MONASH University

Multi-objective Virtual Machine Management in Cloud Data Centers

Md Hasanul Ferdaus

Supervisor: Professor Manzur Murshed
Associate Supervisor: Professor Rajkumar Buyya
Associate Supervisor: Dr. Rodrigo N. Calheiros

A Thesis submitted for the Degree of

Doctor of Philosophy

at Monash University in 2016

Faculty of Information Technology

Copyright Notice

Abstract

Cloud Computing has recently emerged as a highly successful alternative information technology paradigm through on-demand resource provisioning and almost perfect reliability. In order to meet the customer demands, Cloud providers are deploying large-scale virtualized data centers consisting of thousands of servers across the world. These data centers require huge amount of electrical energy that incur very high operating cost and as a result, leave large carbon footprints. The reason behind the extremely high energy consumption is not just the amount of computing resources used, but also lies in inefficient use of these resources. Furthermore, with the recent proliferation of communication-intensive applications, network resource demands are becoming one of the key areas of performance bottleneck. As a consequence, efficient utilization of data center resources and minimization of energy consumption are emerging as critical factors for the success of Cloud Computing. This thesis addresses the above mentioned resource and energy related issues by tackling through data center-level resource management, in particular, by efficient Virtual Machine (VM) placement and consolidation strategies. The problem of high resource wastage and energy consumption is dealt with an online consolidated VM cluster placement scheme, utilizing the Ant Colony Optimization (ACO) metaheuristic and a vector algebra-based multi-dimensional resource utilization model. In addition, optimization of network resource utilization is addressed by an online network-aware VM cluster placement strategy in order to localize data traffic among communicating VMs and reduce traffic load in data center interconnects that, in turn, reduces communication overhead in the upper layer network switches. Besides the online placement schemes that optimize the VM placement during the initial VM deployment phase, an offline decentralized dynamic VM consolidation framework and an associated algorithm leveraging VM live migration technique are presented to further optimize the run-time resource usage and energy consumption, along with migration overhead minimization. Such migration-aware dynamic VM consolidation strategy uses realistic VM migration parameters to estimate impacts of necessary VM migrations on data center and hosted applications. Simulation-based performance evaluation using representative workloads demonstrates that the proposed VM placement and consolidation strategies are capable of outperforming the state-of-the-art techniques, in the context of large data centers, by reducing energy consumption up to 29%, server resource wastage up to 85%, and network load up to 60%.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: 

Print Name: **Md Hasanul Ferdaus**

Date: 23rd October, 2016

Publications during enrolment

Book Chapters

1. **Md Hasanul Ferdaus** and Manzur Murshed. Energy-Aware Virtual Machine Consolidation in IaaS Cloud Computing. *Cloud Computing: Challenges, Limitations and R&D Solutions, Zaigham Mahmood (Editor)*, Pages 179–208, **Springer International Publishing**, 2014.

2. **Md Hasanul Ferdaus**, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. Network-aware Virtual Machine Placement and Migration in Cloud Data Centers. *Emerging Research in Cloud Distributed Computing Systems, Susmit Bagchi (Editor)*, Pages 42–91, **IGI Global**, 2015.

Conference Paper

1. **Md Hasanul Ferdaus**, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. Virtual Machine Consolidation in Cloud Data Centers using ACO Metaheuristic. *Proceedings of the 20th International European Conference on Parallel and Distributed Computing (Euro-Par 2014)*, Pages 306–317, **Springer International Publishing**, 2014. [**CORE Rank: A**]

Journal Papers

1. **Md Hasanul Ferdaus**, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. An Algorithm for Network and Data-aware Placement of Multi-Tier Applications in Cloud Data Centers. *Elsevier Journal of Network and Computer Applications (JNCA)*, 2016. (under review) [**CORE Rank: A**]

2. **Md Hasanul Ferdaus**, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. Multi-objective, Decentralized Dynamic Virtual Machine Consolidation using ACO Metaheuristic in Computing Clouds. *Concurrency and Computation: Practice and Experience (CCPE)*, Wiley and Sons Ltd, 2016. (under review) [**CORE Rank: A**]

Acknowledgements

First of all, I praise Allaah (the Creator and Sustainer of the Worlds), the most gracious, the most merciful, for blessing me with the opportunity, courage, and intellect to undertake this research.

I am profoundly indebted to my supervisors Professor Manzur Murshed, Professor Rajkumar Buyya, and Dr. Rodrigo N. Calheiros for their constant guidance, insightful advice, helpful criticisms, valuable suggestions, and commendable support towards the completion of this thesis. They have given me sufficient freedom to explore research challenges of my choice and guided me when I felt lost. Without their insights, encouragements, and endless patience, this research would not have been completed. I also like to express my sincerest gratitude to Professor Sue McKemmish for her continuous support during my candidature. I deeply acknowledge A/Prof Joarder Kamruzzaman, Dr. Gour Karmakar, Professor Guojun Lu, Dr. Mortuza Ali, Dr. Shyh Wei Teng, and A/Prof Wendy Wright for their keen interests, guidelines, and support towards my research progress.

I am grateful to the Monash University for the financial and logistics support throughout the tenure of my postgraduate research which was indispensable for the successful completion of my degree. I would like to thank all the staffs of Faculty of IT, including the former Gippsland campus (now FedUni) and Clayton Campus, the concern persons in Gippsland Research Office, Monash Graduate Education, and Monash Connect for their support and encouragement. I would also like to thank my associate supervisor Professor Rajkumar Buyya for providing the financial support during my study away period in the CLOUDS Lab, University of Melbourne. My special thanks goes to Ms Cassandra Meagher of FIT Clayton, Ms Linda Butler and Ms Susan McLeish of Gippland Graduate Office, and Ms Freda Webb of Gippsland Student Connect for their sincerest and compassionate care. I would also like to thank Dr. Alex McKnight and Dr. Gillian Fulcher for proofreading some chapters of this thesis. I also express my gratitude to Professor Carlos Westphall and Professor Shrideep Pallickara for examining this thesis and providing constructive feedback. I would like to thank Dr. Amir Vahid and Dr. Anton Beloglazov for their valuable suggestions.

I am endlessly indebted and grateful to my parents and parents-in-law for their sacrifice and heartfelt wishes in completing this work. I deeply acknowledge the unconditional love, persistent encouragement and immeasurable support of my wife Tania Rahman (Munny) and my son Mahdi Ibn Ferdaus, without which this achievement could not be a reality. I am grateful to my lovely wife for her understanding and sacrifice in my critical time. My little family was the key source of inspiration in my foreign life. I would like to show my gratitude to my brother Md Hasanul Banna (Sadi), my sister Rashed Akhter (Luni), sister-in-law Rifah Tamanna Dipty, my only niece Ishrat Zerin Shifa, my Grandparents, my Aunts and Uncles, my cousins, and my brother-in-law and friend Nazmus Sadat Rahman. I would also like to thank specially my friends Dr. Md Aftabuzzaman, Dr. Monir Morshed, Dr. Aynul Kabir, Dr. Ahasan Raja Chowdhury, Dr. Md Kamrul Islam, Dr. Abdullah Omar Arafat, Sheikh Ahasan Ahammad, Dr. Ahammed Youseph, Dr. Mustafa Joarder Kamal, Dr. Muhammad Ibrahim, Dr. Md Mamunur Rahid, Dr. Azad Abul Kalam, and Dr. Md Shamshur Rahmah for their kind support, encouragement, and prayers.

Dedicated to my parents for their immense love and care.
Dedicated to my wife and son for their inspiration and sacrifice.

Acronyms

ACO	Ant Colony Optimization
ACS	Ant Colony System
AE	Application Environment
AMDVMC	ACO-based Migration overhead-aware Dynamic VM Consolidation
AS	Ant System
AVVMP	ACO and Vector algebra-based VM Placement
CP	Constraint Programming
DB	Data Block
DC	Data Center
DF	Distance Factor
FF	First Fit
FFD	First Fit Decreasing
GBMM	Global-best Migration Map
GBS	Global-best-solution
HPC	High Performance Computing
IaaS	Infrastructure-as-a-Service
MCVPP	Multi-objective Consolidated VM Placement Problem
MDBPP	Multi-dimensional Bin Packing Problem
MDVCP	Multi-objective Dynamic VM Consolidation Problem
MDVPP	Multi-dimensional Vector Packing Problem
MM	Migration Map
MMAS	Max-Min Ant System
MO	Migration Overhead
NAPP	Network-aware Application environment Placement Problem
NTPP	Need-to-place Peer
PaaS	Platform-as-a-Service
PCL	Physical Computing Link
PDL	Physical Data Link
PE	Packing Efficiency
PL	Physical Link
PM	Physical Machine
PV	Projection Vector
OF	Objective Function
QAP	Quadratic Assignment Problem

RCV Resource Capacity Vector
RDV Resource Demand Vector
RIV Resource Imbalance Vector
RUV Resource Utilization Vector
SaaS Software-as-a-Service
VCL Virtual Computing Link
VDL Virtual Data Link
VL Virtual Link
VLAN Virtual Local Area Network
VM Virtual Machine
VMM Virtual Machine Monitor
VPC Virtual Private Cloud

Notations

$\Delta\tau$	Pheromone reinforcement
δ	Global pheromone decay parameter
η	Heuristic value
τ	Pheromone matrix
τ_0	Initial pheromone amount
AN	AE node (either a VM or a DB)
ANS	Set of ANs in an AE
$BA(CN_p, CN_q)$	Available bandwidth between CN_p and CN_q
$BA(CN_p, SN_r)$	Available bandwidth between CN_p and SN_r
$BA(p_1, p_2)$	Available bandwidth between PMs p_1 and p_2
$BW(VM_i, DB_k)$	Bandwidth demand between VM_i and DB_k
$BW(VM_i, VM_j)$	Bandwidth demand between VM_i and VM_j
CN	Computing Node
CNS	Set of CNs in a DC
$cnList$	Ordered list of CNs in a DC
C_p	Resource Capacity Vector (RCV) of PM_p
d	Number of resource types available in PM
DB	Data Block
DBS	Set of DBs in an AE
D_i	Resource Demand Vector (RDV) of VM_i
$DN(AN)$	DC node where AN is placed
$DS(CN_p, CN_q)$	Network distance between CN_p and CN_q
$DS(CN_p, SN_r)$	Network distance between CN_p and SN_r
$DS(p_1, p_2)$	Network distance between PMs p_1 and p_2
DT	Total duration during which VM is turned down during a migration
f_1	MCVPP Objective Function
f_2	NAPP Objective Function
f_3	MDVCP Objective Function
HV_p	Set of VMs hosted by PM p
MD	Amount of VM memory (data) transferred during a migration
MEC	Energy consumption due to VM migration
MM	Migration map given by a VM consolidation decision
$MO(v, p)$	Migration overhead incurred due to transferring VM v to PM p
MSV	SLA violation due to VM migration
MT	Total time needed for carrying out a VM migration operation

N_c	Total number of CNs in a DC
NC	Network cost that will be incurred for a migration operation
N_d	Total number of DBs in an AE
N_p	Total number of PMs in a data center
N_{pc}	Number of PMs in a PM cluster
N_r	Number of resource types available in PM
N_s	Total number of SNs in a DC
N_v	Total number of VMs in a cluster or AE or data center
N_{vc}	Number of VCLs in an AE (Chapter 5) or VMs in a PM cluster (Chapter 6)
N_{vd}	Total number of VDLs in an AE
N_{vn}	Average number of NTPP VLs of a VM or a DB
$OG(v, p)$	Overall gain of assigning VM v to PM p
p	Individual Physical Machine
P	Set of PMs in a PM cluster
PCL	Physical Computing Link that connects two CNs
PDL	Physical Data Link that connects a CN and a SN
PL	Physical Link
PM	Physical Machine
PM_p	An individual PM in set PMS
PMS	Set of PMs in a data center
$pmList$	Ordered list of PMs in a data center
r	Single computing resource in PM
RC	Single computing resource in PM
RC_l	An individual resource in set RCS
RCS	Set of computing resources available in PMs
SN	Storage Node
SNS	Set of SNs in a DC
$snList$	Ordered list of SNs in a DC
$UG_p(v)$	Utilization gain of PM p after VM v is assigned in it
U_p	Resource Utilization Vector (RUV) of PM_p
V	Set of active VMs in a PM cluster
v	Individual Virtual Machine
VCL	Virtual Computing Link that connects two VMs
$vclList$	Ordered list of VCLs in an AE
v^{cpu}	CPU demand of a VM v
VDL	Virtual Data Link that connects a VM and a DB
$vdlList$	Ordered list of VDLs in an AE
v^{dr}	Page Dirty Rate of a VM
v^{hp}	Host PM of a VM
VL	Virtual Link
VM	Virtual Machine
v^{mem}	Memory demand of a VM v
$vmList$	Ordered list of VMs in a cluster
VM_i	An individual VM in set VMS
VMS	Set of VMs in a VM cluster or AE or data center
x	VM-to-PM Placement Matrix
y	PM Allocation Vector

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rapid progress in microminiaturization of technologies and the proliferation of the Internet, computing resources are now more powerful, cheaper, and ubiquitously available than ever before. This technological shift has enabled the realization of a new computing paradigm called Cloud Computing. Technically, Clouds are large pools of easily accessible and readily usable virtualized resources, such as hardware, development platforms, and services that can be dynamically reconfigured to adjust to a variable load in terms of scalability, elasticity, and load balancing, and thus, allow opportunities for optimal resource utilization. This pool of virtualized resources is typically provisioned by Cloud infrastructure providers using a pay-per-use business model with extremely high availability and almost perfect reliability (e.g., 99.997% for Amazon EC2 [75]) by means of Service Level Agreements (SLAs). Cloud consumers can access these resources and services based on their requirements without any regard as to the location of the consumed resources and services.

In order to cope with the rapid growth of customer demands for processing power, storage, and communication, Cloud providers, such as Amazon, Google, and Microsoft are deploying large-scale data centers across the globe. Recent report shows that Cloud giant Amazon operates at least 30 data centers in its global network, each comprising 50,000 to 80,000 servers with a power consumption of between 25 to 30 megawatts [86]. As a consequence, a huge amount of electrical energy is required to run the servers and network devices, as well as to keep the cooling systems operating for these data centers. As per the Data Center Knowledge report [99], power is one of the critical TCO (Total Cost of Ownership) variables in managing data centers, and servers and data equipment

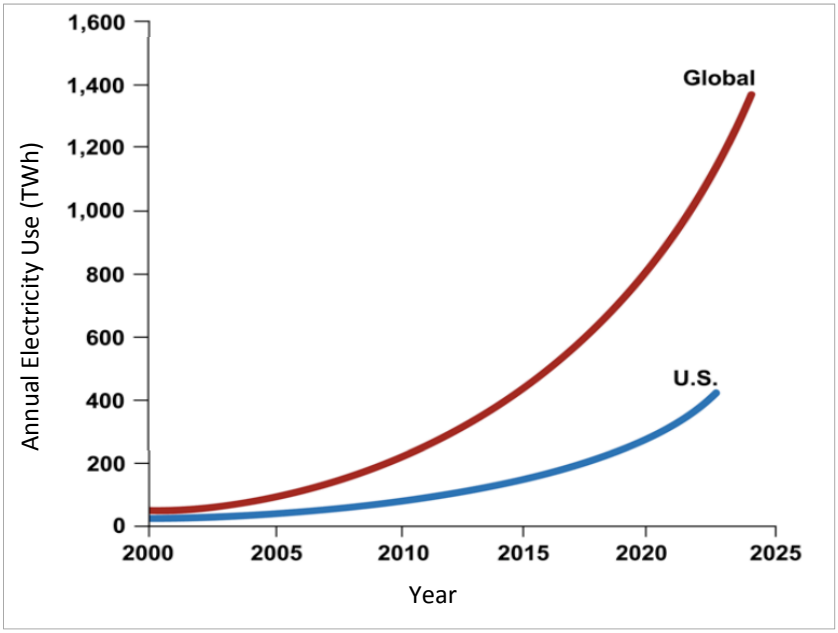

Figure 1.1: Worldwide data center electricity consumption (source: Digital Power Group [87]) (best viewed in color).

are responsible for 55% of energy used by the data center, followed by 30% for the cooling system. In spite of continuous progress in equipment efficiency, statistics of the worldwide data center electricity consumption show non-linear growth throughout the last decade and a similar trend is expected for the upcoming years [87]: a steady rise of 110% from 2010 to 2015 and a predicted rise of 82% from 2015 to 2020 (Figure 1.1).

Large data centers are not only expensive to maintain, but they also have enormous detrimental effects on the environment. Reports claim that the information technology ecosystem alone represents around 10% of the world's electricity consumption [27] and data centers, the main driving element of this ecosystem, are responsible for around 2% of global Greenhouse Gas (GHG) emissions, a share comparable to the aviation industry [119].

This extremely high energy consumption is not just because of the amount of computing resources used and the power inefficiency of the hardware infrastructures, but also due to the inefficient use of these resources. A recent study presented by Reiss *et al.* [102] shows that a 12,000-nodes Google cluster achieves aggregate CPU utilization only of 25%-35% and memory utilization of 40%. A similar underutilization trend has been identified by the researchers from Stanford University showing that a thousand-nodes production cluster at Twitter runs consistently at CPU utilization below 20% and memory usage at around 40%-50%, whereas the overall utilization estimates are even poorer (between 6%

and 12%) for Cloud facilities that do not consider workload co-location [31]. Moreover, the narrow dynamic power range of physical servers further exacerbates the problem— even completely idle servers consume about 70% of their peak power usage [42]. Such low resource utilization, technically termed *Server Sprawl*, contributes to both capital expenses and operational costs due to non-proportional energy consumption. As a consequence, underutilization of data center resources is a major challenge for the ultimate success of Cloud Computing.

Furthermore, rapid development and expansion of Cloud technologies have resulted in data centers experiencing sharp rise in network traffic, a major portion of which is constituted by the data communications within the data center. A recent report [25] published by Cisco Systems Inc. demonstrates that the Cloud data centers will dominate the global data center network traffic flow for the foreseeable future and its importance is highlighted by one of the top-line projections from this forecast, which predicts that more than four-fifths of the total data center traffic will be Cloud traffic by 2019 (Figure 1.2). One important trait on data center traffic, pointed out in this report, is that the majority of global data center traffic is generated due to the data communications within the data centers: in 2014, it was 75.4% and it will be around 73.1% in 2019. This huge amount of intra-center traffic is primarily generated by application components that are interrelated to each other in terms of communication: for example, the computing components of a multi-tier application writing data to the storage array after processing the data. This large growth in data center traffic poses a serious scalability concern for the wide adoption of Cloud Computing, particularly with the recent escalation in the size of data being processed and transmitted within and outside the data center.

Given that Cloud Computing is relatively a recent information technology model with rapidly increasing popularity, infrastructures and services offered by Cloud providers are being expanded with a similar pace. As a consequence, from a market-oriented point of view, design directions and development endeavors are primarily focused on the functionality, scalability, and reliability aspects of Cloud resources and services. Having said that, optimization of resource utilization and energy consumption in Cloud infrastructures is going to be the upcoming high priority area of advancement. Moreover, due to the proprietary nature of large-scale Cloud infrastructures, developments and innovations achieved by the corporate Cloud providers are hardly available in public domain.

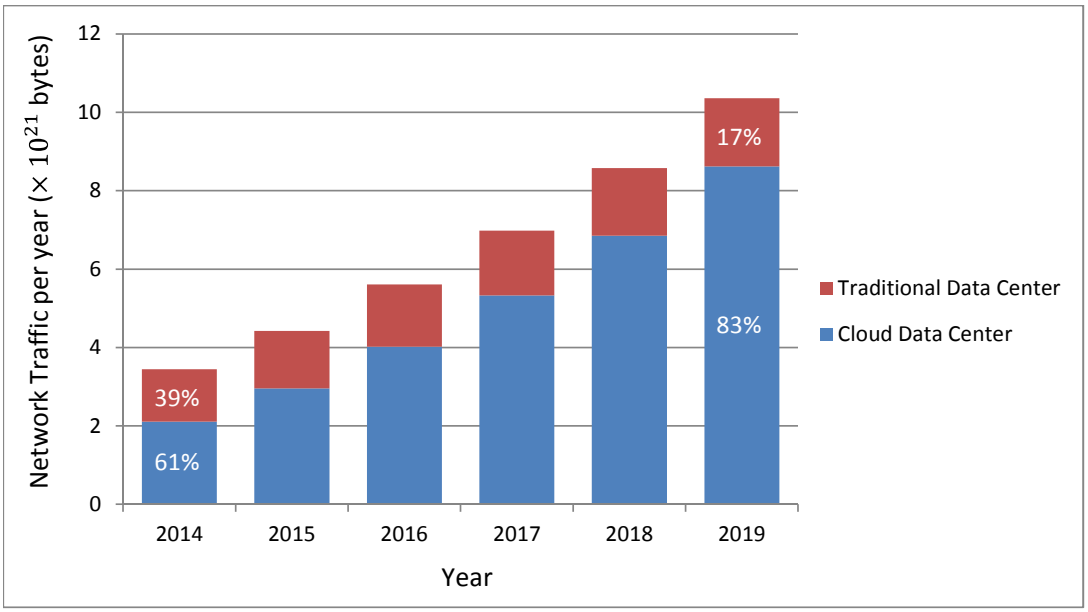

Figure 1.2: Worldwide data center traffic growth (source: Cisco Systems Inc. [25]) (best viewed in color).

