63	65	75	117	595
UCfit-1-1-02	136	65	51	58	60	90	460
UCfit-2-1	145	53	49	51	60	85	443
dp_normC	116	31	30	31	37	74	319
dp_normR	93	31	21	9	13	10	177
UCfit-1-1	79	31	18	18	11	18	175
TRfit-0	82	15	10	2	1	1	111
riv	31	6	0	0	1	0	38
sandpiper	7	0	0	0	0	0	7

bins. They are easier for lower and higher number of bins. The best performing TRfit solves 842 of 5400 problems (15.5%). The best performing UCfit is slightly behind with 758 problems (14.0%). Performance of metric r is relatively good (12.3%), but TRfit is able to achieve 26.2% better performance. Metrics riv and sandpiper again perform the worst with less than 1% performance.

4.3 Summary

In this chapter, we show how the metrics defined in Chapter 3.2.3 can be used as part of some sample VM placement algorithms. In this way, we provide complete and generic VM placement methods that can be applied for various resource types and counts. A metric is always monitored and used while virtual machines are being placed into physical machines to have a utilization state that is as suitable as possible. Hence the metrics are used in selecting VMs and PMs to make a good assignment. The methods are heuristic-based, therefore they run very fast and in polynomial time with respect to request count and machine count.

We compared our methods both internally, for different parameter settings, and also externally with some other methods from literature in terms of number of VM placement problems that can be solved completely — without having any unplaced VMs. We investigated the effect of physical machine count, resource type count, and also problem instance classes. For all cases, our methods perform better than existing methods, by 22.4% overall, and by up to 29.1% when individual problem classes are considered. Relative ranks of our methods against each other varies depending on the case.

Chapter 5

Allocation of Wireless Network Resources in Data Centers

Next, we investigate the problem of allocating network resources in data centers. We focus on the problem of offloading as much traffic as possible from wired to wireless network, so that hybrid wireless data centers could be designed accordingly, to achieve better construction and operating efficiency. Given traffic demands between top-of-the-rack (ToR) switches, our proposed methods establish multi-hop routes to achieve this goal. We begin by presenting a system model under which our methods work.

5.1 System Model

The properties of a wireless network in our hybrid data center system model can be defined by the following parameters. We provide a detailed discussion of how these parameters are chosen in the following section.

1. Wireless communication technology: Data center networks carry a lot of traffic; therefore, bandwidth is one of the most important parameters of

wireless communication technology. Like many other studies, we assumed use of 802.11ad in our model, which works in 60 GHz ISM band.

2. Number of radios per node: The number of radios per node determines the maximum number of wireless connections a node can establish with its neighbors. We assume the availability of two radios per node, i.e., per ToR switch.

3. Number of channels: Dividing the frequency band into channels enables multiple communications to work simultaneously. The number of radios also affects the number of required channels. In our work, we show that using three non-overlapping channels of 802.11ad is a good choice when there are two radios per node.

4. Communication distance: The physical distance between nodes affects wireless communication quality significantly. A good assignment of wireless links should consider the effect of the distance between nodes. We safely assume that wireless communication could be performed with necessary quality up to 5 meters.

5. Unit of traffic: The traffic flow requirement between two nodes in configuration period (i.e., until the next execution of the allocation algorithm to reconfigure the wireless radios and links) can be expressed as the total amount of traffic to be carried, or as the required bandwidth of the flow. In this thesis, we consider traffic requirement as the total amount of traffic to be communicated between nodes in one configuration.

5.1.1 Rationale

When wireless links are assigned in a centralized way, there is no need to run a full-scale medium access control (MAC) protocol between nodes. Nodes would know with which other nodes they would communicate with, using which radio and which channel. Therefore, the cost of using a MAC protocol could be reduced.

Figure 5.1 shows a network in which nodes A and C communicate with node B. If nodes have multiple radios, A and C can simultaneously communicate with B provided that they use different channels. Otherwise, transmissions will interfere at node B. We consider two counter-arguments to this statement.

By using directional antennas the effect of interference at node B could be reduced. The amount of interference depends on the angular difference between two wireless links. This requires that the spatial location of the nodes to be considered when assigning wireless paths. We evaluate that such a consideration would bring unnecessary complexity compared to using different channels for such cases.

A MAC protocol is also not enough to solve the problem of interference at node B. It is very likely that nodes A and C cannot hear each other, when the physical arrangement of data centers and the short distance communication properties of 802.11ad technology is considered. Therefore, we prefer to use a different channel rather than using a MAC protocol in such cases.

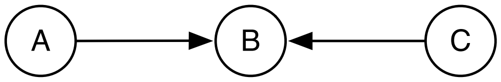


Figure 5.1: An example of interference at node B with a single channel.

Figure 5.2 shows an example of interference at node B that could be addressed by using a MAC protocol. This configuration could be for a traffic pattern in which A is sending traffic to C over B, or A is sending traffic to B while B is sending another traffic to C independent of the traffic A is sending to B. Even when directional antennas are used and when the angular difference between nodes are at optimum value, it is not possible to conduct both communications simultaneously over the same channel. The transmission power from B to C would be too high compared to the power of the signal heard at the other radio of B. If a single channel is to be used, one of the traffic flows should be canceled if we have two flows. If there is a single traffic flowing from A to C, it should be canceled.

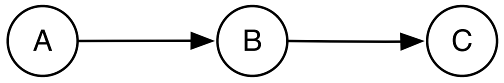


Figure 5.2: Another example of interference at node B with a single channel.

A third pattern of communication is depicted in Figure 5.3. Node B is transmitting to both A and C simultaneously. Regardless of the spatial location of the nodes and the MAC protocol, B could use the same channel for both communications. We assume that nodes correspond to top-of-the-rack (ToR) switches and the physical distance between racks allow both traffic to be transmitted without interference [20].

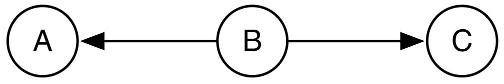


Figure 5.3: An example of no interference at node B with a single channel.

Figure 5.4 shows the channel assignment structure in a larger context. Wireless links between nodes could carry multi- or single-hop traffic. Each node has two radios, one for incoming and one for outgoing traffic. Two channels are necessary to transmit all traffic simultaneously. Channels could be assigned in an alternating fashion as shown in the figure.

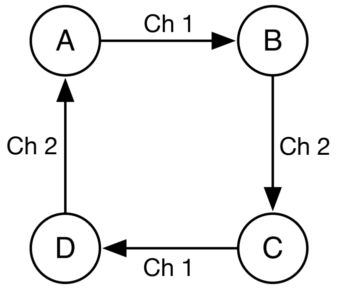


Figure 5.4: An example of a traffic pattern that can be carried using two channels.

Figure 5.5 shows a slightly modified version. Even though each node has two radios, one for incoming and one for outgoing traffic as in the previous case, this time two channels are not enough to transmit all traffic simultaneously. When channels are assigned in an alternating fashion, it is clear that at least three channels are needed.

As demonstrated in Figure 5.3, the direction of flows may reduce interference,

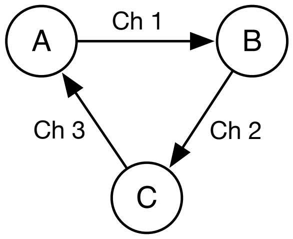


Figure 5.5: An example of a traffic pattern that cannot be carried using two channels.

therefore, the required number of channels. However, in the worst-case each link established at a node may interfere with each other. Therefore, we use an undirected graph to represent wireless connections. Channels assigned to wireless links could be thought of as colors assigned to edges of the graph (edge-coloring problem). According to [63], the edge chromatic number, $\chi'(G)$, of an undirected graph is determined by the formula in Equation 5.1:

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1 \tag{5.1}$$

where $\Delta(G)$ is the degree of the maximum degree vertex in the graph. In our system, each node has N radios, therefore the maximum number of connections a node could establish with other nodes is N. We conclude that $\Delta(G) = N$.

Given any graph, determining if $\chi'(G) = \Delta(G)$ or $\chi'(G) = \Delta(G) + 1$ is NP-complete [64]. Therefore, there is no guarantee that an assignment of wireless links could be allocated N channels. Finding an assignment that could be satisfied with N channels is not practical either.

As a result, we conclude that $N + 1$ channels is an appropriate choice. The number of channels in 802.11ad varies between 3 and 6 by region. In order to have the widest applicability of our system model, we decide to use 2-radio nodes with each radio using one of 3 wireless channels.

5.1.2 Effects of Wireless Communication on Transport Layer Protocols

TCP was designed for wired networks, which have a different packet loss profile than wireless networks. Congestion is the most important reason for packet loss in a wired network, whereas in a wireless network packet loss occurs because of bit errors introduced by factors such as signal interference, obstruction, and mobility. TCP performs worse when the end-to-end path contains a wireless link. The performance of TCP over a multi-hop wireless network is especially affected, because packet loss in *any one of the wireless links* is enough to disrupt end-to-end throughput maintained by TCP.

In our system model, the effect of aforementioned factors is minimized. First, signal interference is minimized by assigning different non-overlapping channels to radios of the same node. Same-channel transmissions that occur between other nodes could still cause interference at a node, but short-distance and line-of-sight communication restrictions of 60 GHz band reduce the impact. Second, radios are placed on top of the rack cabinets, therefore preventing the obstruction of line-of-sight communication by the service personnel. There are different sizes of rack cabinets, but the typical 42U full-height ones used in data centers measure 186-cm high. Radios could be easily elevated higher to reduce the chance of physical obstruction.

In the absence of packet loss, theoretical TCP performance over wireless links is the same as the performance achieved over wired connections. In order to tolerate some packet loss introduced by unforeseen circumstances, TCP with BBR [65] congestion control implementation could be preferred [66].

UDP is less affected by the type of medium in terms of magnitude of packet losses. Considering that we expect TCP to perform well under our system model, UDP should work as well as it does on wired networks.

5.2 Proposed Method

Figure 5.6 shows, in dashed lines, possible wireless links that could be established between ToR switches. In practice, ToR switches could establish wireless connections to more distant ones as well, but to simplify the example it is assumed that only immediate neighbors could communicate wirelessly.

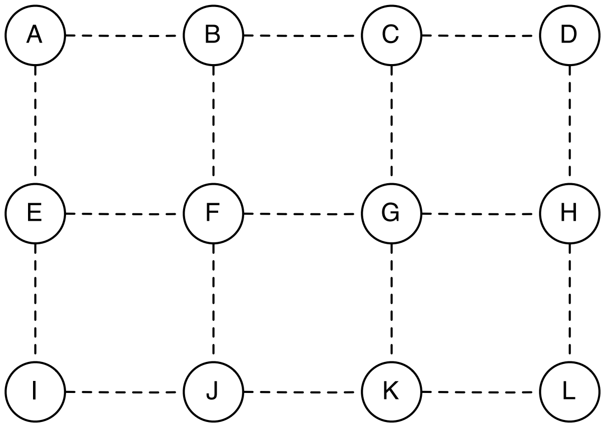


Figure 5.6: Possible wireless connections between ToR switches.

Figure 5.7 displays the amounts of traffic that needs to flow between nodes. 1 unit of traffic flows from Rack B to C, while Rack H sends 3 units of traffic to C. A single wireless link is needed to carry the traffic from Rack B to C. To carry the traffic from H to C, a multi-hop path of at least 2 hops should be allocated.