In light of the above discussion, this thesis addresses the problem of the inefficient use of computing, storage, and network resources in large-scale, virtualization data centers, such as Clouds, through *Virtual Machine* (VM) management, with the goal of minimizing data center power consumption, resource wastage, and network cost, as well as the associated management overhead.

1.1 Motivations

Among all the Cloud service models, the key for the extreme success of Cloud Computing is the *Infrastructure-as-a-Service* (IaaS), which enables Cloud providers to provision the computing infrastructures needed to deliver the services simply by renting resources as long as needed, without ever buying a single component. Cloud infrastructures depend on one or more data centers, either centralized or distributed, and on the use of various cutting-edge resource virtualization technologies that enable the same physical resources (computing, storage, and network) to be shared among multiple application environments [133]. *Server Virtualization* is one of the essential technologies that have enabled Clouds to be highly flexible and dynamically reconfigurable environments where physical resources are provisioned and reclaimed through the creation, resizing, migration, and termination of VMs. Virtualization technologies allow data centers to address

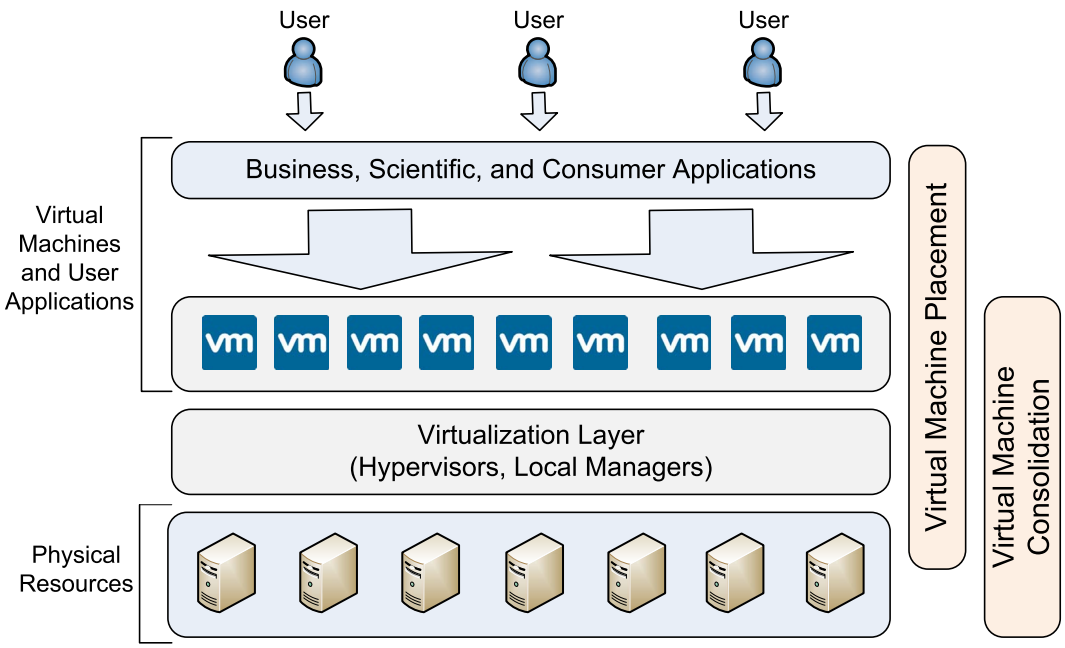

Figure 1.3: Virtual Machine management through placement and consolidation strategies.

resource and energy inefficiencies by (i) provisioning multiple VMs in a single physical server, where each VM represents a run-time environment completely isolated from one another, and (ii) live migrations of VMs [26] from current hosts to other servers; and by this process, providing opportunities to improve resource utilization. In particular, efficient VM placement and consolidation decisions during the VM life cycle offer potential for the optimization of data center resources and power consumption (Figure 1.3).

Underutilization of server resources can be optimized through efficient online VM placement during the initial VM deployment phase. Due to the wide spectrum of heterogeneous applications deployed in the Clouds, such as e-commerce, business workflows, and social networking, and so on, VM resource demands expand across multiple dimensions and exhibit large variations. Focusing on the diverse resource demands of customer applications, Cloud providers, such as Amazon[1] and Google[2], offer various categories of predefined VMs with different set amounts of resources across multiple dimensions (such as CPU, main memory, and network bandwidth), as well as customized VMs, where users can configure the amount of various resources based on their needs. Such multi-dimensionality of VM resource demands can be effectively utilized to optimize VMs placement in the physical servers that will improve resource utilization for various resource types.

[1]Amazon EC2 Instance Types, 2016. `https://aws.amazon.com/ec2/instance-types/`
[2]Google Cloud Platform— Compute Engine, 2016. `https://cloud.google.com/compute/`

Moreover, an effective approach for improving overall server resource utilization is through consolidated VM placement, where a group of requested VMs are efficiently packed so that a minimal number of servers are used for hosting the VMs. This process helps to improve center-wide resource utilization of the active servers which, in turn, facilitates energy efficiency as unused servers can be kept in lower power states, such as turned off [123], and tries to avoid running servers underutilized and reduce non-proportional power consumption. This approach has a further advantage that unused servers can be kept reserved for hosting any future workloads and, therefore, it improves the overall throughput of the data center in terms of workloads.

Online VM placement also helps in addressing network scalability issues arising in the context of large-scale Cloud data centers. Considering the recent trend in modern Cloud applications, it is observed that a large portion of these applications are multi-component-based, such as multi-tier enterprise applications and scientific workflows [68]. As a consequence, a significant portion of the VMs deployed in the Clouds are part of multi-component applications, where the inter-component communication correlations are known during the initial application deployment phase. Online placement decisions of such VMs, with consideration of the communication dependencies, provide opportunities for optimization of network resources; most importantly, link bandwidth. As already mentioned, around three-fourths of the global data center traffic is generated due to the inter-component communications of the applications running within data centers [25], network-aware VM placements can be highly effective in managing this huge network traffic. Moreover, traditional research on network communication and bandwidth optimization have so far focused on the rich connectivity at the edges of the network and dynamic routing protocols to balance the traffic load [85]. Complementary to these approaches, network-aware VM placement policies can be developed in parallel to help addressing network scalability issues from the perspective of data center management.

Furthermore, with the increasing trend in applications being ever more data-dependent, progressively more Cloud applications are deployed with associated data components [58]. Computational components of such applications are coupled with their associated data components in terms of communication dependencies. As a consequence, consideration of such associated data components, along with their correlations with the VMs, has potential to further improve VM placement for network resource optimization.

While online VM placement and allocations strategies can optimize placement decisions during the initial VM deployment phases, active VMs can exhibit variations on run-time resource usage during VM life cycle due to workload variations [89]. Moreover, due to the Cloud features of on-demand resource provisioning and a pay-as-you-go business model, VMs are created and terminated in the data centers dynamically based on customer requests. As a consequence, physical server resources become fragmented in time, which eventually reduces server resource utilization, as well as the overall hosting capacity of the data center. Such underutilization of computing resources at run-time is one of the primary reasons for very high resource wastage in production data centers [30]. In addition, underutilized physical servers contribute to energy wastage due to the narrow dynamic power range: even completely idle servers consume about 70% of their peak power usage [42]. Both the problems of run-time server resource wastage and non-proportional power consumption can be addressed by improving server resource utilization by means of offline, dynamic VM consolidation operation.

Complementary to the online VM placement schemes where VM deployment requests are served as soon as possible with a best-effort policy, offline, dynamic VM consolidation could be run according to a periodic or trigger-driven policy, and thus can be accommodated simultaneously with the online placement strategies. Such dynamic VM consolidation can be achieved by utilizing the VM live migration technique [26], where the running VMs are rearranged and packed into a reduced number of active servers within the data center, while respecting VM resource requirements and server resource capacity constraints. Servers that are released by this process can be switched to lower power states (e.g., standby) in order to save energy.

1.2 Research Problems and Objectives

This thesis deals with the research challenges associated with multi-objective VM placement and consolidation in the context of large-scale virtualized data centers. Given that VMs and servers are characterized by multi-dimensional resources, such placement and consolidation problems effectively fall in the category of $\mathcal{NP}-hard$ combinatorial optimization problems. In particular, the following research problems are addressed in this thesis:

(i) VMs that are requested for placement are characterized by their resource demands across multiple resource dimensions, such as CPU, memory, and network bandwidth. In a dynamic environment, such as the Clouds, VMs are usually rented by various Cloud customers and potentially host different types of applications [15]. Such VM resource demands demonstrate balanced, as well as complementary, amounts of resources across multiple dimensions. This multi-dimensionality of resource types increases the complexity of the VM placement problem since exhausting one type of server resource, such as memory, will eventually make the server incapable of hosting any more VMs, even though other types of resources, such as CPU and network I/O, remain underutilized [88]. Therefore, both the balanced and the complementary resource patterns need to be exploited in order to make efficient VM placements, so that server resources can be utilized in a balanced way across different resource dimensions, that will eventually improve the overall server resource utilization and reduce resource wastage. This raises the problem of how to capture server utilization effectively during VM placement so that the technique would be uniform and could be readily integrated to the VM placement and consolidation strategies.

(ii) When VMs submission requests arrive, the online VM placement subsystem needs to make the decision as to which server to select for allocating each of the VMs, with the aim of improving the overall resource utilization of the active servers, so that both resource wastage and power consumption are minimized. Effectively, this requires the subsystem to determine placement decisions for the VMs so that the number of servers needed for hosting the VMs is minimized. Moreover, the complexity of the problem is further increased by the size of today's large-scale infrastructures, in particular Cloud data centers, that are comprised of thousands of servers, where a large number of VM deployment requests are issued at run-time. In fact, this particular VM placement problem is an instance of *Multi-dimensional Vector Packing Problem* [23] for which there is no known polynomial-bound exact solution [17], which consequently makes designing scalable solutions even harder.

(iii) Placement decisions of VMs with mutual communication correlations need to ensure feasible VM placement both in terms of computational resources and network bandwidth. Taking into account the diversity and structural complexity of composite Cloud applications [57], it is crucial to represent such composite applications in

a generic and appropriate manner. With a focus on improving network scalability, the immediate problem is how to model the network overheads incurred due to the placement of such composite applications.

(iv) Online placement of VMs, that are submitted as a part of a composite application having associated data components, focusing on network resource optimization needs to figure out appropriate servers for the VMs, as well as appropriate storage nodes for the data components. This necessitates the design of efficient algorithms such that it minimizes the network overheads due to the placements, with the ultimate goal of improving network scalability. However, optimal placement of application components that results in minimum network overhead is non-trivial, especially in the context of large-scale data centers, since it requires finding the best placement among all possible combinations of feasible placements. This makes the problem an extended version of the *Quadratic Assignment Problem* (QAP) [80], which is a combinatorial optimization problem that is already shown to be computationally $\mathcal{NP}-hard$ [18].

(v) Run-time optimization of resource utilization through dynamic VM consolidation requires VM live migration operations that have adverse effects on hosted applications and data center components [122]. The problem is how to estimate the overheads of the VM live migrations, which will be necessary to achieve a certain consolidation state. Moreover, active VMs differ in their run-time properties, such as memory size and page dirty rate, that contribute to the impact of VM migrations. As a consequence, different VM migrations have different amounts of migration overheads and oversimplified measures, such as the number of migrations, are inappropriate to use in consolidation decisions.

(vi) Dynamic VM consolidation with the application of VM migration is effectively a multi-objective problem with potentially conflicting goals of maximizing the number of released servers and minimizing the associated migration overheads. Therefore, the problem is to identify which VMs to migrate, along with the corresponding target servers, that will help in releasing the largest number of servers with minimal migration overheads, so that both server resource wastage and power consumption

can be reduced with a nominal impact on hosted applications and data center components. From algorithmic point of view, this problem requires the solution to come up with the VM migration decisions among all possible migrations within the data center that would optimize the defined objectives. Moreover, with the increase in the number of VMs and servers, the search space of the problem expands exponentially. As a consequence, this particular VM consolidation problem secures its place in the spectrum of discrete combinatorial optimization problems.

(vii) Migration-based consolidation operations can suffer serious scalability issues for large-scale data centers. Since VM live migration operations involve non-negligible amounts of memory data transfer across the communication network, data center-wide VM consolidation decisions can lead to a large amount of data transfer through the upper layer switches and, therefore, have the potential to impose large network overheads. As a consequence, it is necessary to design a scalable, dynamic VM consolidation scheme.

The overall goal of this research project is to devise VM management strategies and algorithms with a focus on optimizing resource utilization and power consumption in the context of large-scale virtualized data centers. To this end, both online and offline VM management scenarios are considered in the thesis. In particular, the following specific research objectives are delineated in order to deal with the challenges associated with the above research problems:

- Explore, analyze, and categorize the research in the field of VM management, primarily VM placement, migration, and consolidation, with the goal of achieving systematic knowledge and understanding of the existing techniques and approaches.

- Develop an effective and unified technique to capture balanced utilization of multi-dimensional server resources during the course of VM placement such that it can be readily integrated into VM placement and consolidation schemes.

- Design an online, multi-objective VM placement scheme and associated algorithm to generate VM placement plans that require a minimal number of physical servers and, by this process, improve server resource utilization and reduce power consumption. Moreover, the decision time required by the algorithm needs to be realistic for online scenarios, even for reasonably large-scale problems.

- Formulate appropriate models to represent a generic structure of composite application with VMs and data components, with mutual communication correlations, and define relevant cost functions to capture network overheads incurred by VM placements.

- Design an online, network-aware VM placement strategy and associated algorithm for composite application placement with the goal of reducing the network overheads incurred due to the overall application placement by localizing network traffic among VMs and data components.

- Develop an effective and uniform VM live migration overhead estimation technique, taking into account realistic migration parameters and overhead factors, so that the estimate of the migration overheads can be integrated with the dynamic VM consolidation schemes.

- Design a decentralized, dynamic VM consolidation framework in order to form multiple server clusters within a data center with the aim of improving the scalability of dynamic VM consolidation operations and reducing network overheads due to the associated VM migrations.

- Design an offline, multi-objective VM consolidation technique and associated algorithm in order to generate VM migration plans that consolidate the active VMs into a reduced number of server with the potentially conflicting goals of maximizing the number of released servers with minimal migration overheads.

1.3 Contributions

The key contributions of this thesis are summarized below.

1. A taxonomy and survey of state-of-the-art VM management strategies in the context of large-scale virtualized data centers, including identification of various nomenclatural aspects of VM placement, migration, and consolidation, as well as classification and analysis of the prominent techniques and strategies (**Chapter 3**).

2. A mathematical framework for modeling the multi-objective VM placement problem as an instance of the general *Multi-dimensional Vector Packing Problem* [23] that facilitates problem analysis and guides the designing of a solution approach (**Chapter 4**).

3. A server resource utilization capture technique based on vector algebra that effectively measures the mean across multiple resource dimensions as a unified scalar quantity such that it represents balanced utilization. This utilization capture technique is generic so that it can exploit the complementary resource demand patterns among heterogeneous VMs and be readily integrated to any online or offline VM management strategies (**Chapter 4**).

4. A novel online, multi-objective VM placement algorithm, integrated with the server utilization capture technique, with the goal of reducing power consumption and server resource wastage. The algorithm is application-agnostic and adapts the *Ant Colony Optimization* (ACO) metaheuristic in order to navigate the search space efficiently within polynomial time that makes it suitable for online or real-time VM placement scenarios in the context of large-scale Cloud infrastructures (**Chapter 4**).

5. Models for representing generic, multi-component Cloud applications and placement-related network costs, and a mathematical framework that defines the network-aware, online application placement as a combinatorial optimization problem with the objective of network cost minimization (**Chapter 5**).

6. Novel heuristics for network-efficient, simultaneous placement of VMs and data blocks comprising multi-component applications with a focus on reducing the network overheads on the data center network and, ultimately, improving scalability. The proposed placement scheme strives to reduce the distance that data packets need to travel within

the communication substrate and thereby, help in localizing network traffic and minimizing communication overhead in upper-layer network switches. Moreover, the proposed algorithms compute the placements pretty quickly (in the fractions of a second) that makes it suitable for medium-to-large-scale infrastructures (**Chapter 5**).

7. A mathematical framework with associated models for defining the migration impact-aware, multi-objective dynamic VM consolidation problem as an NP-hard discrete combinatorial optimization problem with potentially conflicting objectives of minimizing data center resource wastage, power consumption, and overall migration overheads due to VM consolidation (**Chapter 6**).

8. VM live migration overhead estimation models, in the context of the pre-copy VM live migration technique, which takes into account realistic migration parameters and overhead factors such that the models are not restricted to any specific VM consolidation method and can be readily integrated to any migration-based VM management strategy (**Chapter 6**).

9. A hierarchical, decentralized VM consolidation framework for clustering servers in a data center based on mutual network costs of data communications where VM consolidation can be performed within each group separately. This decentralized consolidation approach assists in localizing migration-related network traffic and reducing network costs due to VM migrations, thus ultimately facilitating the minimization of migration overheads and improving the scalability of dynamic VM consolidation operations (**Chapter 6**).

10. A novel offline, multi-objective dynamic VM consolidation algorithm, coupled with the migration overhead estimation models, designed through appropriate adaption of the ACO metaheuristic. The consolidation algorithm consolidates the active VMs in a reduced number of servers with the goal of optimizing the run-time resource utilization and power consumption within the data center, while at the same time minimizing the associated VM migration overheads incurred due to the consolidation. Due to the polynomial nature of ACO-based metaheuristics, the proposed algorithm generates VM consolidation decisions reasonably fast that makes it suitable for taking offline optimization decisions (**Chapter 6**).

11. Extensive simulation-based performance evaluation of the proposed VM placement and consolidation techniques, which are compared with the state-of-the-art algorithms across multiple performance metrics and several scaling factors, along with comprehensive analysis and discussion of the observed results (**Chapters 4-6**).

1.4 Thesis Organization

The chapters and their contributions are set out in Figure 1.4. The remainder of the thesis is organized as follows:

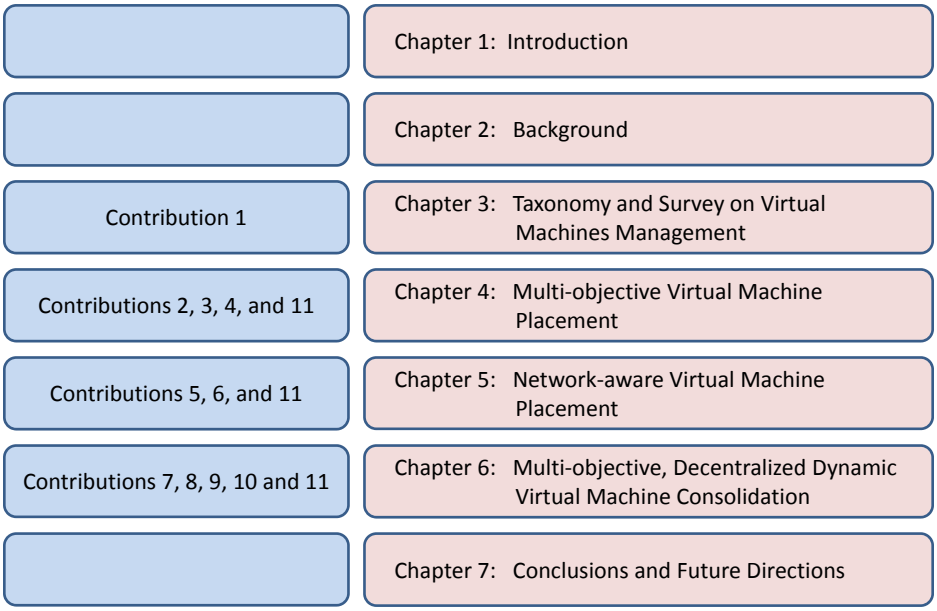

Figure 1.4: Thesis chapters outline.

- Chapter 2 presents an overview of the various concepts, elements, systems, and technologies relating to the research area of this thesis.

- Chapter 3 presents a taxonomy and survey of the VM placement, migration, and consolidation strategies in the context of virtualized data centers, particularly of Cloud data centers. Preliminary results from this chapter have been published in two book chapters [47] and [49] published by Springer International Publishing and IGI Global, respectively.

- Chapter 4 proposes a multi-objective, on-demand VM cluster placement strategy for virtualized data center environments with a focus on energy-efficiency and resource utilization. Preliminary results from this chapter have been published in [48].

- Chapter 5 presents a network-aware, on-demand VM cluster placement scheme along with associated data components in the context of modern Cloud-ready data centers with the objective of reducing network traffic overheads. Journal paper written from preliminary results of this chapter is under second review.

- Chapter 6 proposes an offline, decentralized, dynamic VM consolidation framework and an associated VM consolidation algorithm leveraging the VM live migration technique with the goal of optimizing the run-time resource usage, energy consumption, and associated VM live migration overheads.

- Chapter 7 concludes the thesis with a summary of the contributions, main findings, and discussion of future research directions, followed by final remarks.

Chapter 2

Background

Cloud Computing have been very successful from its very inception and the reason behind its extreme success is the utilization of various technological elements, ranging from physical servers to virtualization technologies and application development platforms. As this thesis focuses on data center-level resource management and energy consumption leveraging Virtual Machine (VM) management strategies, this chapter presents an overview of the various Cloud Computing features and properties from infrastructure point of view, including background on virtualization and data center architectures.

2.1 Introduction

Cloud Computing has been growing with a rapid pace from its very inception. The main reason behind its continuous and steady improvement is the unique features of very high reliability, elasticity, and on-demand resource provisioning. In order to provide these features, Cloud providers are provisioning large-scale infrastructures, leveraging various technological elements, ranging from physical servers to virtualization technologies and application development platforms. Since this thesis addresses the issues of the data center-level resource utilization and energy-efficiency through Virtual Machine (VM) management, an overview of various Cloud Computing features and properties from infrastructure point of view, including background on the virtualization technology and data center architectures will facilitate an informed and smooth reading of the remaining chapters. With this motivation, this chapter presents a brief background on the relevant topics relating to VM management in the context of Cloud data centers.

The rest of this chapter is organized as follows. Section 2.2 presents an background on Cloud Computing from perspectives of the architecture, the deployment models, and the provided services. Various features and categories of the virtualization technologies are discussed in Section 2.3, followed by a description on the VM migration techniques in Section 2.4. Section 2.5 presents a brief overview of the different data center network architectures, followed by a description of the Cloud application workloads and network traffic patterns in Section 2.6. A brief overview of the VM consolidation techniques, along with their pros and cons, are presented in Section 2.7. Finally, Section 2.8 summarizes the chapter.

2.2 Cloud Infrastructure Management Systems

While the number and scale of Cloud Computing services and systems are continuing to grow rapidly, significant amount of research is being conducted both in academia and industry to determine the directions to the goal of making the future Cloud Computing platforms and services successful. Since most of the major Cloud Computing offerings and platforms are proprietary or depend on software that is not accessible or amenable to experimentation or instrumentation, researchers interested in pursuing Cloud Computing infrastructure questions, as well as future Cloud service providers, have very few tools to work with [96]. Moreover, data security and privacy issues have created concerns for enterprises and individuals to adopt public Cloud services [6]. As a result, several attempts and ventures of building open-source Cloud management systems came out of collaborations between academia and industry, including OpenStack[1], Eucalyptus [96], OpenNebula [110], and Nimbus[2]. These Cloud solutions provide various aspects of Cloud infrastructure management, such as:

1. Management services for *Virtual Machine* (VM) life cycle, compute resources, networking, and scalability.

2. Distributed and consistent data storage with built-in redundancy, fail-safe mechanisms, and scalability.

3. Discovery, registration, and delivery services for virtual disk images with support of different image formats (e.g., VDI, VHD, qcow2, VMDK, etc.)

[1]OpenStack Open Source Cloud Computing Software, 2016. `https://www.openstack.org/`
[2]Nimbus is cloud computing for science, 2016. http://www.nimbusproject.org/

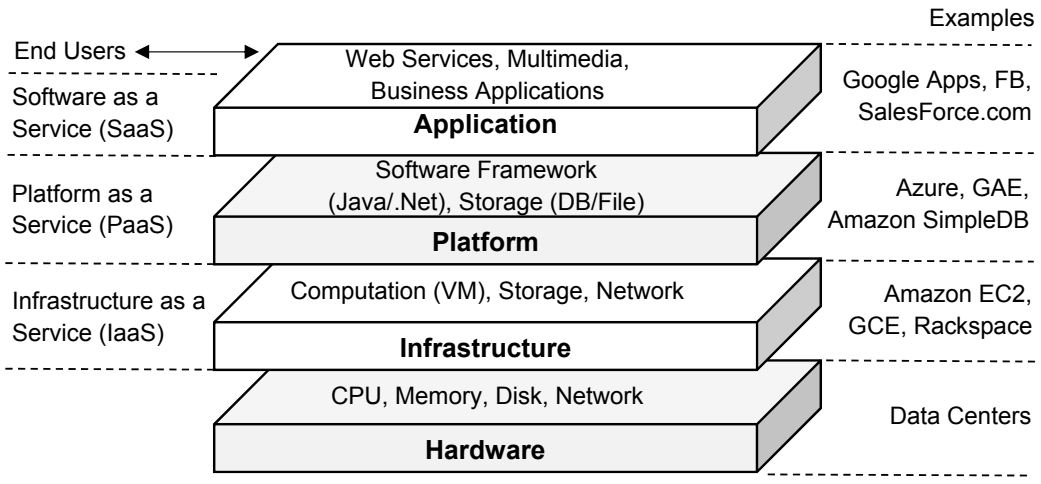

Figure 2.1: The Cloud Computing Architecture.

4. User authentication and authorization services for all components of Cloud management.

5. Web and console-based user interface for managing instances, images, cryptographic keys, volume attachment/detachment to instances, and similar functions.

Figure 2.1 shows the four essential layers of the Cloud Computing environment from the architectural perspective. Each layer is built on top of the lower layers and provides unique services to its upper layers.

1. ***Hardware Layer:*** This layer is composed of the physical resources of typical data centers, such as physical servers, storage devices, load balancers, routers, switches, communication links, power systems, and cooling systems. This layer is essentially the driving element of Cloud services and, as a consequence, operation and management of the physical layer incur continuous costs for the Cloud providers. Example includes the numerous data centers of Cloud providers that spread all over the globe, such as Amazon AWS[3], Rackspace[4], Google[5], Microsoft[6], and Linode[7].

2. ***Infrastructure Layer:*** This layer (also known as *Virtualization Layer*) creates a pool of on-demand computing and storage resources by partitioning the physical resources, utilizing the various virtualization technologies, such as Xen [9] and

[3]Amazon Web Services, 2016. `http://aws.amazon.com/`
[4]Rackspace: The Managed Cloud Company, 2016. `http://www.rackspace.com/`
[5]Google Cloud Platform, 2016. `https://cloud.google.com/compute/`
[6]Microsoft Azure, 2016. `https://azure.microsoft.com`
[7]Linode: SSD Cloud Hosting, 2016. `https://www.linode.com/`

VMware[8]. Efficient allocation and utilization of the virtual resources, in accordance with the computing demands of Cloud users, are important to minimize the Service Level Agreement (SLA) violations and maximize revenues.

3. ***Platform Layer:*** Built on top of the infrastructure layer, the platform layer consists of customized operating systems and application frameworks, that help automate the process of of application development, deployment, and management. In this way, this layer strives to minimize the burden of deploying applications directly on the VM containers.

4. ***Application Layer:*** This layer consists of the actual Cloud applications, which are different from traditional applications and can leverage the on-demand automatic-scaling feature of Cloud Computing to achieve better performance, higher availability, and reliability, as well as the minimization of operating costs.

In alignment with the architectural layers of Cloud infrastructure resources and services, the following three primary service models have evolved and are used extensively by the Cloud community [118]:

1. ***Infrastructure-as-a-Service (IaaS)****:* Cloud provides provision computing resources (e.g., CPU and memory) to Cloud customers in the form of VMs, storage resources in the form of storage blocks, file systems, databases, and so on, as well as communication resource in the form of bandwidth. IaaS provides further provide management consoles and dashboards, APIs (Application Programming Interfaces), and advanced security features for manual and autonomic control and management of the virtual resources. Typical examples are Amazon EC2[9], Google Compute Engine[10], and Rackspace Server Hosting[11].

2. ***Platform-as-a-Service (PaaS):*** PaaS providers offer a development platform, such as programming environments, tools, that allows Cloud consumers to develop Cloud services and applications, as well as deployment platforms that host those

[8] VMware Virtualization, 2016. `http://www.vmware.com/virtualization`
[9] Amazon Elastic Compute Cloud, 2016. `http://aws.amazon.com/ec2`
[10] Google Cloud Platform, 2016. `https://cloud.google.com/compute/`
[11] Rackspace Server Hosting, 2016. `http://www.rackspace.com.au/cloud/servers`

services and applications, and thus support full software life-cycle management. Examples include Google App Engine[12] and Microsoft Azure[13] platforms.

3. ***Software-as-a-Service (SaaS):*** Cloud consumers release their applications on a hosting environment fully managed and controlled by the SaaS Cloud providers and the applications can be accessed through the Internet from various clients, such as Web Browsers, Mobile Apps, and Terminal Emulators. Examples of the SaaS platform include Google Apps[14], Salesforce.com[15], and Twitter[16].

2.3 Virtualization Technologies

One of the main enabling technologies that paved the way of Cloud Computing towards its extreme success is virtualization. Clouds leverage various virtualization technologies (e.g., machine, network, and storage) to provide users an abstraction layer that provides a uniform and seamless computing platform by hiding the underlying hardware heterogeneity, geographic boundaries, and internal management complexities [133]. It is a promising technique by which resources of physical servers can be abstracted and shared through partial or full machine simulation by time-sharing and hardware and software partitioning into multiple execution environments each of which runs as complete and isolated system. It allows dynamic sharing and reconfiguration of physical resources in Cloud Computing infrastructure that makes it possible to run multiple applications in separate VMs having different performance metrics. It is virtualization that makes it possible for the Cloud providers to improve utilization of physical servers through VM multiplexing [84] and multi-tenancy (i.e., simultaneous sharing of physical resources of the same server by multiple Cloud customers). It also enables on-demand resource pooling through which computing resources (like CPU and memory), network, and storage resources are provisioned to customers only when needed [73]. This feature helps avoid static resource allocation based on peak resource demand characteristics. In short, virtualization enables higher resource utilization, dynamic resource sharing, and better energy management, as well as improves scalability, availability, and reliability of Cloud resources and services [20].

[12]Google App Engine, 2016. `https://cloud.google.com/appengine/`
[13]Microsoft Azure: Cloud Computing Platform & Services, 2016. `http://azure.microsoft.com/`
[14]Google Apps for Work, 2016. `https://apps.google.com/`
[15]Salesforce: CRM and Cloud Computing, 2016. `http://www.salesforce.com/`
[16]Twitter, 2016. `https://twitter.com/`

From the architectural perspective, the virtualization approaches are primarily categorized into the following two types:

1. ***Hosted Architecture:*** The virtualization layer is installed and run as an individual application on top of an operating system and supports the broadest range of underlying hardware configurations. Example of such architecture includes VMware Workstation[17] and Player[18], and Oracle VirtualBox[19].

2. ***Hypervisor-based Architecture:*** The virtualization layer, termed *Hypervisor* is installed and run on bare hardware and retains full control of the underlying physical system. It is a piece of software that hosts and manages the VMs on its *Virtual Machine Monitor* (VMM) components (Figure 2.2). The VMM implements the VM hardware abstraction, and partitions and shares the CPU, memory, and I/O devices to successfully virtualize the underlying physical system. In this process, the Hypervisor multiplexes the hardware resources among the various running VMs in time and space sharing manner, the way traditional operating system multiplexes hardware resources among the various processes [108]. VMware ESXi[20] and Xen Server [9] are examples of this kind of virtualization. Since hypervisors have direct access to the underlying hardware resources rather than executing instructions via operating systems as it is the case with hosted virtualization, a hypervisor is much more efficient than a hosted virtualization system and provides greater performance, scalability, and robustness.

Among the different processor architectures, the Intel x86 architecture has been established as the most successfully, widely adopted, and highly inspiring. In this architecture, different privilege level instructions are executed and controlled through the four privilege rings: Ring 0, 1, 2, and 3, with 0 being the most privileged (Figure 2.3) in order to manage access to the hardware resources. Regular operating systems targeted to run over bare-metal x86 machines assume full control of the hardware resources and thus are placed in Ring 0 so that they can have direct access to the underlying hardware, while typical user level applications run at ring 0.

[17] VMware Workstation, 2015. `http://www.vmware.com/products/workstation`
[18] VMware Player Pro, 2015. `http://www.vmware.com/products/player`
[19] Oracle VirtualBox, 2015. `https://www.virtualbox.org/`
[20] VMware ESXi Hypervisor, 2015. `http://www.vmware.com/products/esxi-and-esx/`

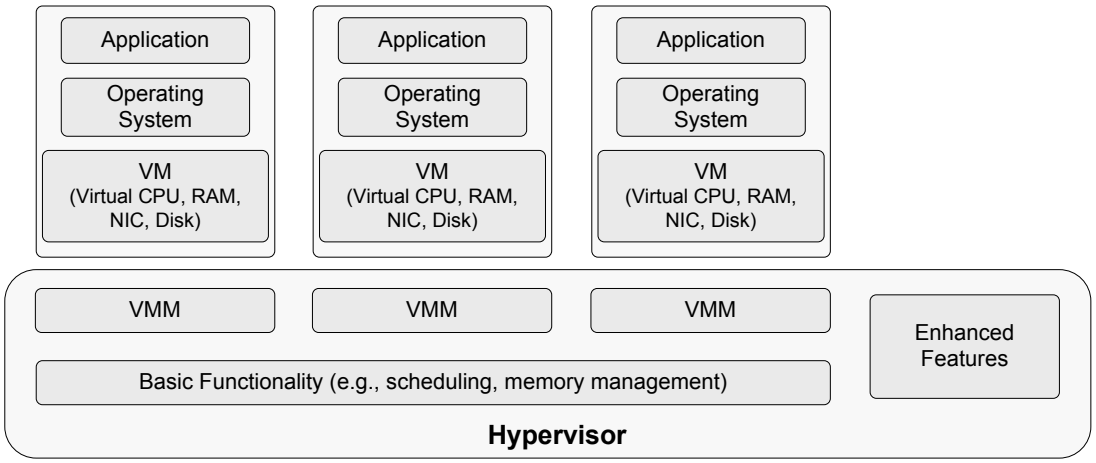

Figure 2.2: The Hypervisor-based Virtualization Architecture.

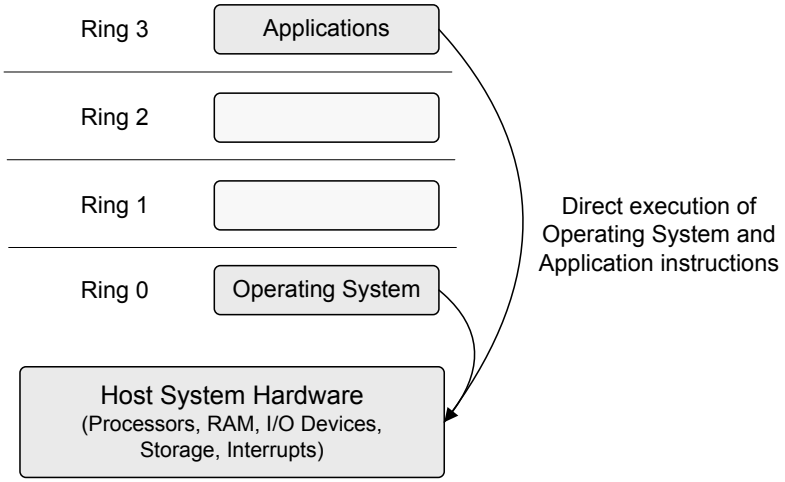

Figure 2.3: The x86 processor privilege rings without virtualization.

Virtualization of the x86 processor required placing the virtualization layer between the operating system and the hardware so that VMs can be created and managed that would share the same physical resources. This means the virtualization layer needs to be placed in Ring 0; however unmodified operating systems assumes to be run in the same Ring. Moreover, there are some sensitive instructions that have different semantics when they are not executed in Ring 0 and thus cannot be effectively virtualized. As a consequence, the industry and research community have come up with three types of alternative virtualization techniques presented in the following subsections.

2.3.1 Full Virtualization

Full Virtualization technique provides complete abstraction of the underlying hardware and facilitates the creation of complete VMs in which guest operating systems can execute [9]. This type of virtualization is achieved through a combination of binary translation and direct execution techniques that allow the VMM to run in Ring 0. The binary translation technique translates the OS kernel level code with alternative series of instructions in order to substitute the non-virtualizable instructions so that it has the intended effect on the virtual hardware (Figure 2.4(a)). As for the user level codes, they are executed directly on the processor to achieve high performance. In this way, the VMM provides the VM with all the services of the physical machine like virtual processor, memory, I/O devices, BIOS, etc. This approach have the advantage of providing total virtualization of the physical machine as the guest operating system is fully abstracted and decoupled from the underlying hardware separated by the virtualization layer. This enables unmodified operating systems and applications to run on VMs, being completely unaware of the virtualization. It also facilitates efficient and simplified migration of applications and workloads from one physical machine to another. Moreover, full virtualization provides complete isolation of VMs that ensures high level of security. VMware ESXi Server and Microsoft Hyper-V[21] are examples of full virtualization.

[21] Microsoft Hyper-V, 2015. `https://technet.microsoft.com/en-us/windowsserver/dd448604.aspx`

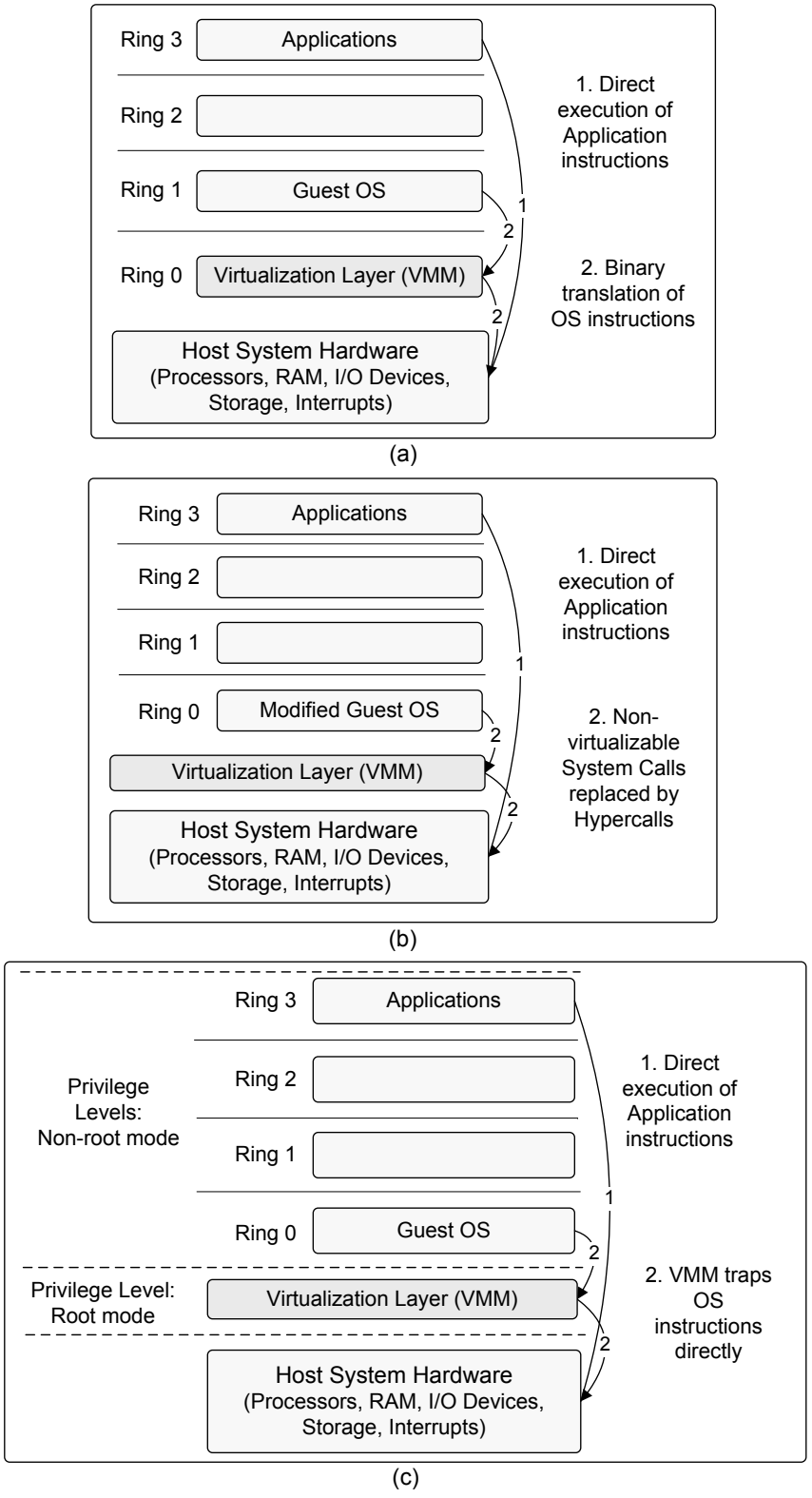

Figure 2.4: Alternative virtualization techniques: (a) Full Virtualization through binary translation, (b) Paravirtualization, and (c) Hardware Assisted Virtualization.

2.3.2 Paravirtualization

Different from the binary translation technique of full virtualization, *Paravirtualization* (also called *OS Assisted Virtualization*) works through the modification of the OS kernel code by replacement of the non-virtualizable instructions with hypercalls that communicate directly with the hypervisor virtualization layer [29] (Figure 2.4(b)). The hypervisor further provides hypercall interfaces for special kernel operations such as interrupt handling, memory management, timer management, etc. Thus, in paravirtualization each VM is presented with an abstraction of the hardware that is similar but not identical to the underlying physical machine. Since paravirtualization requires modification of guest OSs, they are not fully unaware of the presence of the virtualization layer. The primary advantage of paravirtualization technique is lower virtualization overhead over full virtualization where binary translations affect instruction executing performance. However, this performance advantage is dependent on the types of workload running on the VMs. Paravirtualization suffers from poor compatibility and portability issues since every guest OS running on it top of paravirtualized machines needs to be modified accordingly. For the same reason, it causes significant maintenance and support issues in production environments. Example of paravirtualization is the open source Xen project [29] that virtualizes the processor and memory using a modified Linux kernel and virtualizes the I/O subsystem using customized guest OS device drivers.