Routing traffic over the shortest path has several benefits. First of all, latency would be minimal. Secondly, resources would be used more efficiently by not allocating links more than necessary. Because the number of transmissions is minimal, it would cause the minimum amount of potential interference with other transmissions. Finally, it would conflict with fewer number of other flows. Nevertheless, it might be possible to have less conflict by allocating more hops to some of the flows. We compare both methods in our evaluations.

Assume that all single-hop traffic is already allocated. Considering that every node has 2 radios, when the traffic from H to C is routed over G, the traffic from G to K should be canceled, because one of the radios of G will be allocated

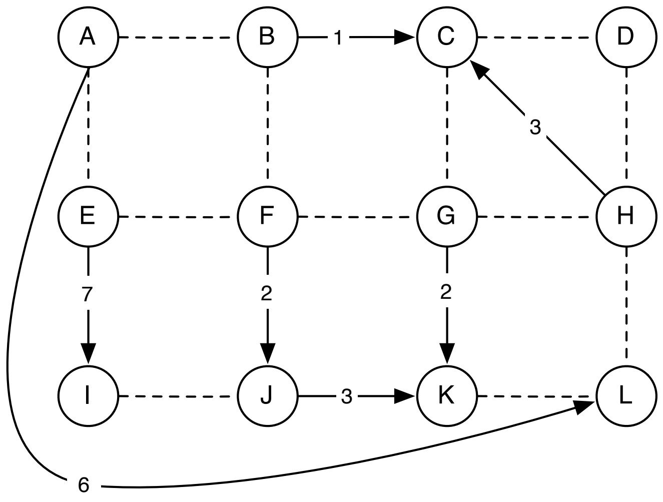


Figure 5.7: Amount of traffic flowing between nodes.

to receive traffic from H, and the other will be allocated to transmit to C. The amount of traffic flowing from H to C, 3, is higher than the amount of traffic from G to K, 2. Therefore, all other things being equal, allocating larger traffic instead of the smaller one makes wireless network carry more traffic without requiring reconfiguration of wireless radios and links, which involves steering the radio antennas to right directions (neighbors), setting channels, and establishing links. On the other hand, the traffic from H to C could be routed over D. Node D is not going to carry any other traffic, so the cost of routing the traffic over D is less than the cost of routing it over G.

The lowest conflict path for the traffic from A to L is over nodes B, F, J, K respectively. In that case, flows from B to C, and G to K will be deallocated, costing 3 units of traffic in total. Though, the benefit of allocating the traffic from A to L is 6 units, which is greater than its cost.

Figure 5.8 presents a multi-hop allocation of wireless links to traffic flows that maximizes the total amount of traffic carried over the network. If only single-hop communication were used, the total amount of traffic would be 15 units. Using multi-hop communication the total traffic amount is increased to 21 units.

After such an allocation is done, the wireless network is configured to carry the

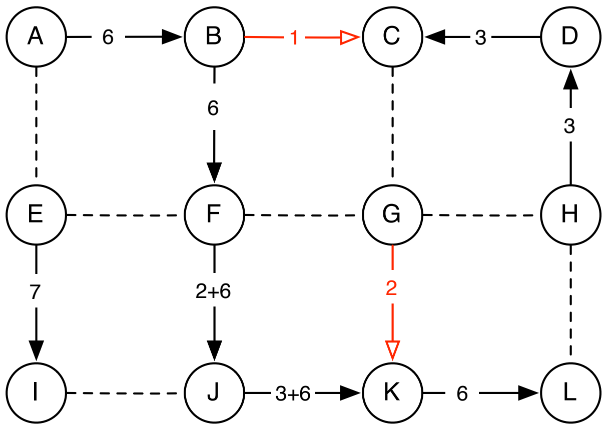


Figure 5.8: Allocation of wireless links to flows maximizing total traffic flow.

flows. For each radio in each node should steer its antenna to the right direction (to the relevant neighbor) to send or receive data traffic so that the links decided by the allocation algorithm to carry assigned flows can be established among the nodes.

A greedy algorithm that assigns multi-hop wireless links to traffic flows is given in Algorithm 6. The algorithm accepts as input a graph (G), a set of traffic flows (F), and the amount of traffic flows (τ). It returns a set of paths corresponding to assigned flows. G is an undirected graph of all possible wireless links between racks, similar to the one depicted in Figure 5.6. F is a set of source-target pairs. τ is a mapping from source-target pairs to their traffic amount (cost).

At the beginning of the algorithm, the set of wireless link allocations, *alloc*, for each flow is initialized. A wireless link is allocated to a flow in Line 20. Cost calculation of a potential allocation uses current set of allocations in Line 13.

Flows are assigned a shortest path from source to target. Lines $4-9$ calculate the distance from source to target, δ, using a breadth-first search on G. The for loop in Line 10 iterates over flows in increasing hop count and decreasing traffic. Beginning from a single-hop flow with highest traffic demand, first the other single-hop flows with lower traffic demands are assigned wireless connections, and then flows with higher number of flows are considered.

Algorithm 6 Allocate fly-paths to flows over shortest paths

```
FLY-PATH-SP($G, F, \tau$)                ▷ Wireless graph, set of flows, traffic demands
 1: for all $(u, v) \in E[G]$ do
 2:     $alloc(u, v) \leftarrow alloc(v, u) \leftarrow \emptyset$        ▷ An allocation is a set of flows
 3: end for
 4: for all $s \in \{s|(s, t) \in F\}$ do    ▷ Calculate length of flows as BFS distances
 5:     BFS($G, s$)
 6:     for all $t \in \{t|(s, t) \in F\}$ do
 7:         $\delta(s, t) \leftarrow d[t]$                    ▷ $d[t]$ is BFS distance from $s$ to $t$
 8:     end for
 9: end for
10: for all $(s, t) \in \text{SORT}_{<(\delta,-c)}(F)$ do
11:     $G_{\text{BFS}} \leftarrow \text{BFS}(G, s)$
12:     for all $(u, v) \in E[G_{\text{BFS}}]$ do         ▷ Calculate costs of edges in BFS DAG
13:         $\text{cost}(u, v) \leftarrow \text{POTENTIAL-COST}(s, t, u, v, alloc)$
14:     end for
15:     DIJKSTRA($G_{\text{BFS}}, cost, s$)                ▷ Find path costs to target, $d[t]$
16:     if $d[t] < \tau(s, t)$ then                  ▷ Path is feasible: cost < demand
17:         $u \leftarrow t$
18:         $\text{path}(s, t) \leftarrow nil$                          ▷ List of allocated edges
19:         do
20:             $alloc(\pi[u], u) \leftarrow alloc(\pi[u], u) \cup \{(s, t)\}$
21:             INSERT-HEAD($\text{path}(s, t), (\pi[u], u)$)
22:             DEALLOC($s, t, \pi[u], u, alloc, path$)      ▷ Deallocate conflicting flows
23:             $u \leftarrow \pi[u]$
24:         while $u \neq s$
25:     end if
26: end for
27: return path
```

Paths of flows are chosen among the paths that BFS algorithm traverses. BFS begins from the source, and progresses towards the target hop by hop. Potential cost of assigning a link (hop) to the flow is calculated in Line 13. Potential cost is the cost that will be incurred because of a conflict with other flows, in case the link is assigned to the flow.

Potential cost is kept as a set of flows that need to be canceled, if the link is assigned to the current flow. The value of potential cost is the sum of costs (traffic demands) of these flows. When path costs, $d[t]$, are calculated by Dijkstra's algorithm in Line 15, the costs are aggregated as a set (using set union operation) and finally converted to its numeric value using Equation 5.2:

$$\sum_{(x,y) \in cost(u,v)} \tau(x, y) \tag{5.2}$$

We cannot keep track of only the value of potential cost, because in that case the value of a flow that conflicts on multiple vertices would be counted multiple times. Keeping potential cost as a set of flows and then calculating the value when needed prevents multiple counting.

Figure 5.9 illustrates the case when edge (A, B) is assigned to flow (s, t), $f_{s,t}$. The nodes that are neighbors of A or B, except A or B themselves, are called x. We also assume that neither A nor B are s or t; therefore, they are called intermediate nodes.

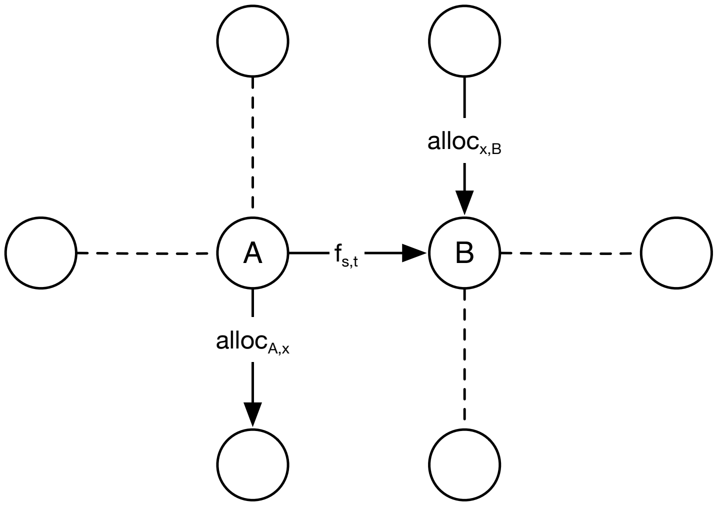


Figure 5.9: Potential cost of assigning (A, B) to flow (s, t): Potential-cost(s, t, A, B, alloc).

As a convention, we consider that, for intermediate nodes, incoming traffic conflicts with incoming traffic and outgoing traffic conflicts with outgoing traffic. Because intermediate nodes need to assign two radios to a flow, one for receiving and one for transmitting, this assumption covers all possible conflicts at each intermediate node. The fact that flow (s, t) is assigned link (A, B) implies that A has already used one of its radios to receive the traffic. When A transmits it to B, the flows that are transmitted to other neighbors of A need to be canceled.

Therefore, the cost includes the set of flows transmitted from A to x. A similar argument applies to B. B will use one of its radios to transmit $f_{s,t}$, therefore it should use the other one to receive it. By our convention, the flows that are received by B from x need to be canceled.

Formally, potential cost is defined in Equation 5.3:

$$\text{Potential-cost}(s,t,u,v,alloc) = \bigcup_{x} alloc(u,x) \cup \bigcup_{x} alloc(x,v),$$
$$\forall x \in Adj[u] \cup Adj[v] - \{u,v\} \text{ if } u \neq s, v \neq t \quad (5.3)$$

When node A is s, or node B is t, potential cost calculation is similar to the ones for intermediate nodes with some differences. Source and target nodes use a single radio for transmission or reception. When at least one of the radios are unassigned, the cost of allocating a new flow to the node is zero. When both radios are in use, the radio with the lower-cost flow set is considered to be conflicting.

Figure 5.10 shows BFS graph, calculated in Line 11, during the allocation of flow (H, C). Edge labels denote the values (costs) of potential conflicts. Because the traffic flow ends at C, which is 2 hops away from H, edges beyond 2 hops are not traversed.

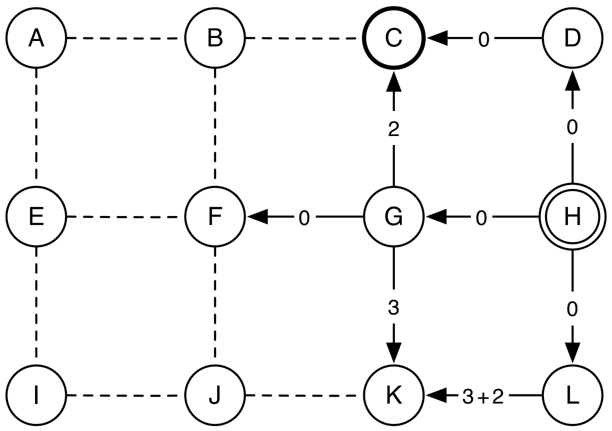


Figure 5.10: BFS graph for traffic flow from H to C, and the values of potential conflicts as edge labels.

Figure 5.11 presents existing allocations during the allocation of flow (H, C).

$f_{H,C}$ is the first 2-hop flow to be allocated, therefore all single-hop flows have been allocated. Given the allocation, we can calculate the potential cost of allocating any edge to a new flow.

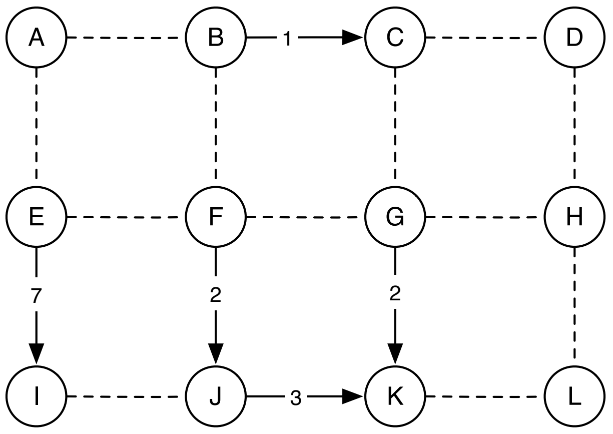


Figure 5.11: State of allocations when trying to allocate flow (H, C).

Consider routing the flow (H, C) over G. First, we examine the cost of allocating link (H, G). By definition, it consists of flows leaving H and flows arriving at G. There are no flows associated with H, so the first component does not contribute to the cost. There are no flows arriving at G either. There is one leaving G, $f_{G,K}$, but its cost contributes to the cost of allocating (G, C), not (H, G); because, by definition, outgoing traffic is considered to conflict with outgoing traffic. Therefore, the cost of allocating link (H, G) to flow (H, C) is $\emptyset$ with value zero, as shown in Figure 5.10.

Next, we calculate the cost of allocating link (G, C). We consider that one radio of G has already been allocated to the current flow, $f_{H,C}$, to receive the data form H. Therefore, we need to cancel other transmissions leaving G, to transmit data to C. There is one such transmission, from G to K, that belongs to flow (G, K). The other component of the cost consists of flows arriving at C. There is one such flow, $f_{B,C}$. Because C is the target node, it does not need to forward $f_{H,C}$ further, therefore it can use its other radio to receive from G without any conflicts. We conclude that the cost of allocating link (G, C) to flow (H, C) is $\{f_{G,K}\}$, and its value is 2.

While constructing G_{BFS} we only find the potential cost of allocating links.

Therefore, the algorithm also calculates the cost of allocating link (G, K) to $f_{H,C}$, even though the route will not lead to C. In case this edge is used, $f_{G,K}$ need not be canceled. However, the flow (J, K) arriving at node K needs to be canceled because K needs to forward the traffic to reach the target, C. Therefore, the cost of allocating (G, K) to flow (H, C) is $\{f_{J,K}\}$, with value 3.

Once BFS graph is constructed for a flow, lowest cost path from source to destination is found using Dijkstra's shortest path algorithm in Line 15. Because costs are sets of flows, aggregate path cost is calculated by the set union operator rather than the conventional addition of costs. Only when the path cost is compared to traffic demand of the current flow in Line 16, value of the path cost is calculated as defined in Equation 5.3.

In case the cost of allocating the flow is less than its traffic demand, we deallocate conflicting flows in Line 22, and allocate the current flow in Line 20. The allocated path is constructed by following parents, π, of each node beginning from the target back to source, Lines 18, 21.

5.2.1 Alternative Method

Algorithm 6 routes flows through a shortest path. Considering the fact that each additional hop uses more resources, creates more potential for conflicts, and increases latency, it is wise to use shortest paths. In some cases, though, longer paths may reduce conflicts by routing around busy nodes. Algorithm 7 is an alternative method that may allocate flows to routes longer than shortest paths.

The differences from Algorithm 6 are in Line 11 and Line 15. Rather than calculating edge costs for edges in the BFS DAG, it calculates costs of all edges in the graph, so that if any path is feasible for this flow, it will be allocated. This method is costlier in terms of computation, though it might allocate more traffic. Performances of these two methods will be compared in simulations.

Algorithm 7 Allocate fly-paths to flows over longer paths

```
 FLY-PATH-LP($G, F, \tau$)            ▷ Wireless graph, set of flows, traffic demands
    ...
10: for all $(s,t) \in$ SORT$_{<(\delta,-c)}(F)$ do
11:                                                          ▷ This line is removed
12:     for all $(u,v) \in E[G]$ do                          ▷ Calculate costs of edges
13:         cost$(u,v) \leftarrow$ POTENTIAL-COST$(s,t,u,v,alloc)$
14:     end for
15:     DIJKSTRA$(G, cost, s)$                    ▷ Find path costs to target, $d[t]$
    ...
26: end for
27: return path
```

5.2.2 Unallocated Flows

Our proposed methods do not guarantee allocating a path to a flow. Such a guarantee is possible, only when G is Hamiltonian, i.e., links could be assigned such that a single path visits all nodes. Obviously such a route maximizes the amount of traffic carried, but it is very impractical because of the latency it would introduce even between physically close nodes. Considering that the primary purpose of the wireless connections in a hybrid data center is to assist the wired infrastructure, it is acceptable that only the most important flows be allocated. Unallocated traffic could still be routed over the wired network.

A practical issue in deployment of proposed methods is when to use them. Wireless links allocated at one time might not be useful in future, so the allocation algorithms should be run periodically with the estimated traffic for that period. Configuration of wireless radios and channels to establish links is done once at the beginning of each period and is valid until the end of the period. A period should be long enough to compensate the cost of reconfiguring the wireless radios and links, and short enough to make traffic estimates reliable.

5.2.3 Algorithm Variations

In both of the algorithm versions, it is possible that a flow is unallocated even though there is a path between its source and target. This can happen in two cases. In the first case, the flow was never allocated, because the cost of allocating it was greater than the amount of traffic it carries. The path between its source and target is assigned to a different flow with larger traffic. In the second case, the flow was allocated initially, but later, was deallocated because of a conflicting higher-traffic flow. That flow gets deallocated in turn, because of an even higher-traffic flow that reestablishes the path of the first flow. In both cases, these unallocated flows can be allocated without incurring any cost, because the paths between their source and target pairs are already established by other flows. We call such allocations cost-free, or conflict-free, allocations.

Cost-free allocations can be handled at different times during the allocation algorithm. The simplest case is after all the flows have been assigned. We designate such versions with "+CF" suffix to the algorithm names (fly-path-SP+CF, fly-path-LP+CF). Cost-free allocations can also be evaluated after each allocation. In this case, there are two types of cost-free allocations: (1) flows that have been deemed infeasible up to that point in the algorithm, and (2) all unassigned flows, including infeasible ones and the ones that have not yet been considered for allocation. We suffix the former case versions with "-CF-infeasible" (fly-path-SP-CF-infeasible, fly-path-LP-CF-infeasible), and the latter case versions with "-CF-unassigned" (fly-path-SP-CF-unassigned, fly-path-LP-CF-unassigned). Together with the original versions we propose 8 algorithms in total.

5.3 Summary

In this chapter, we first propose a practical system model for hybrid wireless data centers. Each top-of-the-rack (ToR) switch is equipped with two radios communicating in 60 GHz band using 802.11ad. We show that three worldwide-available non-overlapping channels of 802.11ad is enough to achieve an interference-free assignment of channels to wireless links. We then propose multi-hop routing algorithms that offload traffic to wireless network under this system model. Our SP family of allocation algorithms route traffic over a shortest-path, hence the name, between source-destination pairs. Our LP family of allocation algorithms are not restricted to a shortest-path; they use longer paths as well. Each family contains an original algorithm and three modified versions. Modified versions run the original algorithm and also perform a conflict- or cost-free allocation at different steps. A cost-free allocation allocates flows for which there is already an established path between their source and destination. Because there are no changes to the wireless link configuration, no other traffic flow is deallocated — there are no costs paid in terms of allocated traffic.

Chapter 6

Data Center Traffic Model and Performance Evaluation

In this chapter, we present a method of randomly generating data center traffic in order to analyze the performances of traffic assignment methods we proposed in Chapter 5. We also present performance comparison of our proposed traffic assignment methods.

6.1 Data Center Traffic

Performances of the traffic assignment algorithms depend on the data center traffic. We generate a random traffic pattern between nodes according to characteristics of Cosmos data center traffic presented in [20] and [16]. Cosmos dataset is the traffic of a Map-Reduce [67] workflow running on O(1K) servers. In order to generate a similar traffic pattern to Cosmos, we first analyze its properties.

6.1.1 Cosmos Data Center Traffic Analysis

Cosmos dataset is not public. We reverse-engineer the traffic pattern given in Figure 11 of [20], reproduced in Figure 6.1.

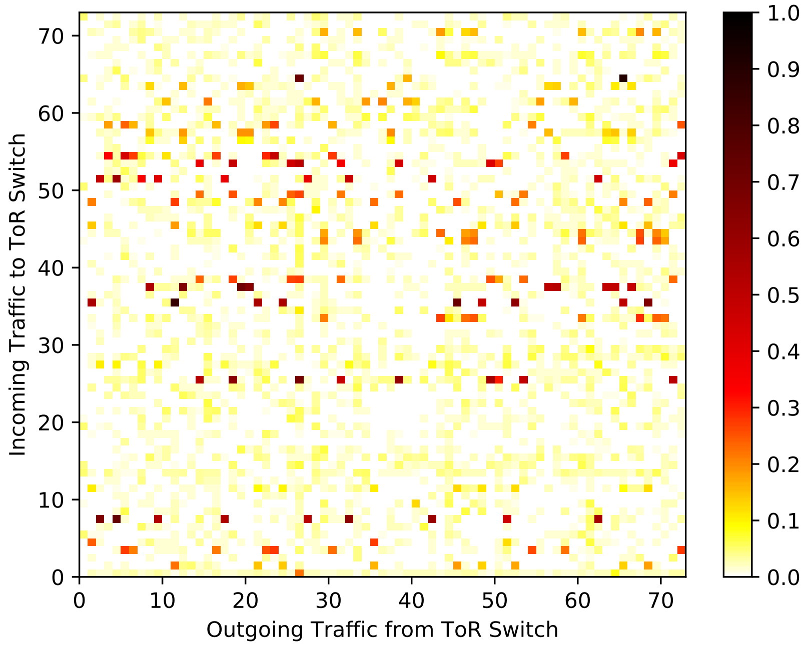


Figure 6.1: Original sample Cosmos traffic pattern.

Colors represent the amount of traffic exchanged between ToR switches in logarithmic scale with the largest value D corresponding to black. The deep red color at value 0.5 represents a traffic amount of $\sqrt{D}$, and traffic less than $D^{0.1}$ is represented with shades of yellow.

In order to find traffic flow values between ToR switches, we first map RGB values of the color scale to exponent values and then interpolate colors of the plot according to the mapping. Finally, we use exponent values to find linear traffic flow amounts.

RGB values of the color scale, however, does not match with the description above. The reason is because RGB values are in AdobeRGB color space, which is the color space of the Portable Document Format (PDF). When we transform colors into sRGB color space, we observe that the values below 0.1 correspond to yellow, as in the description. The fact that the plot seems to be generated with Python matplotlib library verifies the color transformation into sRGB space, because transformed values can easily be represented with a LinearSegmented-Colormap object of matplotlib.