2.3.3 Hardware Assisted Virtualization

In response to the success and wide adaptation of virtualization, hardware vendors have come up with new hardware features to help and simplify virtualization techniques. Intel Virtualization Technology (VT)[22] and AMD-V[23] are first generation virtualization supports allow the VMM to run in a new root mode below Ring 0 by the introduction of a new CPU execution mode. With this new hardware assisted feature, privileged and critical system calls are automatically trapped by the hypervisor and the guest OS state is saved in Virtual Machine Control Structures (VT-x) or Virtual Machine Control Blocks (AMD-V), removing the need for either binary translation (full virtualization) or paravirtualization (Figure 2.4(c)). The hardware assisted virtualization has the benefit that unmodified guest

[22]Intel Virtualization Technology, 2015. `http://www.intel.com.au/content/www/au/en/virtualization/virtualization-technology/intel-virtualization-technology.html`

[23]AMD Virtualization, 2015. `http://www.amd.com/en-us/solutions/servers/virtualization`

OSs can run directly and access to virtualized resources without any need for modification or emulation. With the help of the new privilege level and new instructions, the VMM can run at Ring -1 (between Ring 0 and hardware layer) allowing guest OS to run at Ring 0. This reduces the VMM's burden of translating every privileged instruction, and thus helps achieve better performance compared to full virtualization. The hardware assisted virtualization requires explicit virtualization support from the physical host processor, which is available only to modern processors.

Among the various virtualization systems, VMware, Xen [9], and KVM (Kernel-based Virtual Machine) [70] have proved to be the most successful by combing features that make them uniquely well suited for many important applications:

- VMware Inc. is the first company to offer commercial virtualization technology. It offers VMware vSphere (formerly VMware Infrastructure 4) for computer hardware virtualization that includes VMware ESX and ESXi hypervisors that virtualize the underlying hardware resources. VMware vSphere also includes vCenter Server that provides a centralized point for management and configuration of IT resources, VMotion for live migrating VMs, and VMFS (Virtual Machine File System) that provides a high performance cluster file system. VMware products support both full virtualization and paravirtualization.

- Xen Server is one of a few Linux hypervisors that support both full virtualization and paravirtualization. Each guest OS (termed Domain in Xen terminology) uses a pre-configured share of the physical server. A privileged Domain called Domain0 is a bare-bone OS that actually controls physical hardware and is responsible for the creation, management, migration, and termination other VMs.

- KVM also provides full virtualization with the help of hardware virtualization support. It is a modification to the Linux kernel that actually makes Linux into a hypervisor on inserting a KVM kernel module. One of the most interesting KVM features is that each guest OS running on it is actually executed in user space of the host system. This approach makes each guest OS look like a normal process to the underlying host kernel.

2.4 Virtual Machine Migration Techniques

One of the most prominent features of the virtualization system is the *VM Live Migration* [26] which allows for the transfer of a running VM from one physical machine to another, with little downtime of the services hosted by the VM. It transfers the current working state and memory of a VM across the network while it is still running. Live migration has the advantage of transferring a VM across machines without disconnecting the clients from the services. Another approach for VM migration is the *VM Cold* or *Static Migration* [112] in which the VM to be migrated is first shut down and a configuration file is sent from the source machine to the destination machine. The same VM can be started on the target machine by using the configuration file. This is a much faster and easier way to migrate a VM with negligible increase in the network traffic; however static VM migration incurs much higher downtime compared to live migration. Because of the obvious benefit of uninterrupted service and much less VM downtime, live migration has been used as the most common VM migration technique in the production data centers.

The process of VM live migration is much more complicated than just transferring the memory pages of the VM from the source machine to the destination machine. Since a running VM can execute write instructions to memory pages in the source machine during the memory copying process, the new dirty pages must also be copied to the destination. Thus, in order to ensure a consistent state of the migrating VM, copying process for all the dirty pages must be carried out until the migration process is completed. Furthermore, each active VM has its own share and access to the physical resources such as storage, network, and I/O devices. As a result, the VM live migration process needs to ensure that the corresponding physical resources in the destination machine must be attached to the migrated VM.

Transferring VM memory from one machine to another can be carried out in many different ways. However, live migration techniques utilize one or more of the following memory copying phases [26]:

- ***Push phase:*** The source host VMM pushes (i.e., copies) certain memory pages across the network to the destination host while the VM is running. Consistency of VM's execution state is ensured by resending any modified (i.e., dirty) pages during this process.

- ***Stop-and-copy phase:*** The source host VMM stops the running VM on certain stop condition, copies all the memory pages to the destination host, and a new VM is started.

- ***Pull phase:*** The new VM runs in the destination host and, if a page is accessed that has not yet been copied, a page fault occurs and this page is copied across the network from the source host.

Performance of any VM live migration technique depends on the balance of the following two temporal parameters:

1. ***Total Migration Time:*** The duration between the time when the migration is initiated and when the original VM may be discarded after the new VM is started in the destination host. In short, the total time required to move the VM between the physical hosts.

2. ***VM Downtime:*** The portion of the total migration time when the VM is not running in any of the hosts. During this time, the hosted service would be unavailable and the clients will experience service interruption.

2.4.1 Types of VM Migration

Incorporating the above three phases of memory copying, several VM live migration techniques are presented by the research communities with trade-offs between the total migration time and VM downtime:

Pure stop-and-copy

In Pure stop-and-copy VM migration technique [104], the VM is shut down at the source host, all the memory pages are copied to the destination host, and a new VM is started. This technique is simple and, the total migration time is relatively small compared to other techniques and directly proportional to the size of the active memory of the migrating VM. However, the VM can experience high VM downtime, subject to the memory size, and as a result, this approach can be impractical for live services.

Pure demand-migration

In Pure demand-migration migration technique [131], the VM at the source host is shut down and essential kernel data structures (e.g., CPU state, registers, etc.) are transferred

to the destination host using a short stop-and-copy phase. The VM is then started in the destination host. The remaining pages are transferred across the network when they are first referenced by the VM at the destination. This approach has the advantage of much shorter VM downtime; however the total migration time is generally much longer since the memory pages are transferred on-demand upon page fault. Furthermore, post-migration VM performance is likely to be hampered substantially due to large number of page faults and page transfers across the network.

Post-copy migration

Similar to the pure demand-migration approach, in Post-copy migration migration technique [61], the VM is suspended at the source host, a minimal VM kernel data structure (e.g., CPU execution state, registers values, and non-pageable memory) is transferred to the destination host, and the VM is booted up. Unlike pure demand-migration, the source VMM actively sends the remaining memory pages to the destination host, an activity termed pre-paging. When the running VM at the destination attempts to access a page that is not copied yet, a page fault occurs (known as network faults) and the faulted page is transferred from the source host to the destination host over the communication network. As in the case of pure demand-migration, post-copy migration suffers from VM performance degradation due to on-demand page transfer upon page fault. However, pre-paging technique can help reduce the performance degradation by adapting the page transmission order dynamically in response to the network faults by pushing the pages near the last page fault.

Pre-copy migration

Unlike the above approaches, in Pre-copy migration technique [26], the VM continues running in the source host while the VMM iteratively transfers memory pages to the destination host. Only after a substantial amount of memory pages are copied, or a predefined number of iterations are completed, or any other terminating condition is met, the VM is stopped at the source, the remaining pages are transferred to the destination, and the VM is restarted. Pre-copy migration has the obvious benefit of short stop-and-copy phase since most of the memory page would be copied to the destination by this time. So, the VM downtime is comparatively much shorter than other live migration techniques, making this approach suitable for live services. Furthermore, pre-copy migration offers

higher reliability since it retains an up-to-date state of the VM in the source machine during the migration process, an added advantage absent in other migration approaches. However, pre-copy migration can suffers from longer total migration time since the same memory pages can be transmitted multiple times in several rounds depending on page dirty rate. For the same reason, it can generate much higher network traffic compared to other techniques. Almost all the modern virtualization environments offers VM live migration feature, including Xen Server (through XenMotion [26]), VMware ESXi Server (through VMotion [93]), KVM, Microsoft Hyper-V, Oracle VM VirtualBox, and OpenVZ. A high level flow chart of the logical steps followed during the pre-copy migration technique implemented in Xen Server is depicted in Figure 2.5 [26]. Focusing primarily on high reliability against system failure, the Xen pre-copy migration takes a transactional approach between the source and target hosts:

- *Stage 0 (Pre-migration):* Source host A has an active VM to be migrated. The target host B can be pre-selected in advance in order to speed up future migrations through guaranteed resources required for the migration process.

- *Stage 1 (Reservation):* The request to migrate the VM from source host A to target host B is issued. Host B confirms that it has the required resources and reserves a VM container of that size. If host B fails to secure enough resources, the migration request is discarded and the VM runs on host A without any changes.

- *State 2 (Iterative pre-copy):* In the first iteration, all the memory pages are transmitted (i.e., copied) from host A to host B. In the remaining iterations, only the pages that have been modified during the previous iteration are transmitted.

- *Stage 3 (Stop-and-Copy):* The VM is shut down in host A and all the network traffic is redirected to host B. Then, the critical kernel data structures (e.g., CPU states and registers) and the remaining dirty pages are transmitted. At the end of this stage, the two copies of the VM at both host A and B are consistent; however, the copy at A is still considered primary and is resumed in the incident of failure.

- *State 4 (Commitment):* Host B notifies host A that it has a consistent VM image. Upon receipt, host A sends the acknowledgment message indicating the commitment of the total migration transaction. After this point, the original VM at host A can be abandoned and host B is considered as the primary host of the VM.

- *State 5 (Activation):* Host B activates the migrated VM. The post-migration code runs in order to reattach the device drivers at host B and advertise the moved IP addresses.

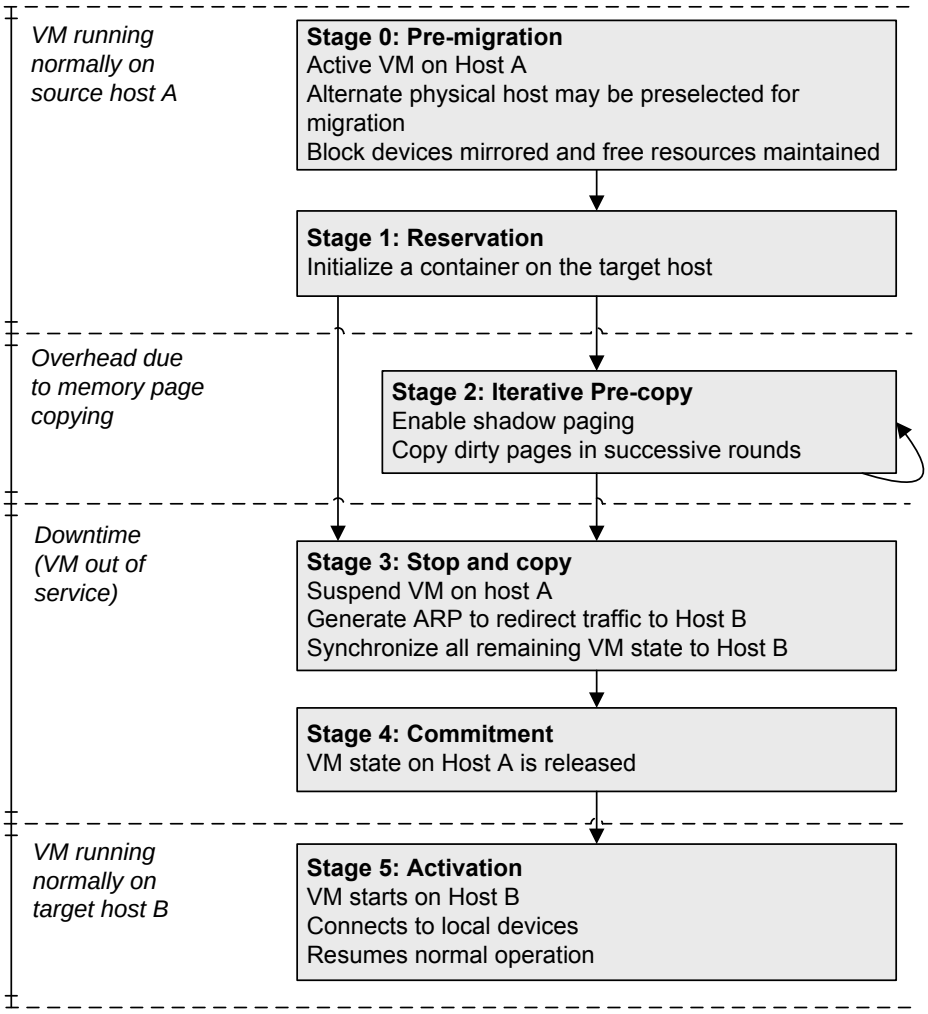

Figure 2.5: Stages of the Pre-copy VM Live Migration Technique [26].

2.5 Data Center Network Architectures

Modern data centers are built primarily according to the generic multi-tier architecture[24]. The most common network topologies follow the three-tier architecture (Figure 2.6), where each tier has specific responsibility and goal in the design and traffic handling. In the bottom tier, known as the *Access Tier* every physical server is connected to one or two (in case of redundancy to increase reliability) access switches, in the *Aggregation*

[24]Cisco Data Center Infrastructure 2.5 Design Guide, 2014. `https://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf`

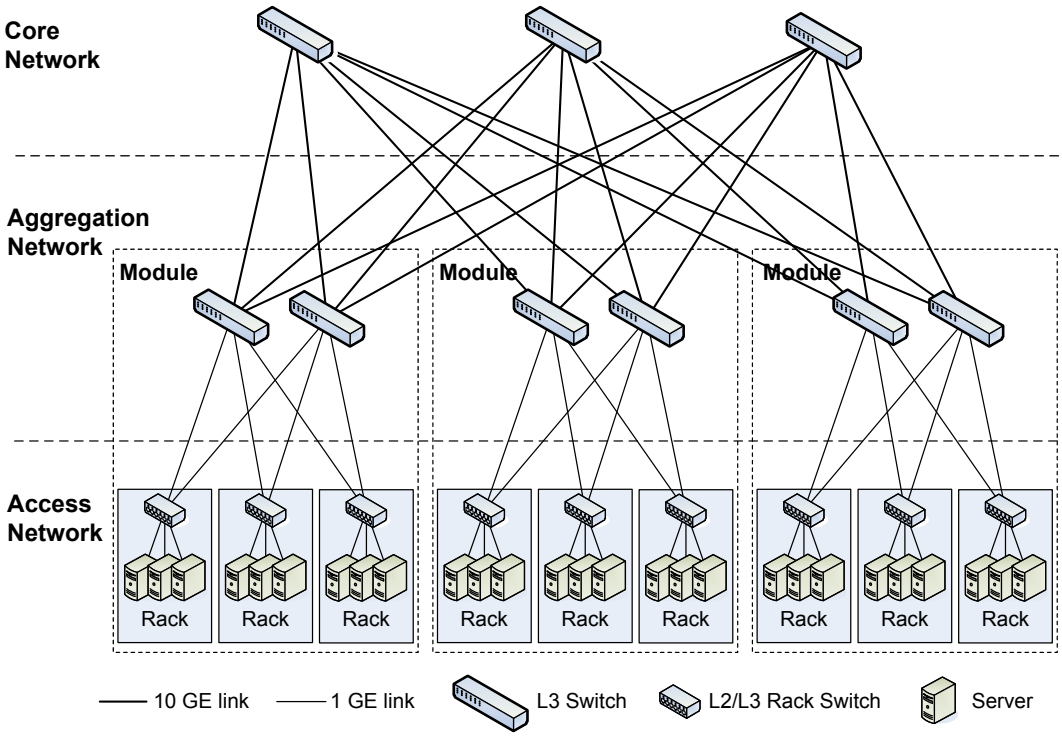

Figure 2.6: The Three-tier Network Architecture.

Tier, each access switch is connected to one or two aggregation switches, and in the *Core Tier* each aggregation switch is connected to more than one core switches. The access switches provide the servers connectivity to other servers and to the upper tiers, the aggregate switches interconnects between the access switches and enables localization of traffic among the servers, and finally, the core switches connects the aggregation switches in such a way that there exists connectivity among each pair of servers and also includes gateways for the traffic to communicate outside the data center.

In three-tier network architectures, the access tier links are normally 1 Gigabit Ethernet (GE) links. Although 10 GE transceivers are available in the commodity market, they are not used for the following reasons: (1) very high price and (2) bandwidth capacity is much more than needed by the physical servers. Servers in data centers are normally grouped in ranks and rack connectivity is achieved through the use of not-so-expensive Top-of-Rack (ToR) switches. Typically, such ToR switches have two 10 GE uplinks with 48 GE links that interconnects the servers within the rack. *Oversubscription Ratio* of a switch is defined the difference between the downlink and uplink capacities of the switch and in this case it is 48:20 or 2.4:1. As a result, though each access link has 1 GE capacity, under full load, only 416 Mb/s will be available to each server [71]. At the aggregation

and core tier, the racks are organized in *modules* with a couple of aggregation switches and oversubscription ratio for these switches is around 1.5:1. Therefore, the available bandwidth for each server is reduced to 277 Mb/s.

Though such network architectures have *multi-rooted forest topology* at the physical level, because of the extensive use of *Virtual LANs* (VLANs) and *Spanning Tree* algorithm the network packets are forwarded according to the logical layer-2 topology. Such layer-2 logical topology always takes the form of a tree, normally rooted at one of the core switches.

Scalability issue of three-tier architecture is normally addressed through scaling up each individual switches by increasing their fan-outs, not by the scaling out of the network topology. For example, according to the Cisco Data Center Infrastructure 2.5 Design Guide, the core tier can have a maximum of 8 switches. Because of such scalability issues regarding topology scaling, high oversubscription ratio, as well as requirement for flat address space, several recent research endeavors produced complex network architectures for the large scale modern data centers and among these, the following are considered as the standard-de-facto solutions:

- ***Fat-tree:*** This is a three-tier architecture based on bipartite graphs [5] and basic building block of this topology is called pods which are collections of access and aggregation switches connected in a complete bipartite graph. Every pod is connected to all the core switches; however links that connect pods to core switches are uniformly distributed between the aggregation switches contained within the pods. Such connection pattern results in a new bipartite graph between aggregation and core switches. In this topology, all the switches need to have same number of ports. The primary advantage of fat-tree topology is that N2/4 paths are available to route the traffic between any two servers.

- ***VL2:*** Somewhat similar to fat-tree, VL2 [52] is also a three-tier topology having a complete bipartite graph between core and aggregation switches, rather than between access and aggregation switches. Moreover, access switch traffic is forwarded through the aggregation and core switches using valiant load balancing techniques that forwards the traffic first to a randomly selected core switch and then back to the actual destination switch. The advantage of such routing is that when traffic is unpredictable, the best way to balance load across all the available network links

is to forward the packets to a randomly selected core switch as an intermediate destination.

- ***PortLand:*** This is also a three-tier architecture that shares the same bipartite graph feature with VL2, however at different levels [91]. It makes use of fat-tree topologies [74] and uses the concept of pods. Such pods are collections of access and aggregations switches that form complete bipartite graphs. Furthermore, each pod is connected to all the core switches, by uniformly distributing the up-links between the aggregation switches of the pod. As a result, another level of bipartite graph is formed between the pods and the core switches. Portland requires that the number of ports of all the switches is same. The number of ports per switch is the only parameter that determines the total number of pods in the topology, and consequently the total number of switches and hosts machines.

- ***BCube:*** It is a multi-level network architecture for the data center defined in a recursive fashion [54]. Host machines are considered as part of the network architecture and they forward packets on behalf of other host machines. It is based on the generalized hypercube architecture [14] with the main difference that the neighboring hosts instead of forming a full mesh network with each other, they connect through switches. In a BCube topology, the total number of connected hosts machines and the total number of required switches is a function of the total number of ports of each switch.

2.6 Cloud Applications and Data Center Traffic Patterns

With the increasing popularity of Cloud hosting platforms (e.g., Amazon AWS and Microsoft Azure) due to the benefits of pay-as-you-go business model, high availability and reliability, as well as extensive computing and storage services, Cloud platforms are enjoying deployment of a wide variety of composite applications, including scientific applications, social networks, video streaming, medical services, search engines and web browsing, various content delivery applications, and so on [24, 63, 118]. Such composite applications are generally composed of multiple compute VMs backed by huge amount of data. As more and more communication-intensive applications are being deployed in data centers, the amount of inter-VM traffic is increasing with rapid pace. Based on the

dynamics on computational and communication requirements, the commonly deployed Cloud application workloads are categories into the following three groups [71]:

1 ***Data-Intensive Workloads:*** Such workloads require less computational resources, but cause heavy data transfers. For example, video file sharing where each user request generates a new video streaming process. For such applications, it is the interconnection network that can be a bottleneck rather than the computing power. In order to maintain the application performance and respect the SLAs, a continuous feedback mechanism need to be present between the network devices (e.g., switches) and the centralized workload scheduler or placement manager. Based on feedback, the scheduler will decide the placement of the workloads with consideration of the run-time network status and congestion levels of communication links. In this way, placement of workloads over congested network links can be avoided even though corresponding servers have enough computing capacity to accommodate the workloads. As a result, data center traffic demands can be distributed over the network in a balanced way and minimize network latency and average task completion time.

2 ***Computationally Intensive Workloads:*** This category represents the *High Performance Computing* (HPC) applications that are used to solve advanced and computationally expensive problems. These applications require very high amount of computing capacity, but causes little data transfer over the communication network. Such applications can be grouped together and placed in a minimum number of computing servers through VM consolidation mechanisms in order to save energy. Because of low data traffic among the VMs, there is very less probability of network congestion and most of network switches can be turned into lower power states (e.g., in sleep mode) and thus help reducing energy consumption in the data center.