We use the linear transformation defined in Equation 6.1 to find exponents. The linear coefficient results from our examination of the sRGB color scale. Exponent values from 1.00 to 0.33 are represented by an increase in red component from 0 to 255. Green and blue components for this value range are 0. Therefore, an exponent of 1.00 corresponds to $[0\ 0\ 0]_{\text{sRGB}}$ (black), 0.33 corresponds to $[255\ 0\ 0]_{\text{sRGB}}$ (red), and the values in between are linear interpolations of these two. Note that $0.33 = 1 - [255/255\ 0\ 0]_{\text{sRGB}} \cdot [0.67\ 0.24\ 0.09]^T$, therefore the equation holds.

$$\text{Exponent} = 1 - \begin{bmatrix} R/255 & G/255 & B/255 \end{bmatrix}_{\text{sRGB}} \cdot \begin{bmatrix} 0.67 \\ 0.24 \\ 0.09 \end{bmatrix} \tag{6.1}$$

Red component remains constant at 255 for all values below 0.33. Exponent values from 0.33 to 0.09 are represented by an increase in blue component from 0 to 255. Green component remains 0 for this value range. Therefore, an exponent of 0.33 corresponds to $[255\ 0\ 0]_{\text{sRGB}}$ (red), 0.09 corresponds to $[255\ 255\ 0]_{\text{sRGB}}$ (yellow). For the values below 0.09, red and blue components remain constant at 255 while green component increases from 0 to 255. An exponent of 0.00 is represented by $[255\ 255\ 255]_{\text{sRGB}}$ (white).

The next step is to find a linear traffic flow amount using the exponent. A value of 0 in linear space cannot be represented by an exponent; nevertheless, the original paper states that white corresponds to no traffic. Therefore, we subtract 1 from the traffic value raised to the exponent, as defined in Equation 6.2. Remember that D is the maximum amount of traffic between two ToR switches in the data center. When exponent is 0, traffic value is 0; when exponent is 1, traffic value is D, as in the definition.

$$\text{Traffic} = (D+1)^{\text{Exponent}} - 1 \tag{6.2}$$

Equation 6.3 defines the inverse transformation, i.e., the transformation from linear traffic flow amount to exponent value. A traffic value of 0 yields an exponent of 0, and a traffic value of D yields an exponent of 1, as in the definition.

$$\text{Exponent} = log_{D+1}(\text{Traffic} + 1) \tag{6.3}$$

The value of D is not given in the referenced literature. We assume that $D = 100$ in our simulations.

Now that we know linear traffic values flowing between ToR switches, we can analyze the traffic better by finding the total incoming traffic to or total outgoing traffic from a ToR switch.

Referenced work states two properties of the traffic: (1) most of the ToR pairs exchange low amounts of traffic, (2) hot ToRs exchange high amounts of traffic with only a few other ToRs.

Authors also state that the second property is apparent by horizontal and vertical streaks in Figure 6.1. We note, however, that horizontal streaks are more dominant than vertical streaks. In order to compare them better, we sort the data by total incoming and outgoing traffic in Figure 6.2. Each row and column is also sorted in descending order by the amount of traffic from left to right and bottom to top respectively.

The most important observation is the difference between incoming and outgoing traffic patterns. While almost all nodes send a high amount of traffic to $1-4$ other nodes (seen on the right-side plot), this traffic is received by only around 25 nodes (seen on the left-side plot). It is also notable that almost all of these 25 nodes receive their traffic from around 9 other nodes.

6.1.2 Traffic Generation

Based on our analysis of Cosmos data center traffic, we propose a 2-step procedure to randomly generate data center traffic: (1) randomly generate lighter-colored

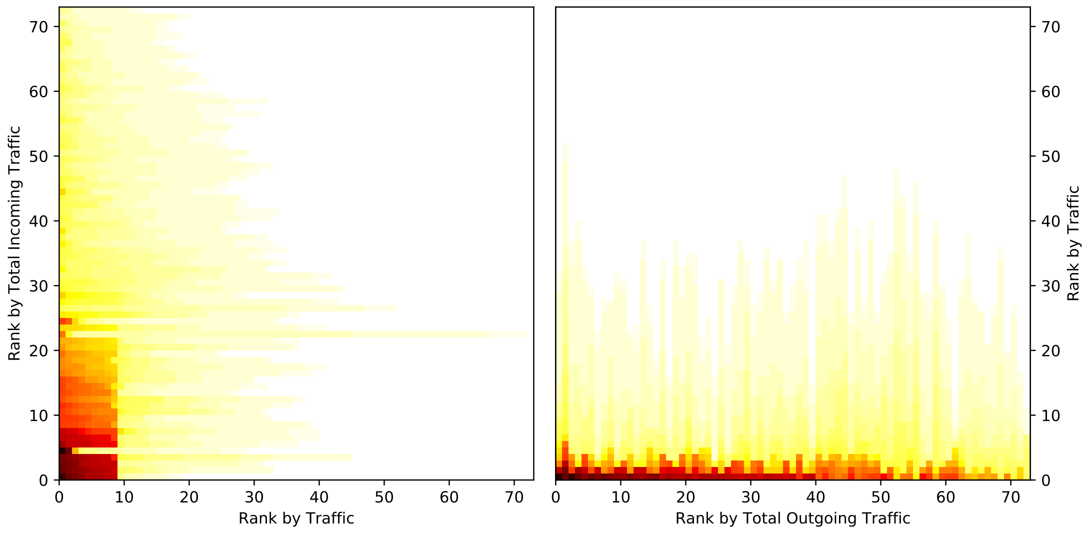


Figure 6.2: Original sample Cosmos traffic pattern sorted by total incoming traffic (left) and total outgoing traffic (right).

base traffic, and (2) randomly generate darker-colored hotspot traffic.

Figure 6.3 shows the distribution of original Cosmos pairwise traffic between nodes, except hotspot traffic (top 2%). Bar plots and curved line plot are associated with left y-axis and show the density of the distribution. Cumulative step plots are associated with right y-axis and show the cumulative distribution. Red lines show an exponential distribution fitted to original traffic data using *expon.fit()* function in *stats* module of *SciPy* python library [68]. The largest difference between original traffic data and the randomly generated one is around the smallest traffic values. (Note that the plot is cropped on the right to better focus on the large difference area. The difference between red and blue lines get smaller in higher values.)

In a data center traffic it is not realistic to have a very small amount of traffic flowing between every pair of nodes. Therefore, there is a gap between no traffic and the smallest traffic value in the original traffic. This gap does not exist in a randomly generated traffic because it is drawn from a continuous probability distribution. In the original data, smallest non-zero traffic value is around 0.05. We set any randomly generated traffic value lower than this threshold to zero. Resulting randomly generated data can be seen on the plot in orange. Up to the

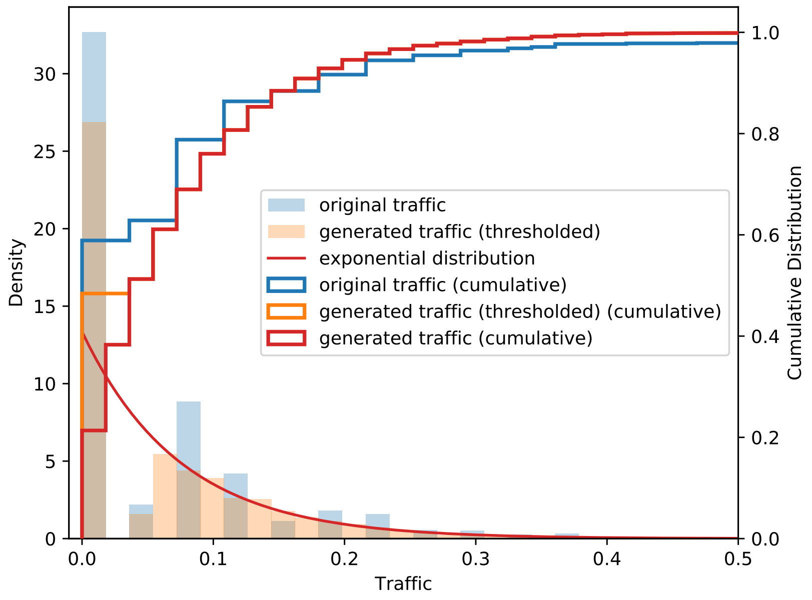


Figure 6.3: Pairwise traffic distribution, except hotspots, between nodes.

threshold value, cumulative distribution (orange line) is much closer to original traffic (blue line) than exponential distribution (red line). Orange and red lines are the same for values greater than the threshold.

To generate hotspot traffic, we use total traffic incoming to each node (sum of rows in Figure 6.1). Figure 6.4 shows distribution of total incoming traffic of original data. Brown lines show a lognormal distribution fitted to original data using *lognorm.fit()* function. Orange bar plot and cumulative step plot is a sample drawn from this distribution. As seen from the plot, lognormal distribution fits original traffic well, especially at lower values.

Details of how individual hotspot traffic is determined is presented in Algorithm 8. Base traffic is generated in Lines $1-5$. Total incoming traffic for each node is generated in Line 6. Total incoming traffic value is a target that will be reached by adding hotspot traffic over the base traffic.

For each node (Line 7), if target incoming traffic is greater than total incoming base traffic (Line 9), h nodes are selected randomly as hotspots for that

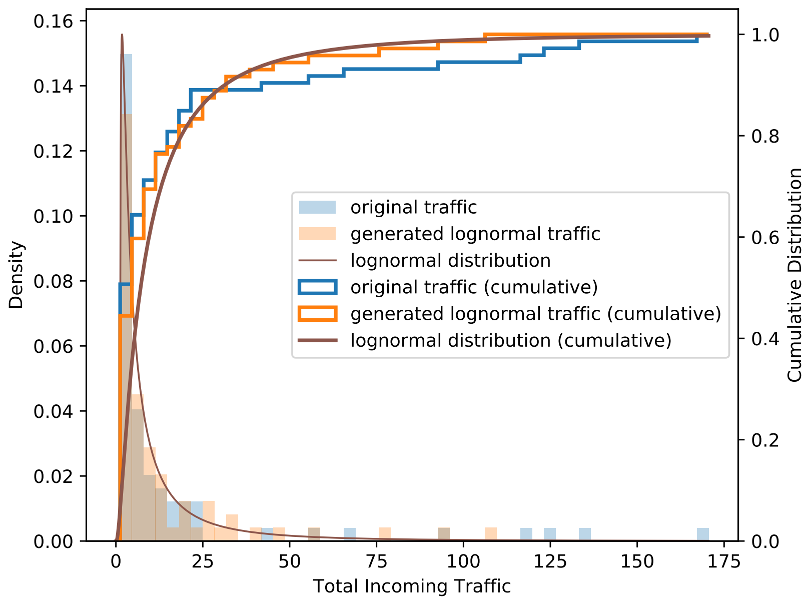


Figure 6.4: Total incoming traffic distribution for each node.

node (Line 10). This random selection ensures that hotspots are distributed uniformly over all nodes for outgoing traffic as depicted in Figure 6.2 (right). We make sure that hotspot traffic cannot be self-traffic in Line 11. We choose the amount of hotspot traffic from a normal distribution with mean μ' and $\sigma = \mu'/3$ (Lines 12, 13). This hotspot traffic is added to base traffic in Line 14. These last two steps introduce variance between hotspot traffic values to different nodes from this node.