3 ***Balanced Workloads:*** Applications that require both computing power and data transfer among the computing nodes (i.e., VMs) are represented by balanced workloads. For example, Geographic Information Systems (GISs) need to transfer large volume of graphical data as well as huge computing resources to process these data. With this type of workloads, the average compute server load is proportional to the amount of data volume transferred over the communication networks. VM placement and scheduling

policies for such application need to account for both current state of compute servers' load and traffic loads on the network switches and links.

Since Cloud data centers host heterogeneous services and application, communication patterns exhibit wide spectrum of variations, ranging from one-to-one and all-to-all traffic matrices. Based on trace analysis of network usage from production data centers, the following trends of network traffic are found to be predominant [41, 69, 85]:

1 ***Highly non-uniform distribution of traffic volume among VMs:*** VMs running on servers exhibit uneven traffic volume among themselves across different VMs. The trace analysis reports show that 80% of the VMs have relatively low traffic rate (800Kbyte/min) over a period of two-weeks, 4% of the VMs have a rate ten times higher. This concludes that the inter-VM traffic rate varies significantly and it is quite hard for the data center administration to estimate the amount of inter-VM traffic accurately and consistently.

2 ***Stable inter-VM traffic volume:*** For a long duration, the average inter-VM traffic rate is found to be relatively stable in spite of the highly skewed traffic rate among VMs. Meng *et al.* [85] showed that for the majority of the VMs, the standard deviation of their traffic rates is less than the double of the mean of the traffic rates. This consistent traffic volume among VMs implies that the run-time communication patterns among the VMs can be estimated and known a priory from the users deploying the VMs in the Clouds.

3 ***Weak correlation between traffic rate and network latency:*** It is further reported from the measurement-based study that there is no any dependency or relationship between inter-VM traffic volume and the network distance between the servers hosting the VMs. That means VM pairs with high traffic rate do not necessarily correspond to low latency and vice versa.

2.7 Virtual Machine Consolidation

While Cloud Computing provides many advanced features, it still has some shortcomings, such as the relatively high operating costs for both public and private clouds. The area of *Green Computing* is also becoming increasingly important in a world with

limited energy resources and an ever-rising demand for computational power. As pointed out before, energy costs are among the primary factors that contribute to the Total Cost of Ownership (TCO) and its influence will grow rapidly due to the ever increasing demands of resources and continuously increasing electricity costs [53]. As a consequence, optimization of energy consumption through efficient resource utilization and management is equivalent to the reduction in operating cost in the context of data center management. In order to optimize the energy consumption of the physical devices, different techniques have been proposed and used, including VM Consolidation, energy-aware resource management frameworks and design strategies, as well as energy-efficient hardware devices.

Resource management and optimization is getting more challenging day-by-day for large-scale data centers, such as the Cloud data centers, due to their rapid growth, high dynamics of hosted services, resource elasticity, and guaranteed availability and reliability. Static resource allocation techniques, that are used in traditional data centers, are simply inadequate to address these newly emerged challenges [60]. With the advent of virtualization technologies, server resources are now better managed and utilized through server consolidation by placing multiple VMs hosting several applications and services in a single physical server, and thus ensuring efficient resource utilization. Energy-efficiency is achieved by consolidating the running VMs in minimum number of servers and transitioning idle servers into lower power states (i.e., sleep or shut-down mode).

VM consolidation techniques provide VM placement decisions that indicates the mapping of each running VM to appropriate server. Depending on the initial condition of data centers, that VM consolidation techniques start with, it is categorized into two variants: *Static* and *Dynamic VM Consolidation.*

2.7.1 Static VM Consolidation

The static VM consolidation techniques start with a set of fully empty physical servers, either homogeneous or heterogeneous with specific resource capacity and a set of workloads in the form of VMs with specific resource requirements. Thus, such consolidation mechanisms require prior knowledge about all the workloads and their associated resource demands. Such techniques are useful in situations like initial VM placement phase or migration of a set of workload from one data center to another. Static consolidation does not

consider the current VM-to-server assignments and thus unaware of the associated VM migration overheads on hosted application performance, hosting physical machines, and underlying network infrastructure [45]. Considering the predominant energy costs of running large data centers and low utilization of servers resulted by traditional resource management technologies, and through the blessings of virtualization techniques, VM placement strategies like server consolidation have become a hot area of research [46, 50, 59, 117].

2.7.2 Dynamic VM Consolidation

VM consolidation mechanisms that consider the current VM-to-server assignments for the consolidation decision fall in the category of dynamic consolidation. Contrary to static consolidations where the current allocations are disregarded and whole new solution of VM placement is constructed without considering the cost of reallocation of resources, dynamic consolidation techniques include the cost or overhead of relocation of existing workloads into the modeling of consolidation and try to minimize relocation overhead and maximize consolidation. Such server consolidation mechanisms employ VM live or cold migration techniques [26, 93] to move around workloads from servers with low utilization and consolidate them into minimum number of servers, thus improving overall resource utilization of the data center and minimizing power consumption.

As clouds offer an on-demand pay-as-you-go business model, customers can demand any number of VMs and can terminate their VMs when needed. As a result, VMs are created and terminated in the cloud data centers dynamically. This causes resource fragmentation in the servers, and thus leads to degradation in server resource utilization. However, efficient resource management in clouds is not a trivial task since modern service applications exhibit highly variable work-loads causing dynamic resource usage patterns. As a result, aggressive consolidation of VMs can lead to degradation of performance when hosted applications experience an increasing customer demand resulting in a rise in resource usage. Since cloud providers ensure reliable Quality of Service (QoS) defined by SLAs, resource management systems in cloud data centers need to deal with the energy-performance trade-off.

In order to estimate the cost of relocation of workloads by the dynamic VM consolidation techniques, several system and network level metrics and parameters are used as modeling elements, such as the number of VM live migrations required to achieve the new VM-to-server placement [45], VM active memory size, speed of network links used for

the migration [4, 60, 121], page dirty rate [120], as well as application-specific performance model [64].

2.7.3 Pros and Cons of VM Consolidation

Virtualization technologies have revolutionized the IT management works and opened up a new horizon of opportunities and possibilities. It has enabled application environments to be compartmentalized and encapsulated within VMs. By the use of server virtualization and VM live migration techniques, virtualized data centers have emerged as highly dynamic environments where VMs hosting various applications are created, migrated, resized, and terminated instantaneously as required. Utilizing virtualization, information technology infrastructure management has widely adapted VM consolidation techniques to reduce operating costs and increase data center resource utilization. The most notable advantages of adopting VM consolidation techniques are mentioned below:

1. ***Reduction in Physical Resources:*** By the help of efficient dynamic VM consolidation, multiple VMs can be hosted in single physical server without compromising hosted application performance. As a result, compared to static resource allocations where computing resources such as CPU cycles and memory frequently lay idle, through dynamic VM consolidation fewer numbers of physical machines can provide the same QoS and maintain SLAs, and thus effectively cut the Total Cost of Ownership (TCO). Reduction in the number of servers also implies reduction in the cooling equipment necessary for the cooling operations in data centers.

2. ***Energy Consumption Minimization:*** Unlike other approaches of energy efficiency (e.g., implementing efficient hardware and operating systems), VM consolidation is a mechanism under the disposal of data center management team. If same level of service can be provided by fewer servers through VM consolidation, it implies minimization of energy costs both for the running servers and the operating cool systems. As energy costs continue to escalate, this implies a significant saving that will continue during the course of the data center operation.

3. ***Environmental Benefits:*** World data centers contribute a significant portion of Greenhouse Gas (GHG) emission and thus, have enormous effects of environment. With recent trend towards Green Data Centers, VM consolidation is a major business drive for the information technology industry to contribute to the Green Computing.

4. ***Minimization of Physical Space:*** Reduction in the number of hardware implies reduction in terms of space needed to accommodate the servers, storage, network, and cooling equipment. Again, this contributes to the reduction of TCO and operating costs.

5. ***Reduced Labor Costs:*** A major portion of TCO of data centers is derived from administrative, support, and outsourced services, and thus, VM consolidation can help trim down these costs significantly by reducing the maintenance effort.

6. ***Automate Maintenance:*** By incorporating autonomic and self-organizing VM consolidation and VM migration techniques, much of the administrative and support tasks can be reduced and automated, and therefore, it can further reduce the maintenance overhead and associated costs.

With all the above-mentioned benefits, if not managed and applied appropriately VM consolidation can be detrimental to the services provided by the data center:

1. ***System Failure and Disaster Recovery:*** VM Consolidation puts multiple VMs hosting multiple service applications in a single physical server, and therefore can create single-point-of-failure (SPOF) for all the hosted applications. Moreover, upgrade and maintenance of a single server can cause multiple applications to be unavailable to users. Proper replication and disaster recovery plans can effectively remedy such situations. Since VMs can be saved in storage devices as disk files, virtualization technologies provide tools for taking snapshots of running VMs and resuming from saved checkpoints. Thus, with the help of shared storage such as Network Attached Storage (NAS) or Storage Area Network (SAN), virtualization can be used as convenient disaster recovery tool.

2. ***Effects on Application Performance:*** Consolidation can have adverse effects on hosted application performances due to resource contention since they would share the same physical resources. Delay sensitive applications such as Voice-over-IP (VoIP) and online audio-visual conferencing services as well as database management systems that require heavy disk activity need to be given special consideration during resource allocation phase of VM consolidation. Such applications can be given dedicated resources whereas delay-tolerant and less resource hungry applications can be scheduled with proper workload prediction and VM multiplexing schemes.

3. ***VM Migration and Reconfiguration Overheads:*** Performing VM consolidation dynamically requires VM live migrations that have overheads on network links of the data center as well as on the CPU cycles of servers executing the migration operations. As a consequence, VM migrations and post-migration reconfiguration can have non-negligible impact on application performance. Experimental results [122] show that applications that are being migrated as well as co-located applications can suffer from performance degradation due to VM live migrations. As a consequence, VM consolidation mechanisms need to minimize the number of VM live migrations and its effects on applications.

Despite all the drawbacks of VM consolidation, due to its benefits in continuous reduction in energy and operating costs and increasing resource utilizations data center owners are increasing adopting VM consolidation mechanisms, especially for large data centers. Since VM consolidation can have adverse effects on application performance, various characteristics and features of data center resources and hosted applications need to be taken into account during the design and implementation of VM consolidation schemes, such as heterogeneity of servers and storage devices, system software and tools, middleware and deployment platforms, physical and virtual network parameters, as well as application types, workload patterns, and load forecasting.

2.8 Summary and Conclusions

This chapter has presented an overview of the various concepts, elements, systems, and technologies relating to resource management and optimization in the context of large-scale data centers, such as the Cloud data centers. A brief description on the architectural features and components, as well as different service models of the newly emerged Cloud distributed computing system are presented. Since virtualization technology is one of the main technological elements that has paved the way for the extreme success of the Cloud Computing, various aspects the virtualization and VM live migration are also described. Moreover, data center network architectures and communication patters of the hosted applications are also presented to provide an overview on the properties of networks and communications within typical data centers. Furthermore, since the main focus of this thesis is data center-level resource management and energy consumption, leveraging VM

placement and consolidation strategies, this chapter has presented a brief background on the VM consolidation approaches, including their benefits and potential drawbacks.

Chapter 3

Taxonomy and Survey on Virtual Machines Management

This chapter presents a comprehensive taxonomy and survey of the existing works VM management in the context large-scale data centers. A brief introduction to the salient features of various state-of-the-art VM management techniques is also presented. Moreover, a detailed taxonomy and survey of recent notable research contributions has been delineated.

3.1 Introduction

Cloud Computing is quite a new computing paradigm and from the very beginning it has been growing rapidly in terms of scale, reliability, and availability. In order to meet the increasing demand of computing, storage, and communication resources, infrastructure Cloud providers are deploying planet-scale data centers across the world, consisting of thousands of servers. These data centers extensively use virtualization technologies in order to utilize the underlying physical resources effectively and with much higher reliability. Optimization of VM placement and migration decisions has been proven to be practical and effective in the arena of physical server resource utilization and energy consumption reduction, and a plethora of research contributions have already been made addressing such problems. Several research attempts have been made to address the VM management problems focusing on physical server resource utilization, network bandwidth optimization, energy consumption minimization, server load and resource constraints, computing and network resource demands of VMs, data storage locations, and so on. These works

not only differ in the addressed system assumptions and modeling techniques, but also vary considerably in the proposed solution approaches and the conducted performance evaluation techniques and environments. As a consequence, there is a rapidly growing need for comprehensive taxonomy and survey of the existing works in this research area. In order to analyze and assess these works in a uniform fashion, this chapter presents the salient features of various state-of-the-art VM management techniques, taxonomy and survey of notable research contributions.

The rest of the chapter is organized as follows. Section 3.2 presents a description of the various nomenclatural aspects, followed by a detailed taxonomy. Section 3.3 presents analysis and survey on the recent prominent VM management studies. Finally, Section 3.4 concludes the chapter with a summary of the contribution.

3.2 Taxonomy of Virtual Machine Placement, Migration, and Consolidation

With the various intricacies of virtualization technologies, enormous scale of modern data centers, and wide spectrum of hosted applications and services, different VM management strategies and algorithms are proposed with various assumptions and objectives. Figure 3.1 presents a full taxonomy of the various aspects of VM management, including placement, migration, and consolidation.

A brief description of the identified aspects of the research works used in the course of taxonomy is provided below:

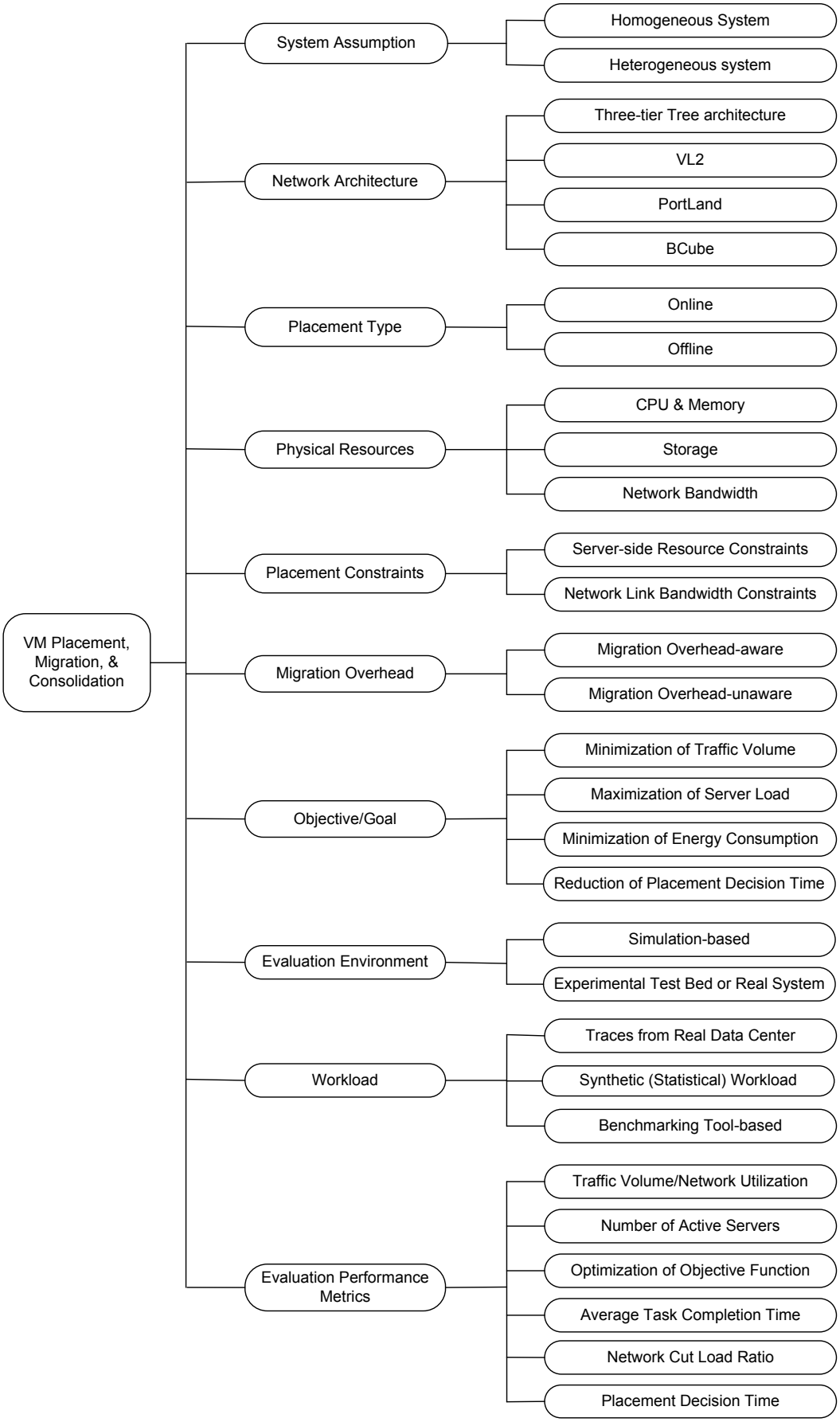

Figure 3.1: Taxonomy of VM Placement, Migration, and Consolidation

1. **System Assumption:** Physical servers and network resources in data centers or IT infrastructures are primarily modeled as homogeneous, and often times as heterogeneous as well. Homogeneous cluster of servers normally represent servers with same capacity for certain fixed types of resources (e.g., CPU, memory, and storage), whereas heterogeneous cluster of servers can either mean servers having different capacities of resources or different types of resources (e.g., virtualized servers powered by Xen or VMware hypervisor, and servers with Graphics Processing Units or GPUs). In practice, commercial data centers evolve over time and thus different parts of the data center can have devices with different capabilities and properties. It is quite common that a recent server installed in a data center would have much higher computing power compared to the old ones; similarly a network switch can be more recent than others and thus can have lower network latency and higher I/O throughput capacity. Moreover, recently there is growing trends towards deploying multi-purpose hardware devices that increase the degree of heterogeneity in data centers. Example of such devices can be some storage devices, such as IBM DS8000 series that have built-in compute capability (POWER5 logical portioning LPAR) that can host applications [1, 72] and network switches, such as Cisco MDS 9000 switches (Cisco MDS 9000 SANTap[1]) that have additional x86 processors capable of executing applications. Efficiency and effectiveness of VM placement and migration strategies are dependent on the assumed system assumptions and properties. VM placement techniques that consider the heterogeneity of the devices in data centers can efficiently utilize various capabilities of the divergent resources and optimize the placements, and thus can reduce the traffic burden and energy consumption.

2. **Physical Resources:** Generally, optimization across different ranges of resources (e.g., processing, memory, network I/O, storage, etc.) is harder than single resource optimization. Often various mean estimators (such as L_1 norm, volume-based, and vector algebra) are used to compute equivalent scalar representation while trying to optimize across multiple types of physical resource utilization. This aspect has direct influence on the modeling techniques applied in the research works as well as on the optimization efficiency. Inter-VM communication requirement is often modeled as *Virtual Link* (VL) which is characterized by the bandwidth demand. VM cluster

[1]Cisco MDS 9000 SANTap, 2014. `http://www.cisco.com/c/en/us/products/collateral/storage-networking/mds-9000-santap/data_sheet_c78-568960.html`

forming an application environment with mutual traffic demand is represented as a graph with VMs denoted by vertices and VLs denoted by edges of the graph.

3. **Network Architecture/Topology:** With the variety of proposed data center network architectures and intricacies of traffic patterns, different VM placement approaches are proved to be efficient for different types of network topologies and inter-VM traffic patterns. Such effectiveness is sometimes subject to the specific analytic or modeling technique used in the proposed placement and migration schemes. Since different network topologies are designed independently focusing on different objectives (e.g., VL2 is good for effective load balancing while BCube has higher degree of connectivity and network distances among hosts), different VM placement techniques see different levels performance gains for existing network topologies. For example, the TVMPP (Traffic-aware VM Placement Problem) optimization technique [85] gains better performance for multi-layer architecture such as BCube, compared to VL2.

4. **Modeling Technique:** As for any research problem, the solution approaches of various VM placement and migration schemes as well as their effectiveness and applicability are highly contingent on the applied modeling (mathematical, analytic, or algorithmic) techniques. Since different models have specific system assumptions and objectives, VM placement problems are presented using various optimization modeling techniques, such as Quadratic Assignment Problem (QAP), Knapsack Problem, Integer Quadratic Programming, Convex Optimization Problem, and so on. The characteristics of VM consolidation problem make it most resemble to the general multi-dimensional bin/vector packing problems. Furthermore, depending on project goal, modeling can vary across other theoretical problems such as Multiple Multi-dimensional Knapsack Problem, Constraint Satisfaction Problem, and Multi-objective Optimization Problems.

5. **VM Placement Constraints:** Individual VM placement feasibility or practicality involves server resource capacity constraints which means that the remaining resource (e.g., CPU cycles, memory, storage, etc.) capacities of the hosting servers need to be enough in order to accommodate the VM. Similarly, while placing two VMs with mutual communication requirement, the bandwidth demand of the VL

connecting the two VMs need to match with the remaining bandwidth capacities of the corresponding physical network links connecting the two hosting servers. When allocating CPU cycles to hosted VMs, static or dynamic threshold-based allocation mechanisms are applied so that processors never get 100% utilization in which case hosted applications can suffer from performance degradation.

6. **Migration Overhead-awareness:** During VM live migration process, additional network traffic is generated during the whole migration period since hypervisor needs to transfer in-memory states of the running VM to the target machine. Furthermore, VM migration causes unavailability of hosted applications due to the VM downtime factor. As a consequence VM living migration is identified as an expensive data center operation that should not be triggered very often [81]. Therefore, efficiency of a VM migration policy also dependents on the overheads associated to the migrations commands issued. While network-aware VM migration strategies opt for optimizing overall network usage and reduce the inter-VM communication delays through migrating communicating VMs into nearby hosts, most of the strategies do not consider the associated VM migration overheads in data center components and resulting application performance degradation. In the case of dynamic VM consolidation schemes, overall VM migration overhead is measured in terms of the number of migrations required for consolidation [45, 82, 90]. Such measure is shown in detail to be an over-simplification of the total migration overhead in Chapter 6.

7. **Goal/Objective:** Energy-aware VM placement and consolidation works primarily set the objective to minimize the overall power consumption of data center and improvement of server resource utilization by increasing the VM/workload packing efficiency using minimum number of active servers. With the consolidation process comes the trade-off between application performance (and hence, SLA) and power consumption. With given importance on SLA violations, some of the works consider the cost of reconfiguration primarily due to VM live migrations and thus incorporate this cost in the objective function modeling. Moreover, some the works further focus on automated and coordinated management frameworks with the VM consolidation as an integral component of the proposed frameworks. Network-aware VM placement and migration policies primarily target on minimization of overall network traffic overhead within the data center. The obvious way to achieve such goal is to

place VMs with large amount of traffic communication in neighboring servers with minimum network delays and enough available bandwidth, most preferably in the same server where the VMs can communicate through memory rather than network links. With this goal in mind, VM placement and migration problem is generally modeled as mathematical optimization framework with minimization objective function. Such objective function can be a measure of total amount of network traffic transferred with the data center, or network utilization of the switches at the different tiers of the network architecture. Since VM placement and migration decision needs to be taken during run-time, reduction of the placement decision time (i.e., problem solving time or algorithm execution time) is also considered as an additional objective.

8. **Solution Approach:** Given the above-mentioned placement constraints and objectives, VM placement and migration are in fact NP-complete problems since they require combinatorial optimization to solve them. As a consequence, solution approaches apply various heuristic methods ranging from simple greedy algorithms to metaheuristic strategies and local search methods so that the algorithms terminate in a reasonable amount of time. Such heuristics are not guaranteed to produce optimal placement decisions; however from time constraint perspective, exhaustive search methods are not practical, especially considering the scale of modern data centers. Greedy approaches such as First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) are very fast in producing results but are not guaranteed to produce optimal solutions. Metaheuristics such as Ant Colony Optimization (ACO), Genetic Algorithms (GA), and Simulated Annealing (SA) work on initial or existing solutions and refine them to improve on objective function value. Nevertheless, adaptation and utilization of these problem solving techniques are still open to explore to address the VM placement and migration problems given various physical constraints.

9. **Evaluation/Experimental Platform:** Evaluation methodologies have direct impact on the performance and practicality of the research works, most importantly in the competency analysis. Proposals that primarily have theoretical contributions mostly apply simulation based evaluation to focus highly on the algorithmic and complexity aspects. This, however, makes sense given the complexity and scale of

modern data centers and the hosted applications. On the other hand, works involving various workload patterns and application characteristics have attempted to validate their proposed placement policies using real test beds, experimental data centers, or on emulated platforms, and have reported various run-time dynamics across different performance metrics, which otherwise would be impossible to report through simulation-based evaluations. However, such evaluations are performed on small-scale test beds with 10 to 20 physical machines and thus, do not necessarily forecast the potential behavior and performance for large scale data centers.

10. **Competitor Approaches:** Comparison of the performance among various competitor placement approaches highly depends on the assumptions and goals of the competitors. Intelligent VM placement and migration policies at data center level is relatively new area of research and as a consequence proposed approaches are often compared to other placement policies that are agnostic to certain placement constraints, network traffic, and network topologies, and have different objectives set in the underlying algorithms.

11. **Workloads:** Because of the lack of enough VM workload data sets from large scale Cloud data centers or other production data centers due to their proprietary nature, statistical distribution-based VM load (compute resource and network bandwidth demands) generation is the most common approach adopted in the simulation-based evaluations. Among others, Normal, Uniform, Exponential, and Poisson distributions are usually used. Such synthetic workload data characterize randomness based on particular trend (e.g., through setting mean and variance in case of normal distribution). Subject to accessibility, workload traces from real data centers of often used to feed data to the simulation based evaluation to imply the effectiveness of the proposed approaches in real workload data. Evaluations based on experimental test beds mostly use real time workload data generated from the applications that are deployed and run in the test bed servers. Although such test beds can capture realistic behaviors of applications and systems, suffer from scalability issues in the domain of VM placement and consolidation.

12. **Evaluation Performance Metrics:** Depending of the goals of the VM placement and migration solutions, various performance metrics are reported in existing

research works that can be broadly categorized into energy efficiency, resource utilization, and SLA violations. It is also common to use specific objective function or cost value that captures the overall performance of the evaluated methods. Placement schemes that have multiple objectives, often try to balance between network performance gain and energy consumption reduction, and report evaluations based on both traffic volume reduction and number of active servers or electricity power consumption in data centers. From energy savings point of view, minimization of the number of active servers in data center through VM consolidation is always an attractive choice.

Figure 3.2 provides a categorization of the various published research works based on the addressed and analyzed subareas of the VM placement problem and the ultimate objectives of the VM placement and migration strategies.

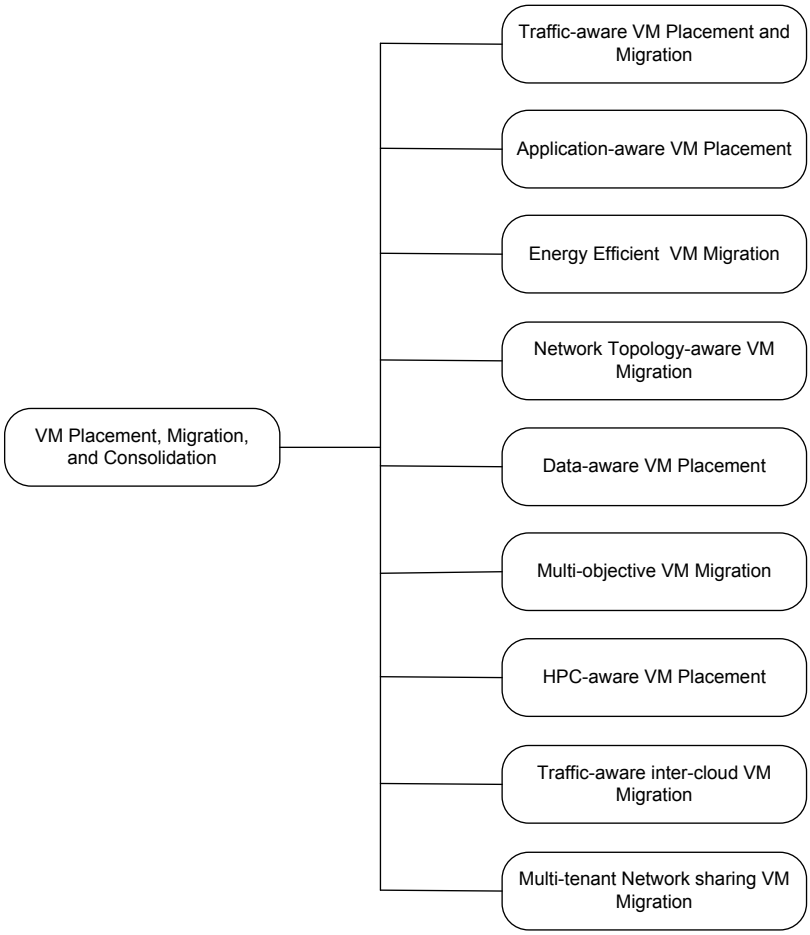

Figure 3.2: Categorization VM Placement, Migration, and Consolidation Approaches

Table 3.1: Aspects of Notable Research Works on Virtual Machines Management

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Multi-objective Virtual Machine Migration in Virtualized Data Center Environments [62]	Homogeneous data center	Tree, VL2, Fat-tree, BCube	Online	Max-min fairness and convex optimization framework	CPU, memory, and storage	Server resource capacity and inter-VM bandwidth requirement	Maximize utilization of physical servers and minimize data center network traffic	Two-staged greedy heuristic (exhaustive search based)	Simulation based on synthetic data center and load characteristics	AppAware: application-aware VM migration [106]	Normal distribution-based load characteristics for VMs and servers, and inter-VM traffic demands	Reduction in network traffic and average impact of migration
Communication Traffic Minimization with Power-aware VM Placement in Data Centers [132]	Homogeneous data center	N/A	Online and offline	Mathematical optimization (minimization)	CPU, memory, and network I/O	Server resource capacity	Minimization of communication traffic and power cost	Greedy heuristic	Simulation based on synthetic data center and load characteristics	Random placement, simple greedy, and First Fit (FF)	Workload traces from production data center	Total communication traffic and number of active servers

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Energy-aware Virtual Machine Placement in Data Centers [63]	Homogeneous data center	Tree, VL2, Fat-tree, BCube	Online	Mathematical optimization framework (Proportional Fairness and Convex Optimization)	CPU, memory, and storage	Server resource capacity and inter-VM bandwidth requirement	Reduction of data transmission and minimization of energy consumption of server and network device	Greedy approach	Simulation based on synthetic data center and load characteristics	Random placement, First Fit Decreasing (FFD), Grouping Genetic Algorithm (GGA), two-stage heuristic algorithm, and optimal placement	Normal distribution-based load characteristics for VMs and servers and inter-VM traffic demands	Objective function value, reduction rate of traffic volume, and number of used servers
Network-aware VM Placement and Migration Approach in Cloud Computing [100]	IaaS Compute and Storage Data Center	Federated and distributed Cloud data centers	Offline and online	Mathematical	CPU and memory	Server resource capacity constraints	Minimization of data transfer time consumption	Exhaustive search	Fixed simulation scenario implemented in CloudSim [21]	Default VM placement policy of CloudSim (namely VMSimpleAllocationPolicy)	Small-scale fixed valued workload data	Average task completion time
Coupled Placement in Modern Data Centers [72]	Heterogeneous data center	SAN data center using core-edge design pattern	Offline	Knapsack Problem and Stable Marriage Problem	CPU and storage	Server CPU and storage capacity	Minimization of total network cost across all application communication links	Individual and pairwise greedy heuristic, and iterative refinement-based heuristic	Heterogeneous SAN data center and synthetic workload-based simulation	Linear Programming-based optimal placement	Normal distribution-based compute and storage resource demands and network I/O rates	Defined network cost function value and placement computation time

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Network Topology-aware VM Cluster Placement in IaaS Clouds [51]	Homogeneous IaaS Cloud data center	PortLand	Offline	Graph search	CPU, memory, and storage	VM anti-colocation, server resource capacity constraints	Minimization of network utilization, low decision time overhead regardless of infrastructure size	Greedy approach, recursive, backtracking	Simulation (JgraphT library)	First Fit Decreasing (FFD)	Workflow structures (Pipeline, Data Aggregation, and Epigenomics)	Network utilization
Improving the Scalability of Data Center Networks with Traffic-aware VM Placement [85]	Homogeneous	Tree, VL2, Fat-tree, and BCube	Offline and online (periodic)	NP-hard Combinatorial optimization problem, instance of Quadratic Assignment Problem (QAP)	CPU and memory	Maximum placement of one VM per PM	Improvement of scalability by minimizing the aggregated traffic rates at each network switch	Greedy approximate heuristic employing divide-and-conquer strategy and minimum k-cut graph algorithm	Trace-driven simulation using global and partitioned traffic model, and hybrid traffic model	Local Optimal Pairwise Interchange (LOPI) [7] and Simulated Annealing (SA) [19]	Inter-VM traffic rates (aggregated incoming and outgoing) collected from production data centers	Objective value and CPU time

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Stable Network-aware VM Placement for Cloud Systems [16]	Homogeneous data center tree	Fat-tree and VL2	Offline	NP-hard Integer Quadratic Programming	CPU and memory	Server resource capacity constraints, physical link bandwidth capacity constraints	Minimization of the maximum ratio of the demand and capacity across all network cuts	Integer Programming techniques employing divide-and-concur strategy and greedy heuristics	Simulation-based using IBM ILOG CPLEX mixed integer mathematical solver	Random and optimal placement	Gaussian distribution based inter-VM and VM-gateway traffic demands, equal server resource capacity and VM resource demand	Worst case and average network cut load ratio, solving time, dropped packets, and packet delivery delay
Communication Aware and Energy-Efficient Scheduling for Parallel Applications in Virtualized Data Centers [114]	Homogeneous data center	Three-level Tree topology	Online	Simple peer-based inter-VM communication pattern	CPU, memory, and network I/O	Server resource capacity and inter-VM bandwidth requirement	Minimization of energy consumption by servers and network components, and average network utilization	Iterative greedy that ranks VMs based on the number of in/out traffic flow	Simulation-based (network and memory subsystem implemented on CloudSim [21])	Simple CPU utilization-based random VM placement	NPB parallel application benchmark used as HPC application	Uniformity of VM placement on servers, average utilization of network links, and application performance degradation

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Application Aware VM Placement in Data Centers [109]	Homogeneous data center	Tree, VL2, Fat-tree, and BCube	Online	Proportional Fairness and Convex Optimization	CPU, memory, and storage	Inter-VM bandwidth requirement and server resource capacity	Reduction of data transmission and energy consumption	Two-tier algorithm	Simulation based on synthetic data center and load characteristics	Random placement and First Fit Decreasing (FFD)	Normal distribution-based load characteristics for VMs and servers, and inter-VM traffic demands	Objective function value and reduction rate of traffic volume
Application Aware VM Migration in Data Centers [106]	Homogeneous data center	Tree and VL2	Online	Mathematical optimization, multiple knapsack problem	CPU, memory, and storage	Server resource capacity	Minimization of network overhead due to VM migration	Greedy heuristic (exhaustive)	Simulation based on synthetic data center and load characteristics	Optimal placement (CPLEX solver) and Sandpiper VM migration scheme [127]	Normal distribution-based server resource and VM demands and, normal, exponential, and uniform distribution-based inter-VM traffic demands	Data center traffic reduction and objective function value

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Autonomic Virtual Resource Management for Service Hosting Platforms [95]	Heterogeneous	N/A	Offline	Knapsack Problem, Constraint Satisfaction Problem	CPU and memory	Server resource capacity	Optimization of global utility function through maximization of SLA fulfillment and minimization of operating costs (number of servers)	Time-bound exhaustive search using Constraint Programming	Simulation based on Choco constraint solver [66]	N/A	Synthetic Web workload distributed in a round-robin fashion	Response time, Utility value, CPU utilization, and number of requests and active hosts
EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments [77]	Homogeneous system	high-speed LAN	Offline	Bin Packing Problem	CPU and main memory	Server resource capacity constraints	Energy-aware application scheduling and placement	Greedy heuristic	EnaCloud framework implemented in iVIC[2] virtual computing environment and running on Cloud server pool powered by Xen [9]	First Fit (FF) and Best Fit (BF) algorithms	Randomly generated workloads for Web and database servers, compute-intensive applications, and common applications	Number of active servers, energy consumption, resource utilization, and VM migration times

Continued on next page

[2]http://www.ivic.org.cn

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
pMapper: Power and Migration Cost-aware Application Placement in Virtualized Systems [121]	Heterogeneous system	N/A	Offline	Bin Packing Problem	CPU	Server resource capacity constraints	Power consumption minimization under performance constraints	Greedy heuristic based on modified First Fit Decreasing (FFD)	Simulation based on pMapper framework running on VMWare ESX-based testbed	Comparison among various proposed algorithms	Server utilization trace data from server farm	Power and energy consumption, migration cost, power savings and penalty
Entropy: A Consolidation Manager for Clusters [60]	Heterogeneous	N/A	Offline	2-dimensional Bin Packing, Knapsack Problem, and Constraint Satisfaction Problem	CPU and main memory	Server resource capacity constraint	Dynamic VM consolidation with goal of minimizing of the number of active servers and VM migrations	Exhaustive search (depth first) based on Constraint Programming	Simulation and Xen-powered Grid'5000 experimental testbed	First Fit Decreasing (FFD)	Randomly generated synthetic data and NASGrid benchmark data	Percentage of minimized configurations, reconfiguration cost, number of satisfied VMs vs. number of unsatisfied VMs, number of unused nodes, and run-time.

Continued on next page

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Performance and Power Management for Cloud Infrastructures [117]	Homogeneous	N/A	Offline	Constraint Satisfaction Problem	CPU and main memory	Server resource capacity	Utility maximization and energy cost minimization	Choco constraint programming solver-based [66] brute-force search	Xen-powered testbed running cluster of Apache servers and farm of rendering applications controlled by Condor Grid scheduler	N/A	Web workload generated by CLIF[3] load injector and batch workload using constant stream of jobs	Response time, Utility value, and number of requests, allocated VMs, and active hosts
A Multi-objective Ant Colony System Algorithm for Virtual Machine Placement in Cloud Computing [50]	Heterogeneous	N/A	Offline	Multi-objective optimization and Multi-dimensional Vector Packing Problem	CPU and memory	Server resource capacity constraints	Minimization of power consumption and server resource wastage	ACO-based [36] algorithm	Ad hoc simulation toolkit	FFD, SACO [46], and MGGA [129]	Randomly generated VM resource demands	Power consumption, resource wastage, non-dominated vector generation, spacing, and run-time

Continued on next page

[3]CLIF is a Load Injection Framework `http://clif.ow2.org/`

Table 3.1 – *Continued from previous page*

Research Project	System Assumption	Network Topology	Placement Type	Modeling Technique	Physical Resource	Placement Constraint	Objective or Goal	Solution Approach	Evaluation Platform	Compared Approaches	Workloads	Performance Metric
Energy-aware Ant Colony-based Work-load Placement in Clouds [46]	Homogeneous servers	N/A	Online	Multi-dimensional Bin Packing Problem	CPU, memory, network I/O, and storage	Server resource capacity constraints	Minimization of number of number of active servers and energy consumption	Max-Min Ant System-based [111] algorithm	FFD, CPLEX[4]	Ad hoc simulation toolkit	Randomly generated VM resource demands	Number of active servers, energy consumption, run-time, and number of utilized hosts
A Case for Fully Decentralized Dynamic VM Consolidation in Clouds [45]	Heterogeneous system	Unstructured Peer-to-peer (P2P) network of server	Offline	Mathematical framework on minimization function	CPU, main memory, and network I/O	Server resource capacity constraints	Maximization of VM packing efficiency and scalability, and minimization of VM migrations	Cyclon membership protocol-based [125] decentralized VM consolidation schema and Max-Min Ant System-based [111] VM consolidation algorithm	Python-based Cyclon P2P system emulator running on Grid'5000 experimental testbed	FFD, V-MAN [82], and Sercon [90]	Randomly generated VM resource demands	Number of active and released servers, number of VM migrations, number of migrations per VM, and VM packing efficiency

Continued on next page

[4]CPLEX Optimizer `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

Table 3.1 – *Continued from previous page*

Research Project	**System Assumption**	**Network Topology**	**Placement Type**	**Modeling Technique**	**Physical Resource**	**Placement Constraint**	**Objective or Goal**	**Solution Approach**	**Evaluation Platform**	**Compared Approaches**	**Workloads**	**Performance Metric**
Adaptive Threshold-based Approach for Energy-efficient Consolidation of Virtual Machines in Cloud Data Centers [10]	Heterogeneous servers	N/A	Online	Bin Packing Problem	CPU, main memory, network I/O, and storage	Server resource capacity	Threshold-based dynamic VM consolidation, minimization of energy consumption, SLA, and number of VM migration	Greedy Approach, Modified Best Fit Decreasing algorithm	Simulation based on CloudSim [21]	Comparison among various proposed algorithms	CPU utilization data from CoMon monitoring project for PlanetLab	Energy consumption, SLA violation, number of VM migrations
SLA-Aware Virtual Resource Management for Cloud Infrastructures [116]	Homogeneous	N/A	Offline	Multiple Knapsack Problem and Constraint Satisfaction Problem	CPU and main memory	Server resource	Autonomic dynamic VM provisioning and placement with utility maximization, active server minimization, and reduction of reconfiguration cost due to VM live migrations	Self-optimization through the combination of utility functions and constraint programming	Simulation environment based on four servers running cluster of Web servers and multiplayer online game and the Choco constraint solver [66]	N/A	Synthetic workload distributed in a round-robin algorithm	Number of requests, CPU time, global utility, and response time

3.3 A Survey on the State-of-the-art Virtual Machine Management Strategies

This section presents a thorough review, analysis, and remarks on some of the recent notable research works on VM management in the following subsections. Table 3.2 illustrates the most significant aspects of the reviewed research projects that are highly relevant to the VM placement, migration, and consolidation strategies.