It is possible that target incoming traffic is lower than total incoming base traffic of a node (Line 15). In that case, we clear traffic to randomly selected nodes in Lines $17-19$, until base traffic is lower than the target incoming traffic.

6.1.3 Generated Traffic Verification

We verify our proposed traffic generation algorithm by comparing the traffic it generates to original traffic. Figure 6.5 compares total outgoing traffic of nodes in a randomly generated traffic and the original traffic. Total outgoing traffic is a good comparison for validation, because it is not sampled from a random

Algorithm 8 Generate random data center traffic

```
GENERATE-TRAFFIC($n, h$, min-traffic, $\lambda, \mu, \sigma$)
 1: base-traffic $\leftarrow$ EXP-DIST($\lambda, n \times n$)
 2: base-traffic[base-traffic $<$ min-traffic] $\leftarrow 0$
 3: for $i \in \{0, 1, \dots n-1\}$ do
 4:     base-traffic[$i, i$] $\leftarrow 0$                    ▷ Clear self-traffic
 5: end for
 6: incoming-traffic $\leftarrow$ LOGNORMAL-DIST($\mu, \sigma, n$)
 7: for $i \in \{0, 1, \dots n-1\}$ do
 8:     base-incoming $\leftarrow$ SUM(base-traffic[$i$])
 9:     if base-incoming $<$ incoming-traffic[$i$] then
10:         hotspots $\leftarrow$ UNIFORM-DIST($0, n-1, h$)
11:         hotspots $\leftarrow$ hotspots $-\{i\}$
12:         $\mu' \leftarrow \frac{\text{incoming-traffic}[i] - \text{base-incoming}}{h}$
13:         hotspot-traffic $\leftarrow$ NORMAL-DIST($\mu', \mu'/3, h$)
14:         base-traffic[$i$, hotspots] $+=$ hotspot-traffic
15:     else
16:         repeat
17:             rand-node $\leftarrow$ UNIFORM-DIST($0, n-1, 1$)
18:             base-traffic[$i$, rand-node] $\leftarrow 0$
19:         until base-incoming $<$ incoming-traffic[$i$]
20:     end if
21: end for
22: return CLIP-MIN(base-traffic, 0)
```

distribution in the traffic generation algorithm; rather, it emerges from the whole procedure. It can be seen that the original and generated distributions are similar.

Overall looks of original and generated traffic could be compared in Figure 6.6. Properties of original traffic that are explained in Chapter 6.1.1 could be observed in generated traffic as well. Horizontal stripe pattern indicates that distribution of incoming traffic is more skewed than outgoing traffic, which is rather uniform among nodes.

Comparison of incoming traffic is given in Figure 6.7. Note that generated distribution of hotspots has similar patterns to that of original traffic. Not every node has exactly h hotspots as in the original traffic, because the amount of hotspot traffic for each node is drawn from a normal distribution. The variation in distribution sometimes yields a very low amount of traffic that looks like base

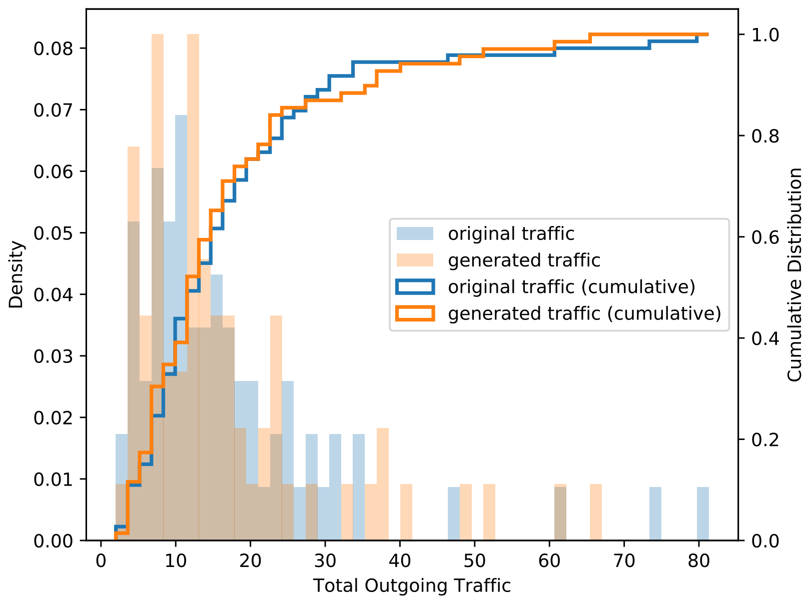


Figure 6.5: Comparison of total outgoing traffic distribution.

traffic. Another similarity is apparent in low-traffic nodes. In the original data, nodes with the least amount of traffic communicate with fewer other nodes compared to higher-traffic nodes. This pattern is visible in generated data as well, because of the logic in Lines 17 − 19.

Comparison of outgoing traffic is presented in Figure 6.8.

Finally, we verify that random traffic generation can scale to different network sizes. When the number of nodes is different than the original (73), we scale all parameters, except min-traffic threshold, in proportion. Characteristic traffic patterns can be seen in different network sizes in Figure 6.9.

6.2 Simulations and Results

We report results that are averaged over 30 different randomly generated traffic for each network size. In order to compare different network sizes better, we use only square grid networks.

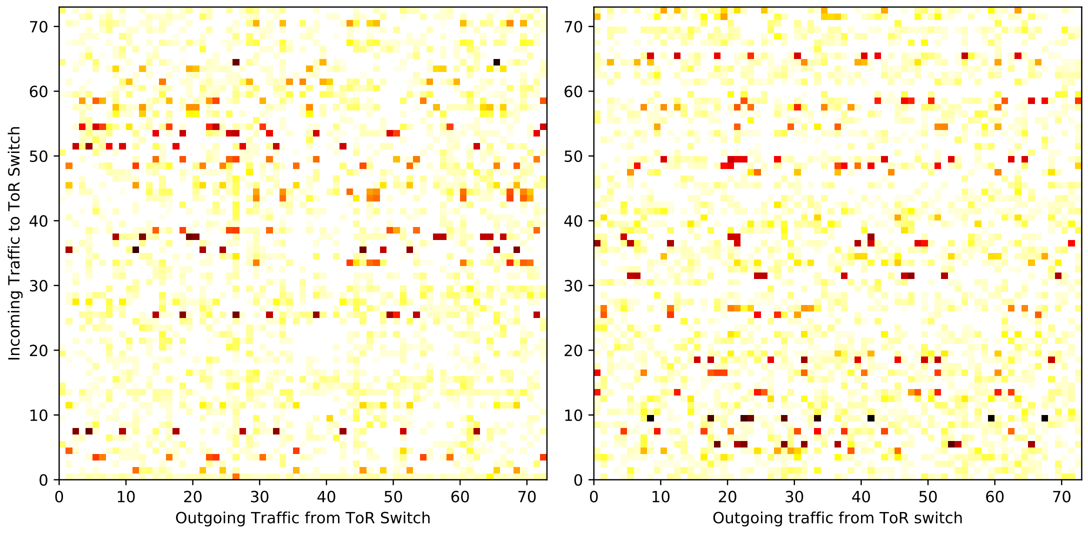


Figure 6.6: Comparison of original (left) and generated traffic (right).

Figure 6.10 shows that the mean traffic per source-destination pair is the same over different network sizes. From this result we conclude that the method of scaling traffic generation parameters to different network sizes is successful. Error bars show one standard deviation over the generated traffic patterns. Because smaller networks have fewer number of pairs of nodes (smaller sample size), their standard deviation is higher as a statistical phenomenon.

Figure 6.11 shows the amount of allocated traffic for different network sizes. For the smallest two network sizes of 9 and 16 nodes, Shortest-path (SP) and longer-path (LP) algorithms perform closely, with LP version performing 7 and 11% better than the corresponding SP version, respectively. As the network gets larger, LP allocates $24-48\%$ more traffic than SP. The increase in allocation is explained by larger networks having an abundance of alternative paths longer than the shortest paths. LP exploits these longer paths to allocate more traffic, while SP is restricted to using shortest paths. Lack of alternative routes causes more conflicts between flows, therefore, allocation of less traffic.

Table 6.1 presents performance improvement of cost-free algorithms over the original ones. CF and CF-infeasible versions perform consistently better (3.22% on the average) than the original versions of SP and LP in all network sizes.

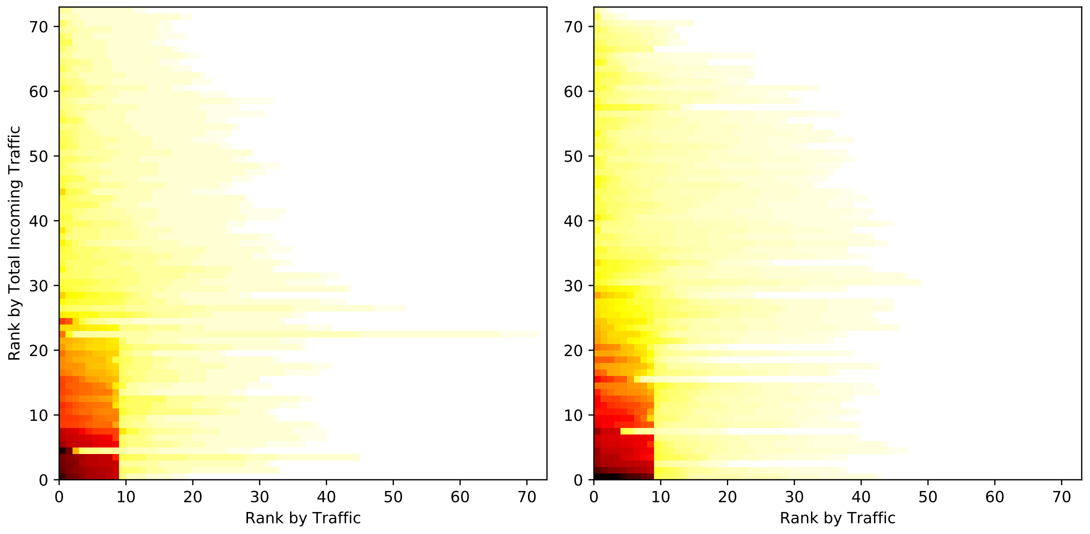


Figure 6.7: Comparison of original (left) and generated traffic (right) sorted by total incoming traffic.

Both versions perform similarly, though CF has a slight advantage of 0.51% over CF-infeasible overall. Considering that CF is also simpler to implement and faster in running time than CF-infeasible, it is practically the best alternative.

Performance of CF-unassigned has a more complicated profile. Compared to the original version, it performs 1.13% better for SP and 1.72% better for LP, on the average. However, it does not perform better consistently across all network sizes. SP-CF-unassigned performs 0.40% worse than the original SP for 16- and 36-node networks. LP-CF-unassigned performs 0.45% worse than the original LP for 9- and 36-node networks.

Compared to CF and CF-infeasible versions, CF-unassigned performs 1.73% worse on the average. SP-CF-unassigned performs consistently worse (2.47% on the average) than SP+CF and SP-CF-infeasible across different network sizes. Unlike its SP counterpart, LP-CF-unassigned performs better than (a) LP+CF (1.07% on the average) for 25- and 49-node networks and (b) LP-CF-infeasible (1.84% on the average) for 25-, 49-, and 64-node networks.

Compared to shorter routes that SP allocates, longer routes that LP allocates have higher chance of connecting yet-to-be-considered (*unassigned*)

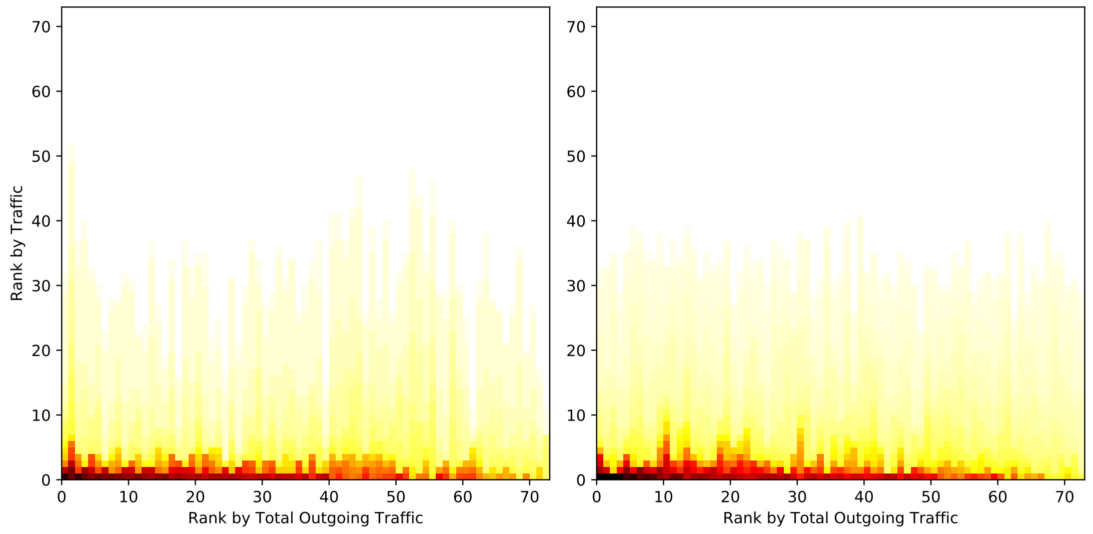


Figure 6.8: Comparison of original (left) and generated traffic (right) sorted by total outgoing traffic.

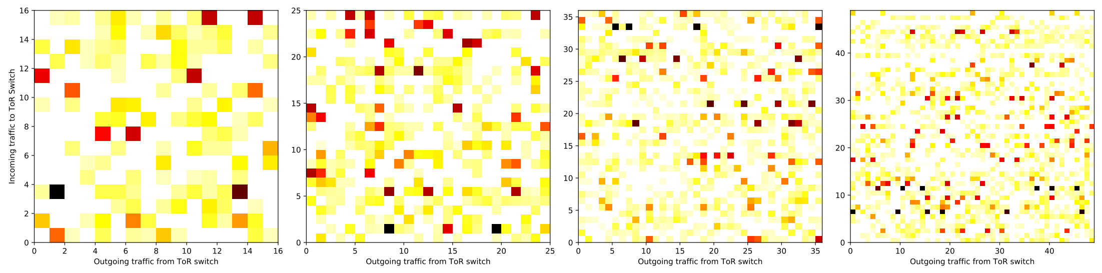


Figure 6.9: Randomly generated traffic for network sizes of 16, 25, 36, and 49 nodes.

source-destination pairs. Therefore, LP version of CF-unassigned is more successful than its SP version.

Because the performance of CF-unassigned is not consistent across different variables, we conclude that it is not a good alternative for reliability purposes.

As a measure of how much traffic can be offloaded from wired to wireless network, Figure 6.12 shows allocated traffic (wireless) as a percentage of total data center traffic (wired and wireless). For the smallest size cloud, wireless network is able to carry a large portion, 65 − 75%, of the traffic. The ratio of allocated traffic falls as the network grows, because both the number of source-destination pairs and the distance between them increase with the network size, causing more

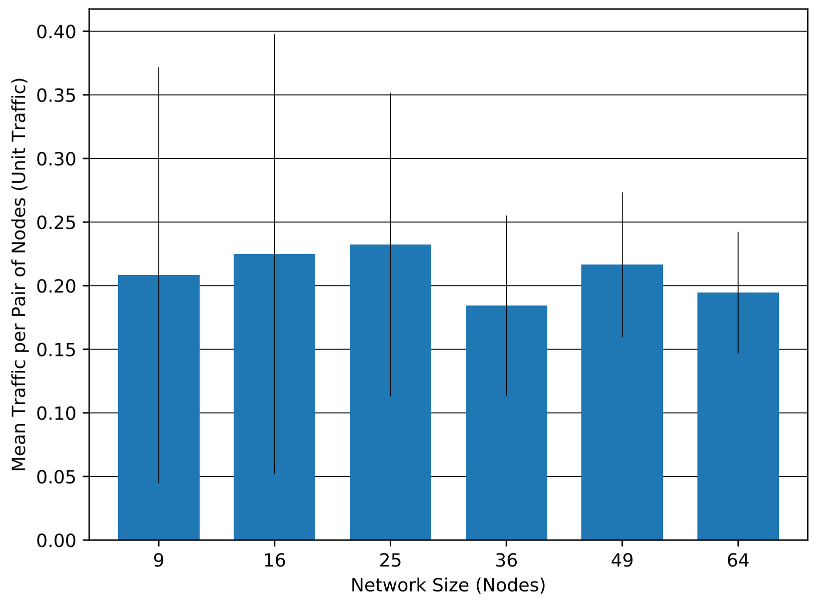


Figure 6.10: Mean traffic amount (in unit traffic) for different network sizes.

	Performance improvement over SP			Performance improvement over LP		
Network Size	SP+CF	SP-CF-infeasible	SP-CF-unassigned	LP+CF	LP-CF-infeasible	LP-CF-unassigned
9	3.57%	3.59%	0.51%	3.01%	2.53%	-0.27%
16	2.49%	2.55%	-0.75%	2.98%	2.88%	0.60%
25	2.57%	2.42%	1.83%	2.78%	0.84%	2.95%
36	4.95%	3.65%	-0.05%	3.23%	3.81%	-0.64%
49	4.61%	4.87%	2.96%	3.69%	2.56%	5.73%
64	4.81%	4.22%	2.28%	3.05%	1.61%	1.94%
Average	**3.83%**	**3.55%**	**1.13%**	**3.12%**	**2.37%**	**1.72%**

Table 6.1: Performance improvement of cost-free versions over the original algorithms.

conflicts. In the largest network, wireless network is able to carry at least 20% of the traffic.

Figure 6.13 shows mean path length of allocated flows. Remember that SP allocates flows to a shortest path between source and destination, whereas LP can use longer paths. The longest path SP can assign is bounded by the diameter of the network, i.e., the distance between most distant two nodes. Length of an LP-allocated path is bounded by the network size minus one, because in the worst case LP may traverse all other nodes. In practice, though, LP-assigned paths are at most half of the network size.

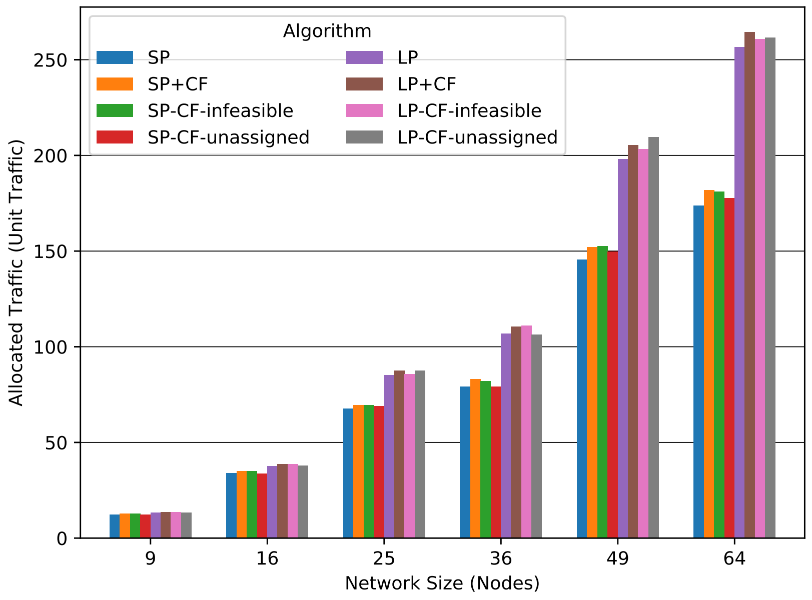


Figure 6.11: Allocated traffic.

Results show that path length increases quickly with network size for LP algorithms. Path length of SP algorithms remain very small, increasing by less than 1 hop from 1.97 hops at 9 nodes to 2.88 hops at 64 nodes. Path length of LP algorithms increase by more than 7 hops from 2.72 hops at 9 nodes to 9.81 nodes at 64 nodes. Because the path length determines latency, LP algorithms lose their advantage of allocating more traffic by trading it with higher latency.

Variations of SP have similar path length properties. Contrary to original SP algorithm, CF versions are not restricted to allocating a flow over a shortest path. A flow can be assigned to a longer path as long as it provides a path from source to destination. Despite the fact that CF versions can assign longer paths, in practice, assigned path length is at most 2 hops more than the shortest path, because existing paths are all chosen among shortest paths. Mean path length of CF versions (2.45 hops on the average) is even lower than that of the original SP algorithm (2.66 hops on the average), because most of the cost-free allocations are for close source-destination pairs.

Figure 6.14 shows completion time of allocated traffic. Completion time of a flow is determined by its share of bandwidth at the bottleneck link along its

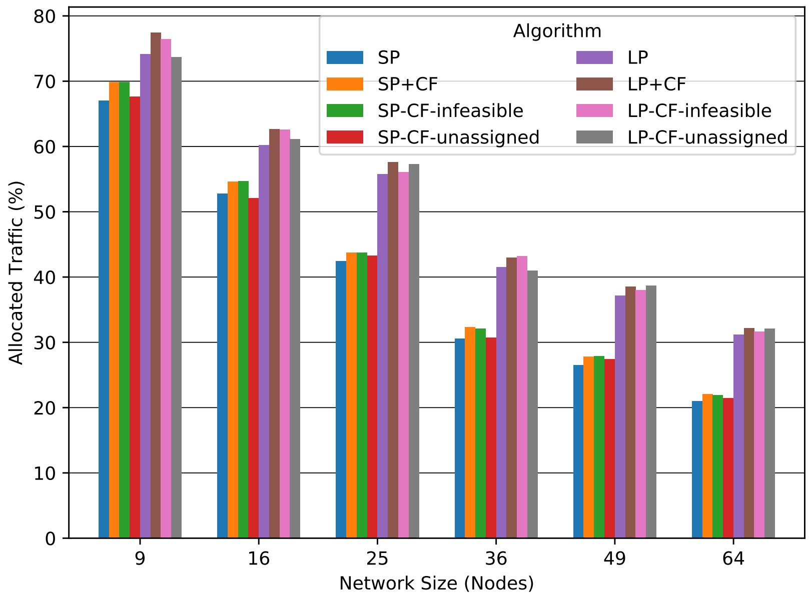


Figure 6.12: Allocated traffic as a percentage of total data center traffic.

route. Flows share bandwidth of a bottleneck link proportional to the traffic they carry. Consider that a link is a bottleneck for two flows that carry 2 and 3 units of traffic respectively. The first flow is allocated 2/5 of, and the second flow is allocated 3/5 of the bandwidth of the link.

Completion time of all allocated traffic is the completion time of the latest finishing flow, which is determined by the link that carries the most traffic, i.e., bottleneck of the network.