3.3.1 Traffic-aware VM Placement and Migration Techniques

Network Topology-aware VM Cluster Placement in IaaS Clouds

Georgiou *et al.* [51] have investigated the benefits of user-provided hints regarding inter-VM communication patterns and bandwidth demands during the actual VM placement decision phase. The authors have proposed two offline VM-cluster placement algorithms with the objective to minimize the network utilization at physical layer, provided that the physical server resource capacity constraints are met. VM deployment request is modeled as Virtual Infrastructure (VI) with specification of the number and resource configuration (CPU core, memory, and storage demands) of VMs, bandwidth demands of inter-VM communication within the VI, modeled as Virtual Links (VLs), as well as possible anti-colocation constraint for pairs of VMs. The underlying physical infrastructure is modeled as a homogeneous cluster of servers organized according to the PortLand [91] network architecture. The authors have argued that conventional tree-like network topologies often suffer from over-subscription and network resource contention primarily at the core top-levels, leading to bottlenecks and delays in services, PortLand network architecture can play a significant role in effective management of computational resources and network bandwidth in Cloud data centers.

The authors have also presented a framework comprising of two layers: physical infrastructure consisting of homogeneous servers organized as PortLand network topology and a middleware on top of the infrastructure. The middleware layer is composed of the following two main components: Planner and Deployer. As input, the Planner gets VM deployment request as VI specification (in XML format), and possible suggestions regarding desired features in VI from user as well as the current resource state information of the infrastructure layer, executes the VM placement algorithms to determine the VM-to-PM and VL-to-physical link mappings, and finally passes over the placement decision to the

Deployer. The Deployer can be a third-party provided component that takes care of the VMs deployment on the physical layer components.

With the goal of minimizing network utilization of the physical layer during the VI deployment decision, the authors have proposed two algorithms based on greedy approach. The first algorithm, Virtual Infrastructure Opportunistic fit (VIO) tries to place the communicating VMs near to each other in the physical network. Starting with a sorted list of VLs (in decreasing order of their bandwidth demands) connecting the VMs, the VIO picks up the front VL from the list and attempts to place the VMs connected by the VL in the nearest possible physical nodes (preferably in the same node when anti-colocation is not set), provided that physical node resource capacity constraints, network link bandwidth capacity constraints, as well as user provided constraints are met. In case VIO reaches a dead-end state where the VL at hand cannot be placed on any physical link, VIO employs a backtracking process where VLs and corresponding VMs are reverted back to unassigned state. Such VL placement inability can occur due to three reasons: (1) No physical node with enough resource is found to host a VM of the VL, (2) No physical path with enough bandwidth is found to be allocated for the VL, and (3) Anti-colocation constraint is violated.

Backtracking process involves de-allocation of both server resource and network bandwidth of physical links. In order to limit the number of reverts for a VL and terminate the algorithm with a reasonable amount of time, a revert counter is set for each VL. When the maximum amount of reverts has been reached for a VL, the VI placement request is rejected and the VIO terminates gracefully. The second algorithm, Vicinity-BasEd Search (VIBES) based on the PortLand network architecture characteristics, tries to detect an appropriate PortLand neighborhood to accommodate all the VMs and VLs of the requested VI, and afterward applies VIO within this neighborhood. In order to identify fitting neighborhood, VIBES exploits PortLand's architectural feature of pods (cluster of physical nodes under the same edge-level switch). The authors also presented formula for ranking all neighborhoods based on the available resources in the servers and bandwidth of the physical links within each neighborhood. VIBES starts with the pod with the most available resources and invokes VIO. Upon rejection from VIO, VIBES expands the neighborhood further by progressively merging the next most available pod to the set of already selected pods. The search for a large enough neighborhood proceeds until a

neighborhood with enough available resources is found or the search window is growing beyond a customizable maximize size in which case the VI placement request is rejected.

Performance evaluation of VIO and VIBES is conducted through simulation of physical infrastructures and compared against network-agnostic First Fit Decreasing (FFD) algorithm. Online VI deployment and removal is simulated using three different data flow topologies: Pipeline, Data Aggregation, and Epigenomics [13]. The simulation results show that the proposed algorithms outperforms FFD with respect to network usage: VIO trims down the network traffic routed through the top-layer core switches in the PortLand architecture by up to 75% and incorporation of VIBES attains a further 20% improvement. The authors have also suggested future research directions such as optimization of the power usage of network switches through exploitation of reduced network utilization, testing VIO and VIBES for other network topologies such as BCube [54] and VL2 [52].

Stable Network-aware VM Placement for Cloud Systems

With focus on communication pattern and dynamic traffic variations of modern Cloud applications, as well as non-trivial data center network topologies, Biran *et al.* [16] have addressed the problem of VM placement with the objective to minimize the maximum ratio of bandwidth demand and capacity across all network cuts and thus maximize unused capacity of network links to accommodate sudden traffic bursts. The authors have identified several important observations regarding network traffic and architectures: (1) due to several factors such as time-of-day effects and periodic service load spikes, run-time traffic patterns undergo high degree of variations, and (2) modern data centers are architected following non-trivial topologies (e.g., Fat-tree [5] and VL2 [52]) and employ various adaptations of dynamic multi-path routing protocols.

Considering the above-mentioned points, the authors presented two VM placement algorithms that strive to satisfy the forecasted communication requirements as well as be resistant to dynamic traffic variations. The authors have introduced the Min Cut Ratio-aware VM Placement (MCRVMP) problem and formally formulated using the Integer Quadratic Programming model considering both the server side resource capacity constraints and network resource constraints evolving from complex network topologies and dynamic routing schemata. Since the MCRVMP problem definition works only for tree topology, the authors have also proposed graph transformation techniques so that MCRVMP can be applied to other complex network topologies, for example VL2 and

Fat-tree. Considering the fact the MCRVMP is a NP-hard problem, the authors have proposed two separate heuristic algorithms for solving the placement problem and compared these against optimal and random placements. Both the proposed VM placement heuristic algorithms utilize the concept of Connected Components (CCs) of the running VMs in the data center. Such a CC is formed by the VMs that exchange data only between themselves or with the external gateway (e.g., VMs comprising a multi-tier application) and thus clustering VMs in this way helps minimize the complexity of the problem. First algorithm, termed 2-Phase Connected Component-based Recursive Split (2PCCRS) is a recursive, integer programming technique-based algorithm that utilizes the tree network topology to define and solve small problem instances on one-level trees. By adopting a two-phase approach, 2PCCRS places the CCs in the network and then expands them to place the actual VMs on the servers. Thus, 2PCCRS reduces the larger MCRVMP problem into smaller sub-problems and solves them using mixed integer programming solver in both the phases. Second algorithm, called Greedy Heuristic (GH) entirely avoids using mathematical programming and greedy places each VM individually. Similar to 2PCCRS, GH works in two phases. In the first phase, GH sorts all the traffic demands in decreasing values and sorts all CCs in decreasing order based on the accumulated traffic demands among the VMs within a CC. In the second phase, GH iteratively processes the ordered traffic demands by placing each VM on the physical server that results in minimum value of the maximum cut load values.

The efficiency of the proposed algorithms is evaluated in two phases. In the first phase, 2PCCRS and GH algorithms were compared to random and optimal placement approaches with focus on placement quality in terms of worst and average cut load ratio and solution computation time. As reported by the authors, for small problem instances both 2PCCRS and GH reach worst case and average cut load ratio very close to optimal algorithm with nearly zero solving time; whereas for larger problem sizes, 2PCCRS significantly outperforms GH, while requiring much higher solving time due its use of mathematical programming techniques. In the second phase, the authors have validated the resilience of MCRVMP-based placements under time-varying traffic demands with NS2-based simulations focusing on the percentage of dropped packets and average packet delivery delay. Simulation results show that with no dropped packets, both 2PCCRS and GH can absorb traffic demands up to three times the nominal values. Furthermore, placements produced

by the 2PCCRS algorithm have average packet delivery delays lower than GH-based ones due to the less loaded network cuts.

The authors have also remarked that the proposed MCRVMP problem formulation is not meant for online VM placement where new VM requests are served for data center having already placed VMs. In addition, the authors have ignored the potential VM migration costs entirely. As per future works, the authors have indicated potential extension of MCRVMP by incorporating traffic demand correlation among VMs to further cut down the amount of dropped packets and by preventing MCRCMP to produce solutions with very high local compute-resource overhead due to inter-memory communications.

Scalability Improvement of Data Center Networks with Traffic-aware VM Placement

Meng *et al.* [85] have addressed the scalability problem of modern data center networks and proposed solution approaches through optimization of VM placement on physical servers. Different from existing solutions that suggest changing of network architecture and routing protocols, the authors have argued that scalability of network infrastructures can be improved by reducing the network distance of communicating VMs. In order to observe the dominant trend of data center traffic-patterns, the authors have claimed to have conducted a measurement study in operational data centers resulting with the following insights:

1. There exists low correlation between average pairwise traffic rate and the end-to-end communication cost,

2. Highly uneven traffic distribution for individual VMs, and

3. VM pairs with relatively heavier traffic rate tend to constantly exhibit the higher rate and VM pairs with low traffic rate tend to exhibit the low rate.

The authors have formally defined the Traffic-aware VM Placement Problem (TVMPP) as a combinatorial optimization problem belonging to the family of Quadratic Assignment Problems [80] and proved its computational complexity to be NP-hard. TVMPP takes the traffic matrix among VMs and communication cost matrix among physical servers as input, and its optimal solution would produce VM-to-PM mappings that would result in minimum aggregate traffic rates at each network switch. The cost between any two

communicating VMs is defined as the number of switches or hops on the routing path of the VM pair. The authors have also introduced a concept of slot to refer to one CPU/memory allocation on physical server where multiple such slots can reside on the same server and each slot can be allocated to any VM.

Since TVMPP is NP-hard and existing exact solutions cannot scale to the size of current data centers, the authors have proposed two-tier approximate algorithm Cluster-and-Cut based on two design principles:

1. Finding solution of TVMPP is equivalent to finding VM-to-PM mappings such that VM pairs with high mutual traffic are placed on PM pairs with low-cost physical links and

2. Application of the divide-and-conquer strategy.

The Cluster-and-Cut heuristic is composed of two major components: SlotClustering and VMMinKcut. SlotClustering partitions a total of n slots in the data center into k clusters using the cost between slots as the partition criterion. This component produces a set of slot-clusters sorted in decreasing order of their total outgoing and incoming cost. The VMMinKcut partitions a total of n VMs into k VM-clusters such that VM pairs with high mutual traffic rate are placed within the same VM-cluster and inter-cluster traffic is minimized. This component uses the minimum k-cut graph algorithm [105] partition method and produces k clusters with the same set of size as the previous k slot-clusters. Afterwards, Cluster-and-Cut maps each VM-cluster to a slot-cluster and for each VM-cluster and slot-cluster pair, it maps VMs to slots by solving the much smaller sized TVMPP problem. Furthermore, the authors have shown that the computational complexity of SlotClustering and VMMinKcut are $O(nk)$ and $O(n^4)$, respectively, with total complexity of $O(n^4)$.

The performances evaluation of Cluster-and-Cut heuristic is performed through trace-driven simulation using hybrid traffic model on inter-VM traffic rates (aggregated incoming and outgoing) collected from production data centers. The results show that Cluster-and-Cut produces solution with objective function value 10% lower than its competitors across different network topologies and the solution computation time is halved.

However, the proposed approach considers some assumptions that cannot be hold in the context of real data centers. TVMPP does not incorporate the link capacity constraints

that can lead to VM placement decisions with congested links into the data center [16]. Furthermore, Cluster-and-Cut algorithm places only one VM per server that can result in high amount of resource wastage. Additionally, it is assumed that static layer 2 and 3 routing protocols are deployed in the data center. Finally, VM migration overhead incurred due to the offline VM shuffling is not considered.

Through discussion the authors have indicated the potential benefit of combining the goals of both network resource optimization and server resource optimization (such as power consumption or CPU utilization) during the VM placement decision phase. They also emphasized that reduction of total energy consumption in a data center requires combined optimization of the above-mentioned resources. The authors have also mentioned potential of performance improvement by employing dynamic routing and VM migration, rather than using simple static routing.

3.3.2 Network-aware Energy-efficient VM Placement and Migration Approaches

Multi-objective Virtual Machine Migration in Virtualized Data Center Environments

Huang *et al.* [62] have addressed the problem of overloaded VM migration in data centers having inter-VM communication dependencies. Indicating the fact that most of the existing works on VM migrations focus primarily on the server-side resource constraints with the goal of consolidating VMs on minimum number of servers and thus improving overall resource utilization and reducing energy-consumption, the authors have argued that VMs of modern applications have mutual communication dependencies and traffic patterns. As a result, online VM migration strategies need to be multi-objective focusing both on maximizing resource utilization and minimizing data center traffic overhead. Following a similar approach as in [63], the authors have presented three stages of the joint optimization framework:

1. Based on the dominant resource share and max-min fairness model, the first optimization framework tries to maximize the total utilities of the physical servers; in order words tries to minimize the number of used servers and thus reduce power consumption,

2. Considering the complete application context with inter-VM traffic dependencies, the second optimization framework strives to minimize the total communication costs among VM after necessary VM migrations, and

3. Based on the above two frameworks, the third optimization framework combines the above goals subject to the constraints that the allocated resources from each server is not exceeded its capacity and the aggregated communication weight of a server is lower or equal to its bandwidth capacity.

The authors have further proposed a two-stage greedy heuristic algorithm to solve the defined optimization problem: Base Algorithm and Extension Algorithm. The Base Algorithm takes as input the set of VMs, set of servers, and the dominant resource share of user servers, and the set of overloaded VMs. Then, it sorts the overloaded VMs in decreasing order of their dominant resource share before migration. After incorporation of application dependencies (i.e., inter-VM communication dependencies), the Extension Algorithm selects candidate destination server for migration to the server with the minimum dominant resource share and application-dependent inter-VM traffic. The VM migration effect is computed as the impact based on both distance effect and inter-VM traffic pattern-based network cost after migration. For each overloaded VM, the total communication weight is computed as the sum of all related inter-VM communication weights and the overloaded VM is migrated to the server with minimum migration impact.

The authors have shown simulation-based evaluation of the proposed multi-objective VM placement approach with comparison to AppAware [106] application-aware VM migration policy. The following four different network topologies are used as data center network architecture: Tree, Fat-Tree [5], VL2 [52], and BCube [54]. Data center server capacity, VM resource demand, and inter-VM traffic volume is generated synthetically based on normal distribution with varying mean. The results show that the achieved mean reduction in traffic of the proposed algorithm is higher for BCube compared to Tree topology. Compared to AppAware, the proposed algorithm can achieve larger reduction in data center network traffic volume, by generating migrations that decreased traffic volume transported by the network up to 82.6% (for small number of VMs). As per average impact of migration, it decreases with the increase of server resource capacity. It is attributed that since the multiplier factor in the migration impact formulation includes dominant resource share of the migrating VM and it is decreased after migration. However, with the increase

of VM resource demands, the average impact of migration is increased. This is attributed for the fact that the demand of VMs has a direct impact on the inter-dependencies among the VMs of multi-tier applications. Finally, with the increase of inter-VM communication weights, the average impact of migration increases since communication weights influence the cross-traffic burden between network switches.

Communication Traffic Minimization with Power-aware VM Placement in Data Centers

Zhang *et al.* [132] have addressed the problem of static greedy allocations of resources to VMs, regardless of the footprints of resource usage of both VMs and PMs. The authors have suggested that VMs with high communication traffic can be consolidated into minimum number of servers so as to reduce the external traffic of the host since co-located VMs can communicate using memory copy. With goal of minimizing communication traffic within a data center, the authors have defined dynamic VM placement as an optimization problem. The solution of the problem would be a mapping between VMs and servers, and such a problem is presented to be reduced from a minimum k-cut problem [130] that is already proved to be NP-hard. Since an idle server uses more than two-third of the total power when the machine is fully utilized [73], the authors set power-consumption minimization as a second objective of their proposed VM placement scheme.

The authors have provided formal presentation of the optimization problem using mathematical framework that is set to minimize the total communication traffic in the data center, provided that various server-side resource constraints should be satisfied. Such problem can be solved by partitioning the VMs into clusters in such a way that VMs with heavy communication can be placed in the same server. As a solution, the author proposed the use of K-means clustering algorithm [130] that would generate VM-to-server placement mappings. Utilizing the K-means clustering approach, the authors proposed a greedy heuristic named K-means Clustering for VM consolidation that starts by considering each server as a cluster. Such cluster definition has got some benefits:

1. The number K and the initial clusters can be fixed to minimize the negative impact from randomization,

2. There is an upper-bound for each cluster that corresponds to the capacity constraints of each server, and

3. Fixed clusters can reduce the number of migrations.

In each iteration of the K-means Clustering for the VM Consolidation algorithm, the distance between a selected VM and a server is determined. Using this, the VM is placed in the server with minimum distance. This step is repeated until every VM has a fixed placement on its destination server. The authors have further reported that the greedy algorithm has a polynomial complexity of $O(tmn)$, where t is the number of iterations, n is the number of VMs, and m is the number of servers in the data centers. The authors have further presented algorithms for computing the distance between a VM and a cluster, and for online scenarios where greedy heuristic handles new VM requests.

Performance evaluation based on simulation and synthetic data center load characteristics is reported with superior performance gain by the proposed algorithm compared to its three competitors: (1) random placement, (2) simple greedy approach (puts the VM on the server which communicates most with current VM), and (3) First Fit (FF) heuristic. Both the random placement and FF heuristics are unaware of inter-VM communication. The results show that the proposed greedy algorithm achieved better performance for both performance metrics: (1) total communication traffic in data center and (2) number of used server (in other words, measure of power cost) after consolidation. For the online VM deployment scenario, the clustering algorithm is compared against greedy algorithm and it is reported that the greedy algorithm can perform very close to the clustering method where the number of migrations is significantly larger than the greedy method and the greedy method can deploy new VM requests rapidly without affecting other nodes.

As for future work directions, the authors expressed plan to introduce the SLA to approach a better solution where the data center can provide better performance for the applications because of less communication traffic. This metric would be included in the cost model. Furthermore, the migration cost would be taken as a metric of the proposed distance model.

Energy-aware Virtual Machine Placement in Data Centers

Huang *et al.* [63] have presented a joint physical server and network device energy consumption problem for modern data centers hosting communication-intensive applications. The authors have staged several data center facts in order to signify the importance of multi-objective VM placement:

1. Increasing deployment of wide spectrum of composite applications consisting of multiple VMs with large amount of inter-VM data transfers,

2. Continuous growth in the size of data centers,

3. Existing VM placement strategies lack multiple optimizations, and

4. Rise of electricity cost.

In response to the above issues, the authors have investigated the balance between server energy consumption and energy consumption of data center transmission and switching network.

The multi-objective VM placement problem is modeled as an optimization problem in three stages. Considering server resource capacities (CPU, memory, and storage) and VM resource demands, the first optimization framework is targeted on VM placement decisions that would maximize server resource utilizations and eventually reduce energy consumption (by turning idle servers to lower power state, e.g., standby) following proportional fairness and without considering inter-VM communication pattern. The second optimization framework considers inter-VM data traffic patterns and server-side bandwidth capacity constraints, and is modeled as a Convex Programming Problem that tries to minimize the total aggregated communication costs among VMs. Finally, the energy-aware joint VM placement problem is modeled using fuzzy-logic system with trade-off between the first two objectives that can be in conflict when combined together. The authors have further proposed a prototype implementation approach for the joint VM placement following a two-level control architecture with local controllers installed in every VM and a global controller at the data center level responsible to determining VM placement and resource allocations.

As solution approach, the authors have put forward two algorithmic steps: VMGrouping and SlotGrouping. VMGrouping finds VM-to-server mappings such that VM pairs with high traffic communication are mapped to server pairs with low cost physical link. Such VM-to-server mappings are modeled as Balanced Minimum K-cut Problem [105] and a k-cut with minimum weight is identified so that the VMs can be partitioned into k disjoint subsets of different sizes. Afterwards, SlotGrouping maps each VM group to appropriate servers in closest neighborhood respecting the server side resource constraints.

The authors have validated the proposed multi-objective VM placement approach using simulation-based evaluation under varying traffic demands, and load characteristics of VMs and physical servers using normal distribution under different means as well as for different network architectures, such as Tree [5], VL2 [52], Dcell [55], and BCube [54]. Focusing on the formulated objective function value and total data center traffic volume as performance metrics, the proposed joint VM placement policy is compared against random placement and First Fit Decreasing (FFD) heuristic-based placement policies. The results show that the joint VM placement achieves higher objective values and much reduced traffic flow (up to 50% to 81%) compared to other approaches, resulting in lower communication cost and resource wastage. In order to assess performance from energy-consumption reduction point of view, the proposed placement approach is compared against Grouping Genetic Algorithm (GGA) [2], FFD, two-stage heuristic algorithm [56], random placement, and optimal placement considering the number of used PMs as performance metric. It is reported that the proposed energy-aware joint placement method achieves better performance over random placement, GGA, and the two stage heuristic algorithm, and inferior performance over FFD and optimal placement. Such performance pattern is rationalized by the trade-offs between multiple objectives (i.e., minimizing both resource wastage and traffic volume simultaneously) that the joint VM placement policy strives to achieve.

In this research work, the authors have brought about a very timely issue of balancing both energy- and network-awareness while VM placement decisions are made. Most of the existing works focus on either one of the objectives, not both at the same time. However, this work has not considered the impact of the necessary VM live migrations and reconfiguration on both the network links and hosted applications performance, which can have substantially detrimental effects on both applications SLAs and network performance given that the new VM placement decision requires large number of VM migrations.