The traffic allocated by LP algorithms takes 87% longer to finish on the average, because LP algorithms allocate more traffic than SP algorithms. However, the benefit of allocating 35% more traffic on the average seems to be offset by increased completion time. In order to examine the trade-off between allocated traffic and completion time in more detail, Figure 6.15 shows achieved throughput, calculated by dividing the amount of allocated traffic by completion time.

The increase in throughput with increased network size shows that algorithms are able to allocate more simultaneous traffic distributed over the network by utilizing local wireless network resources. SP algorithms achieve 35% higher

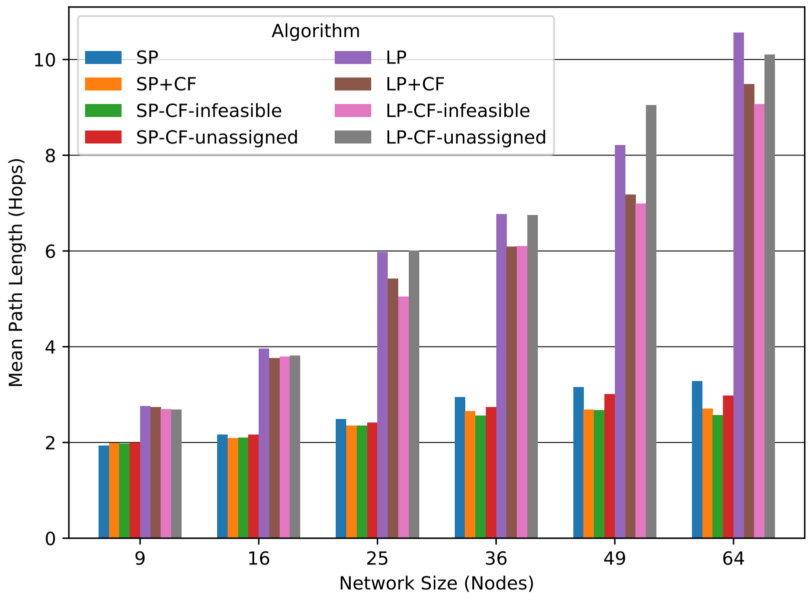


Figure 6.13: Mean path length of allocated flows.

throughput on the average than LP algorithms. The increase in throughput for SP algorithms from 1.62 at 9 nodes to 3.79 at 64 nodes (2.34-fold) is also higher than the increase for LP algorithms from 1.56 at 9 nodes to 2.58 at 64 nodes (1.65-fold). Results show that even though LP algorithms allocate more traffic, they do so at the cost of completion time, compared to SP algorithms. The reason LP algorithms achieve lower throughput is because longer routes cause more flows to share the same link, therefore increasing the number of flows restricted by bottlenecks.

The difference between algorithm variations is also important. Remember that for both SP and LP, CF and CF-infeasible variations allocate 3.22% more traffic on the average than the original versions. They also achieve relatively lower completion time, therefore higher throughput of 2.89% on the average. We conclude that these algorithm versions are preferable over the original versions.

Finally, Figure 6.16 shows bandwidth utilization per wireless link established. As discussed earlier, completion time of a flow is determined by its share of bandwidth at the bottleneck link along its route. There is no benefit for the flow to consume more bandwidth at non-bottleneck links. To the contrary, doing

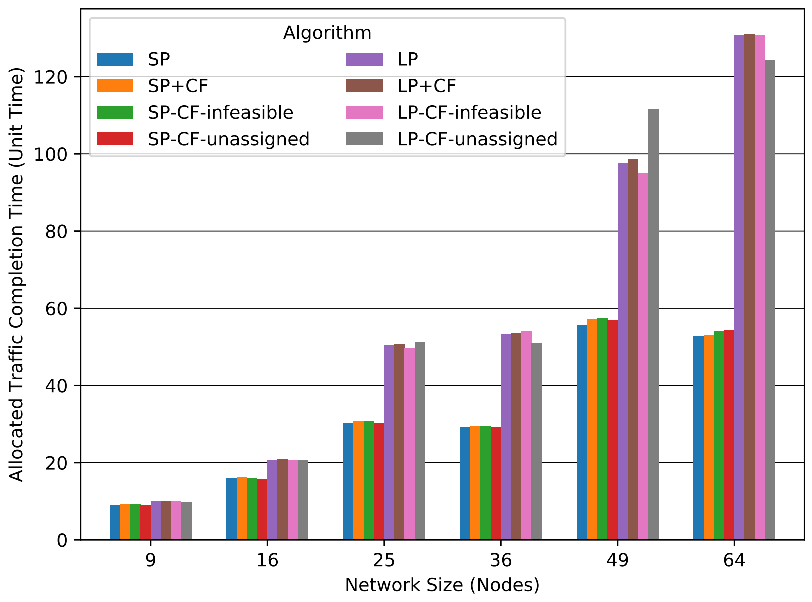


Figure 6.14: Completion time of allocated traffic.

so would increase buffering cost at each hop. Therefore, we consider that a flow consumes the same bandwidth at each link along its route. By definition, bottleneck links are utilized 100%. In an ideal allocation, all wireless links are fully utilized.

Results show that bandwidth utilization is not affected by the network size. Algorithms are able to utilize expanding wireless networking resources. SP algorithms achieve 77% bandwidth utilization on the average, while LP algorithms achieve 69%. SP algorithms perform better than LP algorithms for the same reason that SP algorithms achieve more bandwidth: longer routes allocated by LP algorithms force flows through the same link, therefore reducing bandwidth utilization at non-bottleneck links.

Similar to previous results, CF and CF-infeasible versions are more favorable by performing 1.76% better on the average than the others in terms of bandwidth utilization per link.

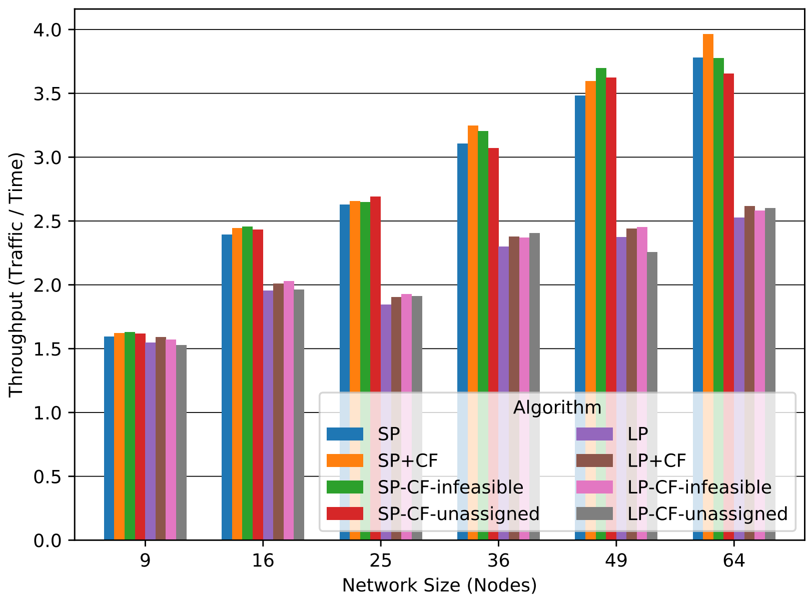


Figure 6.15: Effective bandwidth.

6.3 Summary

In order to test the performance of our proposed traffic allocation algorithms, we propose a method to randomly generate data center traffic based on a real-world data center traffic pattern. First, we analyze the properties of the real-world traffic pattern. Then, we find probability distributions that fit well onto some of those properties. Finally, we propose a procedure that uses these distributions to randomly generate a traffic pattern. Our proposed traffic generation method is able to generate traffic patterns for different network sizes. We verify the results by comparing the properties of the real-world and randomly generated traffic patterns.

We run our proposed multi-hop allocation algorithms on randomly generated traffic and evaluate results in terms of different metrics. LP algorithms allocate 35% more traffic than SP algorithms, but SP algorithms achieve 58% lower latency between source-destination pairs by keeping the routes short. The percentage of allocated traffic to total traffic decreases from 69 and 75% to 22 and 32% with increased network size for SP and LP algorithms respectively. The

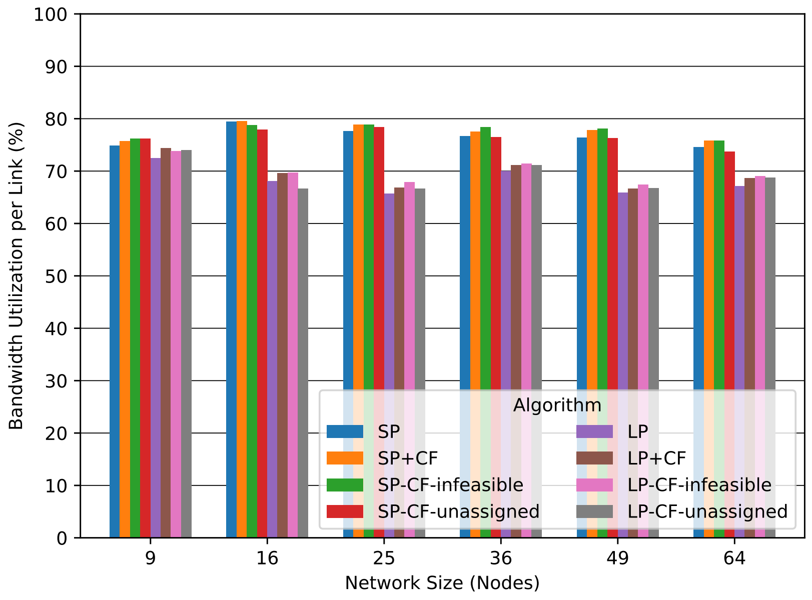


Figure 6.16: Bandwidth utilization per link.

mean latency increases by only 1 hop for SP, and by 7 hops for LP from smallest to largest network size. SP-allocated traffic finishes in 46% less time than LP-allocated traffic, but because the amount of allocated traffic is different for each family, we measure throughput to have a fair comparison. SP algorithms achieve 35% higher throughput, i.e., carry more traffic per unit time than LP algorithms. Finally, we measure bandwidth utilization of wireless links. SP algorithms use 77% of the available bandwidth at each link for all network sizes, while LP algorithms use 69%.

Among cost-free versions, CF and CF-infeasible perform better than the original versions in almost every metric and network size. CF and CF-infeasible perform similarly, but CF is simpler in terms of implementation and faster in running time than CF-infeasible, therefore is more practical. CF-unassigned is not preferred for reliability purposes because its performance is inconsistent across different variables.

Chapter 7

Conclusion and Future Work

We examine server and wireless network resource allocation in data centers. The resources that are used by a virtual machine in the cloud are called server resources. These typically consist of, but are not limited to, CPU, memory, storage, IO, and GPU. Wireless network resources consist of radios and channels.

We model server resource allocation problem as a vector bin packing problem. Server resources represent a multidimensional vector space. The problem is to allocate a set of virtual machine requests into a set of physical machines. We propose a metric-based approach. A metric quantifies the quality of an allocation, so that alternative allocations could be compared. First, we propose a design criteria for metrics that lead to efficient allocations. Then, we propose two novel metrics obeying the criteria. Finally, we propose allocation heuristics that use these metrics to allocate a set of VMs into a set of PMs.

Vector bin packing with heterogeneous bins (VBHPB) benchmark results show that our metrics are able to find feasible solutions to more allocation problem instances compared to other metrics from the literature.

Wireless network resource allocation problem we consider is to offload the maximum amount of traffic from wired to wireless network, so that data centers can be designed accordingly. We first propose a system model for hybrid wireless data centers. In our system model, each top-of-the-rack (ToR) switch has two radios operating in 60 GHz band using 802.11ad. We prove that three world-wide available channels of 802.11ad can be assigned without any conflicts to any configuration of links that could be established between radios in the system.

Given traffic demand between ToR switches, our proposed algorithms route traffic over multi-hop wireless links. Not all of the traffic could be carried over wireless network. Unassigned traffic is routed over wired network. In practice, traffic demand between ToR switches is not known beforehand; therefore, our algorithms are run periodically with estimated traffic flow for that period.

We also propose a method to randomly generate data center traffic. We analyze a real data center traffic and fit probability distributions that match some of its qualities. We use output of these probability distributions to generate traffic demands between ToR switches. Our proposed method can generate traffic for different network sizes.

We evaluate the performance of wireless network resource allocation problem using the randomly generated data center traffic. Results show that our methods are able to offload a significant amount of traffic from wired to wireless network. At the same time, they achieve low latency, high throughput, and high utilization of wireless link bandwidth.

As a future work, the server resource allocation methods we proposed could be extended for allocating requests with proximity constraints, so that VMs that would serve for the same application could communicate faster among each other. Such a method could consider cloud as a tree of resources, and allocate subsets of a set of VMs in neighboring sub-trees according to fitness values.

It is also possible to allocate wireless network resources so that the network distance between communicating VMs are reduced. The benefit of using wireless resources for this purpose is the flexibility of configuration. Compared to static wired network architecture, wireless links can be established dynamically to satisfy the most demanding communication between VMs, which change over time according to workload. Bringing VMs closer over the wireless network also reduces the need for VM migration due to communication requirements. The cost of migration could be avoided in this manner.

Wireless communication cannot reduce the network distance between VMs that are physically distant to each other. Therefore, VMs should be allocated in racks that are physically close to each other. The final future work we propose is a VM allocation method that considers the constraints of server, wired, and wireless network resources together.

Bibliography

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Special Publication 800-145.*, September 2011.

[2] Amazon.com, Inc., "Amazon Elastic Compute Cloud (Amazon EC2)." `http://aws.amazon.com/ec2/`, 2006. Online, accessed 26 June 2012.

[3] Microsoft Corporation, "Microsoft Azure." `https://azure.microsoft.com/en-us/free/services/virtual-machines/`, 2012. Online, accessed 13 May 2020.

[4] Rackspace Technology, Inc., "Cloud Server and Virtual Server Hosting by Rackspace." `http://www.rackspace.com/cloud/servers`, 2008. Online, accessed 26 June 2012.

[5] P. Smulders, "Exploiting the 60 GHz band for local wireless multimedia access: prospects and future directions," *IEEE Communications Magazine*, vol. 40, no. 1, pp. 140 – 147, 2002.

[6] K. Ramach, R. Kokku, and R. Mahindra, "60 GHz Data-Center Networking: Wireless =>Worry less?." NEC Research Paper, 2008.

[7] IEEE, "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band," *IEEE Std 802.11ad-2012 (Amendment to IEEE Std 802.11-2012, as amended by IEEE Std 802.11ae-2012 and IEEE Std 802.11aa-2012)*, pp. 1 – 628, 2012.

[8] T. Nitsche, C. Cordeiro, A. B. Flores, E. W. Knightly, E. Perahia, and J. C. Widmer, "IEEE 802.11ad: Directional 60 GHz communication for multi-gigabit-per-second Wi-Fi [Invited Paper]," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 132 – 141, 2014.

[9] J.-Y. Shin, E. G. Sirer, H. Weatherspoon, and D. Kirovski, "On the feasibility of completely wireless datacenters," in *8th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 3 – 14, 2012.

[10] H. Vardhan and R. Prakash, "Concurrency in polygonally arranged wireless data centers with all line-of-sight links," in *2014 International Conference on Computing, Networking and Communications (ICNC)*, pp. 716 – 720, 2014.

[11] H. Vardhan, N. Thomas, S. Ryu, B. Banerjee, and R. Prakash, "Wireless data center with millimeter wave network," in *2010 IEEE Global Telecommunications Conference (GLOBECOM)*, pp. 1 – 6, 2010.

[12] H. Vardhan, S.-R. Ryu, B. Banerjee, and R. Prakash, "60 GHz wireless links in data center networks," *Computer Networks*, vol. 58, pp. 192 – 205, 2014.

[13] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "FireFly: A reconfigurable wireless data center fabric using free-space optics," in *2014 ACM Conference on SIGCOMM (SIGCOMM)*, pp. 319 – 330, 2014.

[14] J. Luo, Y. Guo, S. Fu, K. Li, and W. He, "Virtual resource allocation based on link interference in cayley wireless data centers," *IEEE Transactions on Computers*, vol. 64, no. 10, pp. 3016 – 3021, 2015.

[15] X. Wu, S. Zhang, and A. Özgür, "STAC: Simultaneous transmitting and air computing in wireless data center networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 4024 – 4034, 2016.

[16] S. Kandula, J. Padhye, and V. Bahl, "Flyways to de-congest data center networks," Tech. Rep. MSR-TR-2009-109, Microsoft, August 2009.

[17] Y. Cui, H. Wang, and X. Cheng, "Channel allocation in wireless data center networks," in *2011 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1395 – 1403, 2011.

[18] Y. Cui, H. Wang, and X. Cheng, "Wireless link scheduling for data center networks," in *5th International Conference on Ubiquitous Information Management and Communication (ICUIMC)*, 2011.

[19] Y. Cui, H. Wang, X. Cheng, and B. Chen, "Wireless data center networking," *IEEE Wireless Communications*, vol. 18, no. 6, pp. 46 – 53, 2011.

[20] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 38 – 49, 2011.

[21] W. Zhang, X. Zhou, L. Yang, Z. Zhang, B. Y. Zhao, and H. Zheng, "3D beamforming for wireless data centers," in *10th ACM Workshop on Hot Topics in Networks (HotNets)*, pp. 1 – 6, 2011.

[22] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *1st ACM Workshop on Research on Enterprise Networking (WREN)*, pp. 65 – 72, 2009.

[23] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *10th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pp. 267 – 280, 2010.

[24] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *2010 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1154 – 1162, 2010.

[25] V. Shrivastava, P. Zerfos, K. won Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *2011 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 66 – 70, 2011.

[26] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *2012 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 963 – 971, 2012.

[27] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pp. 187 – 198, 2012.

[28] M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens, "A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 11, pp. 1497 – 1508, 2011.

[29] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *9^{th} IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 92 – 99, 2009.

[30] 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, 2012.

[31] X. Ye, Y. Yin, and L. Lan, "Energy-efficient many-objective virtual machine placement optimization in a cloud computing environment," *IEEE Access*, vol. 5, pp. 16006 – 16020, July 2017.

[32] X. Zhang, Y. Tian, and Y. Jin, "A knee point-driven evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 761 – 776, 2015.

[33] H. Singh, M. hau Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465 – 481, 2000.

[34] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter," *Journal of Network and Computer Applications*, vol. 45, pp. 108 – 120, October 2014.

[35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 124 – 131, May 2009.

[36] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand clouds," in *IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 91 – 98, 2011.

[37] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *2008 ACM/IEEE Conference on Supercomputing (SC)*, pp. 1 – 12, 2008.

[38] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems," *Management Science*, vol. 21, no. 12, pp. 1417 – 1427, 1975.

[39] E. Arzuaga and D. R. Kaeli, "Quantifying load imbalance on virtualized enterprise servers," in *1st Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW)*, pp. 235 – 242, 2010.

[40] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: Application to the machine reassignment problem," *Annals of Operations Research*, vol. 242, pp. 161 – 194, July 2016.

[41] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Sandpiper: Blackbox and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923 – 2938, 2009.

[42] M. Mishra and A. Sahoo, "On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach," in *IEEE International Conference on Cloud Computing (CLOUD)*, pp. 275 – 282, 2011.

[43] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *2014 IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1033 – 1041, April 2014.

[44] D. S. Johnson, "Approximation algorithms for combinatorial problems," *The Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256 – 278, 1974.

[45] L. U. Rina Panigrahy, Kunal Talwar and U. Wieder, "Heuristics for vector bin packing," Tech. Rep. , Microsoft Research, January 2011.

[46] R. Panigrahy, V. Prabhakaran, K. Talwar, U. Wieder, and R. Ramasubramanian, "Validating heuristics for virtual machines consolidation," Tech. Rep. , Microsoft Research, January 2011.

[47] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation algorithms for virtualized service hosting platforms," *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962 – 974, 2010.

[48] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Journal of Network and Computer Applications*, vol. 66, pp. 106 – 127, May 2016.

[49] J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 64, pp. 23 – 42, April 2016.

[50] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: Flexible wireless links for data centers," in *ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pp. 443 – 454, 2012.

[51] Y. Zhu, X. Zhou, Z. Zhang, L. Zhou, A. Vahdat, B. Y. Zhao, and H. Zheng, "Cutting the cord: A robust wireless facilities network for data centers," in *20th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pp. 581 – 592, 2014.

[52] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 1 – 12, 2007.

[53] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41 – 54, 2005.

[54] Y. Katayama, T. Yamane, Y. Kohda, K. Takano, D. Nakano, and N. Ohba, "MIMO link design strategy for wireless data center applications," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 3302 – 3306, 2012.

[55] T. Yamane and Y. Katayama, "An effective initialization of interference cancellation algorithms for distributed mimo systems in wireless datacenters," in *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 4249 – 4254, 2012.

[56] Y. Yu, C. Chuang, H. Lin, and A. Pang, "Efficient multicast delivery for wireless data center networks," in *38th Annual IEEE Conference on Local Computer Networks (LCN)*, pp. 228 – 235, 2013.

[57] L. Zhu, J. Wu, G. Jiang, L. Chen, and S.-K. Lam, "Efficient hybrid multicast approach in wireless data center network," *Future Generation Computer Systems*, vol. 83, pp. 27 – 36, June 2018.

[58] E. Baccour, S. Foufou, R. Hamila, and M. Hamdi, "A survey of wireless data center networks," in *49th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1 – 6, 2015.

[59] A. S. Hamza, J. S. Deogun, and D. R. Alexander, "Wireless communication in data centers: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1572 – 1595, 2016.

[60] TeamJ19ROADEF2012, "Variable-Size-Vector-Bin-Packing." `https://github.com/TeamJ19ROADEF2012/Variable-Size-Vector-Bin-Packing`, 2016. Online, accessed 15 May 2019.

[61] NumFOCUS Foundation, "pandas: Python Data Analysis Library." `https://pandas.pydata.org`, 2008. Online, accessed 15 May 2019.

[62] NumFOCUS Foundation, "NumPy." `https://www.numpy.org`, 2006. Online, accessed 15 May 2019.

[63] V. G. Vizing, "On an estimate of the chromatic class of a p-graph," *Diskret Analiz*, no. 3, pp. 23 – 30, 1964.

[64] I. Holyer, "The NP-completeness of edge-coloring," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 718 – 720, 1981.

[65] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, pp. 20 – 53, October 2016.

[66] K. Nguyen and H. Sekiya, "TCP behavior on multi-gigabit IEEE 802.11ad link," in *2020 International Conference on Green and Human Information Technology (ICGHIT)*, pp. 58 – 61, 2020.

[67] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107 – 113, 2008.

[68] NumFOCUS Foundation, "SciPy: Scientific Computing Tools for Python." `https://www.scipy.org/`, 2001. Online, accessed 04 May 2020.