3.3.3 Network- and Data-aware VM Placement and Migration Mechanisms

Coupled Placement in Modern Data Centers

Korupolu *et al.* [72] have addressed the problem of placing both computation and data components of applications among the physical compute and storage nodes of modern virtualized data centers. The authors have presented several aspects that introduce heterogeneity in modern data centers and thus make the optimization problem of compute-data pairwise placement non-trivial:

1. Enterprise data centers evolve over time and different parts of the data center can have performance variations (e.g., one network switch can be more recent than others and have lower latency and greater I/O throughput),

2. Wide spread use of multi-purpose hardware devices (e.g., storage devices with built-in compute resources), and

3. Large variance of the I/O rates between compute and data components of modern applications.

Taking into considering the above factors, the Coupled Placement Problem (CPP) is formally defined as an optimization problem with the goal of minimizing the total cost over all applications, provided that compute server and storage node capacity constraints are satisfied. The cost function can be any user defined function and the idea behind it is that it captures the network cost that is incurred due placing the application computation component (e.g., VM) in a certain compute node and the data component (e.g., data block or file system) in a certain storage node. One obvious cost function can be the I/O rate between compute and data components of application multiplied by the corresponding network distance between the compute and storage nodes.

After proving the CPP as a NP-hard problem, the authors proposed three different heuristic algorithms to solve it:

1. Individual Greedy Placement (INDV-GR), following greedy approach, tries to place the application data storage sorted by their I/O rate per unit of data storage where storage nodes are ordered by the minimum distances to any connected compute node. Thus, INDV-GR algorithm places highest throughput applications on storage nodes having the closest compute nodes.

2. Another greedy algorithm, Pairwise Greedy Placement (PAIR-GR) considers the compute-storage node affinities and tries to place both compute and data components of each application simultaneously by assigning applications sorted by their I/O rate normalized by their CPU and data storage requirements on storage-compute node pairs sorted by the network distance between the node pairs.

3. Finally, in order to avoid early sub-optimal placement decisions resulting due to the greedy nature of the first two algorithms, the authors proposed Coupled Placement Algorithm (CPA) where CPP is shown to have properties very similar to the Knapsack Problem [101] and the Stable-Marriage Problem [83]. Solving both the Knapsack and the Stable-Marriage Problem, the CPA algorithm iteratively refines placement decisions to solve the CPP problem in three phases:

   (a) CPA-Stg phase where data storage placement decision is made,

   (b) CPA-Compute phase where computation component placement decision is taken provided the current storage placements, and

   (c) CPA-Swap phase that looks for pairs of applications for which swapping their storage-compute node pairs improves the cost function and performs the swap.

The performance of INDV-GR, PAIR-GR, and CPA is compared against the optimal solutions through simulation-based experimentation. The authors have used CPLEX ILP solver for small problem instances and MINOS solver based on LP-relaxation for larger problems. Cost function values and placement computation times are considered as performance metrics and the experiments are carried out across four different dimensions:

1. Problem size/complexity through variations in simulated data center size,

2. Tightness of fit through variations of mean application compute and data demands,

3. Variance of application compute and data demands, and

4. Physical network link distance factor.

Through elaborate analysis of results and discussion, the proposed CPA algorithm is demonstrated to be scalable both in optimization quality and placement computation time, as well as robust with varying workload characteristics. On average, CPA is shown to produce placements within 4% of the optimal lower bounds obtained by LP formulations.

However, the optimization framework takes some simplistic view of the application models and resource capacity constraints. Firstly, the CPP has considered each application as having one compute and one data storage components whereas modern applications usually have composite view with multiple compute components with communications among themselves as well as communication with multiple data storage components. Secondly, on the part of compute resource demand, only CPU is considered whereas memory and other OS-dependent features make the problem multi-dimensional [48]. Thirdly, no assumption is made regarding the overhead or cost of reconfiguration due to the new placement decision, in which VM migrations and data movement would be dominating factors. Finally, no network link bandwidth capacity constraint is no taken into account during the CPP formulation. Nonetheless, the authors have pointed out couple of future research outlooks: inclusion of multi-dimensional resource capacity constraints and other cost models focusing on different data center objectives like and energy utilization.

Network- and Data Location-aware VM Placement and Migration Approach in Cloud Computing

Piao *et al.* [100] have addressed the problem of achieving and maintaining expected performance level of data-intensive Cloud applications that need frequent data transmission from storage blocks. The studied scenario is focused on modern Cloud data centers comprising of both compute Clouds (e.g., Amazon EC2) and storage Clouds (e.g., Amazon S3) where hosted applications access the associated data across the Internet or Intranet over communication links that can either be physical or logical. Moreover, the authors have suggested that under current VM allocation policy, the data can be stored arbitrarily and distributed across single storage Cloud or even over several storage Clouds. Furthermore, the brokers allocate the applications without consideration of the data access time. As a consequence, such placement decisions can lead to data access over unnecessary distance.

In order to overcome the above-mentioned problem, the authors have proposed two algorithms based on exhaustive search: VM placement approach and VM migration approach. For both the solutions, the per application data is modeled as a set of data blocks distributed across different physical storage nodes with varying distances (either logical or physical) from physical compute nodes. Network speed between physical compute node and storage node is modeled using $Speed(s, \Delta t)$ function that depends on the size of the

data s and packet transfer time slot Δt. Finally, for each physical compute node, the corresponding data access time is formulated as the sum of product of each data block size and the inverse of the corresponding network speed value. The VM placement algorithm handles each new application deployment request and performs an exhaustive search over all the feasible compute nodes to find the one with minimum data access time for the corresponding data blocks for the submitted VM, subject to the compute node resource capacity constraints are satisfied. The VM migration algorithm is triggered when the application execution time exceeds the SLA specified threshold. In such a situation, a similar exhaustive search over all the feasible compute nodes is performed to find the one with minimum data access time for the corresponding data blocks for the migrating VM, subject to the compute node resource capacity constraints as satisfied.

The efficacy of the proposed algorithms is validated through simulation based on the CloudSim [21] simulation toolkit. The evaluation is focused on the average task completion time and the proposed algorithms are compared against the default VM placement policy implemented in CloudSim 2.0, namely VMAllocationPolicySimple that allocates the VM on the least utilized host following a load-balancing approach. The simulation is setup with small scale data centers comprising of 3 VMs, 3 data blocks, 2 storage nodes, and 3 compute nodes with fixed resource capacities. It is shown that the proposed approaches needed shorter average task completion time, which is emphasized as due to the optimized location of hosted VMs. In order to trigger the proposed VM migration algorithm, the network status matrix is changed and as a consequence some of the VMs are migrated to hosts that resulted in lower average task completion time.

Besides considering very simplistic view of federated Cloud data centers, the proposed VM placement and migration algorithms take an exhaustive search approach that may not scale for very large data centers. Moreover, the experimental evaluation is performed in a tiny scale and compared with a VM placement that is fully network-agnostic. Furthermore, VM migration or reconfiguration overhead is not considered in the problem formulation or solution schemes.

As for future work directions, the authors suggested inclusion of negotiation between service provider and user in terms of data access time to guarantee SLA enforcement. In order to avoid some users' tasks always occupying a faster network link, priority-based scheduling policy is recommended through extension of the payment mechanisms.

3.3.4 Application-aware VM Placement and Migration Strategies

Communication-aware Scheduling for Parallel Applications in Virtualized Data Centers

Takouna *et al.* [114] have introduced the problem of scheduling VMs that are part of HPC applications and communicate through shared memory bus (when placed in the same server) and shared networks (when placed in different servers). The authors have identified some limitations of existing VM placement and migration approaches with regards to the HPC and parallel applications:

1. VM placement approaches that optimize server-side resources (e.g., CPU and memory) are unaware of the inter-VM communication patterns, and as a result are less efficient from network utilization and ultimately from application performance point of view, and

2. Recent network-aware VM placement approaches focus on optimal initial VM placement and overlook the real-time communication patterns and traffic demands, and thus are not reactive to changes.

In order to address the above shortcomings, the authors have proposed communication-aware and energy-efficient VM scheduling technique focusing on parallel applications that use different programming models for inter-VM communication (e.g. OpenMP and Message Passing Interface (MPI)). The proposed technique determines the run-time inter-VM bandwidth requirements and communication patterns and upon detection of inefficient placement, reschedules the VM placement through VM live migrations.

In order to handle potential VM migration requests, the authors have presented a brief overview of the system framework consisting of VMs with peer-VM information (i.e., VMs that have mutual communication) and a central Migration Manager (MM). HPC jobs are executed in individual VMs and each VM have a list of its peer-VMs at run-time. It is the responsibility of the MM to determine the communication pattern of the whole parallel application. It is further assumed that each physical server have enough free resources (10% to 20% of CPU) to handle potential VM migration. The authors have further proposed an iterative greedy algorithm, namely Peer VMs Aggregation (PVA) that would be run by the MM upon getting migration requests from VMs. The ultimate goal of the PVA algorithm is to aggregate the communicating VMs with mutual traffic into

the same server so that they can communicate through the shared memory bus, so as to reduce the inter-VM traffic flow in the network. This would both localize the traffic (and thus reduce network utilization) and minimize the communication delays among VMs with mutual communication dependencies (and thus improving application performance). The PVA algorithm is composed of the following four parts:

1. *Sort:* The MM ranks the VMs that are requesting migration in a decreasing order based on the number of input/output traffic flows while ignoring the requests of VMs assigned on the same server),

2. *Select:* MM selects the highest ranked VM to be migrated to the destination server where its peer VMs are assigned,

3. *Check:* MM examines the feasibility of VM migrations to the destination servers in terms of server resource (CPU, memory, and network I/O) capacity constraints, and

4. *Migrate:* If MM finds the server suitable for the migrating VM, it directly migrates the selected VM to that server; otherwise the MM tries to migrate a VM from the destination server to free enough resources for the selected VM to be placed in the same server of its peer VMs (in that case the selected VM should also be suitable to be migrated). However, if the destination server does not host any VM, the MM can assign the selected VM on a server that shares the same edge switch with the server of its peer VMs.

The PVA approach is reported to minimize the total data center traffic significantly by reducing the network utilization traffic by 25%. The authors have claimed to have implemented the network topology and memory subsystem on the popular CloudSim simulation toolkit [21] and used the NAS Parallel Benchmarks (NPB) as HPC application which is divided into two groups: kernel benchmarks and pseudo-applications [113]. While compared to CPU utilization-based random placement algorithm, PVA is reported to have aggregated all the VMs belonging to an application into the same server and thus produced perfect VM placement after determining the traffic pattern of the communicating VMs. Moreover, the proposed approach have been shown to have outperformed the CPU-based placement in terms of reducing network link utilization through transferring inter-VM communication from shared network to shared memory by aggregating communicating VMs. In addition, the application performance degradation is computed and compared

against the ideal execution time of the individual jobs and it is reported that 18% of the VMs suffer performance degradation while using PVA, whereas 20% performance degradation is experienced in the case of CPU-based placements.

Though PVA approach mentions where to migrate a VM, it does not make it clear when a VM requests for migration. Moreover, the associated VM migration overhead is not taken into account. Furthermore, it would not be always the case that all the VMs consisting of a parallel/HPC application can be aggregated into a single server. Finally, the evaluation lacks the reporting of the energy-efficiency aspect of the proposed approach. The authors have presented a few future research work directions: (1) performance evaluation using different number of VMs for each application and (2) comparison with communication- and topology-aware VM placement approaches.

Application-aware VM Placement in Data Centers

Song *et al.* [109] have presented an application-aware VM placement problem focusing on energy-efficiency and scalability of modern data centers. The authors have pointed out several factors of modern data center management:

1. Increasing use of large-scale data processing services deployed in data centers,
2. Due to the rise of inter-VM bandwidth demands of modern applications, several recent network architecture scalability research works have been conducted with the goal of minimizing data center network costs by increasing the degree of network connectivity and adopting dynamic routing schemes,
3. Focusing on energy- and power-consumption minimization, several other recent works proposed mechanisms to improve server resource utilization and turning inactive servers to lower power states to save energy, and
4. Existing VM placement tools (e.g., VMware Capacity Planner[5] and Novell PlateSpin Recon[6]) are unaware of inter-VM traffic patterns, and thus can lead to placement decisions where heavily communicating VMs can be placed in physical servers with long distance network communication.

Similar to the works by Huang *et al.* [63], this work has expounded a VM placement problem based on proportional fairness and convex optimization to address the combined

[5]VMware Capacity Planner, 2014. `http://www.vmware.com/products/capacity-planner`
[6]Novell PlateSpin Recon, 2014. `https://www.netiq.com/products/recon/`

problem of reducing energy-consumption and data center traffic volume in order to improve scalability. During the problem formulation, both server-side resource capacity constraints and application-level inter-VM traffic demands are considered. However, given the problem definition, no algorithm or placement mechanism is presented in the work in order to solve the problem. Furthermore, simulation-based evaluation is presented and it is claimed that the combined VM placement algorithm outperforms random and FFD-based VM placement algorithms.

Application-aware VM Migration in Data Centers

Shrivastava *et al.* [106] have addressed the load balancing problem in virtualized data centers trough migration of overloaded VMs to underloaded physical servers such that the migration would be network-aware. The authors have argued that when VMs (part of multi-tier applications) are migrated to remove hot spots in data centers can introduce additional network overhead due to the inherent coupling between VMs based on communication, especially when moved to servers that are distant in terms of network distance. With the goal of finding destination servers for overloaded VMs that would result in minimum network traffic after the migration, the authors have formulated the VM migration as an optimization problem and proposed a network topology-aware greedy heuristic.

The proposed optimization problem is called application-aware since the complete application context running on top of the overloaded VM is considered during the migration decision. A view of the interconnections of the VMs comprising a multi-tier application is modeled as a dependency graph consisting of VMs as vertices and inter-VM communications as edges of the graph. The authors have also modeled the network cost function as a product of traffic demand of edge and network distance of the corresponding host servers, where such network distance can be defined as latency, delay, or number of hops between any two servers. Furthermore, server-side resource capacity constraint is also included in the problem formulation.

Since such optimization problem is NP-complete, the authors have proposed a greedy approximate solution named AppAware that attempts to reduce the cost during each migration decision step while considering both application-level inter-VM dependencies and underlying network topology. AppAware has the following four stages:

1. *Base Algorithm:* for each overloaded VM in the system, the total communication weight is computed and based on this the overloaded VMs are sorted in decreasing order, and then for each feasible destination server, the migration impact factor is computed. The impact factor gives a measure of the migration overhead based on the defined cost function due to the potential migration. Finally, the base algorithm selects the destination host for which the migration impact factor is the minimum, provided that the destination host has enough resources to accommodate the migrating VM.

2. *Incorporation of Application Dependency:* this part of AppAware computes the total cost to migrate a VM to a destination server as the sum of its individual cost corresponding to each of its peer VM that the migrating VM has communication.

3. *Topology Information and Server Load:* this part of AppAware considers network topology and neighboring server load while making migration decisions since a physical server that is close (in terms of topological distance) to other lightly loaded servers would be of higher preference as destination for a VM due to its potential for being capable of accommodating it dependent VMs to nearly servers.

4. *Iterative Refinements:* AppAware is further improved by incorporating two extensions to minimize the data center traffic. The first extension computes multiple values of the migration impact over multiple iterations of the AppAware base algorithm and the second extension further refines upon the previous extension by considering expected migration impact of future mappings of other VMs for a given candidate destination server at each iteration.

Based on numerical simulations, the authors have reported performance evaluation of AppAware by comparing with the optimal solution and Sandpiper black-box and grey-box migration scheme [127]. Run-time server-side remaining resource capacity (CPU, memory, and storage) and VM resource demands are generated using normal distribution, whereas inter-VM communication dependencies are generated using normal, exponential, and uniform distributions with varying mean and variance. Since the formulated migration problem is NP-hard, the performance of AppAware and Sandpiper are compared with optimal migration decisions only for small scale data centers (with 10 servers) and AppAware is reported to have produced solutions that are very close to the optimal solutions.

For large data centers (with 100 servers), AppAware is compared against Sandpiper and it is reported that AppAware outperformed Sandpiper consistently by producing migration decisions that decreased traffic volume transported by the network by up to 81%. Moreover, in order to assess the suitability of AppAware against various network topologies, AppAware is compared to optimal placement decisions for Tree and VL2 network topologies. It is reported that AppAware performs close to optimal placement for Tree topology, whereas the gap is increased for VL2.

AppAware considered server-side resource capacity constraints during VM migration, but it does not consider the physical link bandwidth capacity constraints. As a consequence, subsequent VM migrations can cause network links of low distance to get congested.

3.4 Summary and Conclusions

This chapter has presented the most notable features of the recent research works on VM management, in particular VM placement, migration, and consolidation with various optimization goals, including network cost and energy consumption reduction, and optimization of resource utilization. It has also presented a detailed taxonomy of VM management that can help analyze other research works in this area. A detailed survey and analysis has also been presented with tabular format so facilitate comparison on the similarities and differences of the studied works.

Chapter 4

Multi-objective Virtual Machine Placement

This chapter presents a multi-objective Virtual Machine (VM) placement approach for virtualized data center environments with the goal of minimizing both server power consumption and resource wastage. The VM placement problem is modeled as an instance of a combinatorial optimization problem which is computationally $\mathcal{NP}-hard$. A modified Ant Colony Optimization (ACO) metaheuristic-based VM placement algorithm is proposed, incorporation with a balanced resource utilization capture method across multiple resources, such as CPU, memory, and network I/O. Simulation-based performance evaluation demonstrates superior performance of the proposed approach compared to representative VM placement strategies where the proposed technique reduces resource wastage up to 89% and power consumption up to 16%.

4.1 Introduction

As outlined in Chapter 1, inefficient use of resources and high energy consumption are two of the main challenges in large-scale data center management. In order to address these issues, this chapter presents a technique for modeling multi-dimensional server resource utilization and an online *Virtual Machine* (VM) placement strategy focusing on consolidated VM cluster deployment. The proposed VM placement algorithm is application-agnostic and adapts *Ant Colony Optimization* (ACO) metaheuristic to navigate the search space effectively within a reasonable amount of time, making it suitable for online or real-time VM placement scenarios. The proposed placement algorithm strives to consolidate VMs into a

minimum number of servers (homogeneous) with the goal of minimizing the accumulated energy consumption and server resource wastage across multiple resource dimensions.

Underutilization of computing resources is one of the main reasons for high resource wastage in enterprise data centers. Furthermore, inefficient use is one of the key factors for the extremely high energy consumption, because of the narrow dynamic power range of physical servers— completely idle servers consume about 70% of their peak power usage [42]. Virtualization technologies enable data centers to address the problems of energy and resource inefficiencies by hosting multiple VMs in a single server, where hosted VMs share the underlying physical resources. One of the main applications of virtualization technology in enterprise data centers is *VM Consolidation*, a strategy to consolidate a group of VMs into a minimal number of servers (homogeneous) to achieve better utilization and reduce resource wastage [123]. Due to the non-proportional power usage of servers, consolidated VM placement, which requires a minimal number of servers, eventually results in reduced energy consumption. In addition to these obvious advantages, consolidated VM placement has other benefits, such as data center physical space minimization, maintenance automation, and reduced labor costs.

Applications deployed in Clouds vary in their resource demands perspectives— some applications require balanced resources including cluster computing and mid-range databases, whereas others require high processing power, such as web servers and batch processing, and others require higher memory capacities for applications such as high performance databases and large deployment of enterprise applications. In order to meet the diversified resource demands of customer applications, Infrastructure-as-a-Service (IaaS) Cloud providers such as Amazon, Google, and Microsoft offer various categories of predefined VMs with different amounts of resources across multiple dimensions (such as CPU, memory, and network bandwidth), as well as custom VMs, for which customers can choose the amount of resources for their VM instances[1]. Such multi-dimensionality of resources adds extra complexity to the problem of reducing the number of servers used. Exhausting the available resource in a particular dimension (e.g., CPU) causes a server to be unable to accommodate more VMs whereas the available resources in other dimensions (e.g., memory and network bandwidth) are sufficient to meet other VM requirements. This causes

[1]Google Cloud Platform— Compute Engine, 2016. `https://cloud.google.com/compute/`

resource fragmentation in active servers and, as a consequence, wastes large amounts of server resources in the data center.

VM categories offered by IaaS Clouds include general purpose instances with balanced resources that can be used to host regular Internet applications, as well as VMs with optimized resources focusing on specific types of applications. VMs with optimized resources often demonstrate the pattern of complementary amounts of resources across different dimensions. Table 4.1 shows typical Amazon EC2 instances[2] (VMs) categorized into three classes along with the number of Virtual CPUs (vCPU) and main memory (GiB) capacity. The General Purpose instances offer balanced resources, whereas Compute Optimized and Memory Optimized instances offer higher CPU and memory capacities, respectively. It is clear from Table 4.1 that Compute Optimized instance models c4.xlarge, c4.2xlarge, c4.4xlarge, and c4.8xlarge have almost double the amount of vCPU and half the amount of memory compared to Memory Optimized models r3.large, r3.xlarge, r3.2xlarge, and r3.4xlarge, respectively. These VM instance types demonstrate the explicit pattern of complementary resource demands that can be exploited by VM placement policies to achieve higher server resource utilization across multiple dimensions and reduce overall server resource wastage. A higher degree of server resource utilization will eventually enable Cloud providers to accommodate larger numbers of VMs in the data centers as resource wastage is minimized, and thus foster the achievement of higher scalability. Furthermore, as argued in Section 4.2, improvement of overall utilization through consolidated VM placement helps to keep the number of active servers at a minimal number, and thus reduce energy consumption.

Most of the existing studies on energy-aware consolidated VM placement techniques apply simple greedy heuristics such as First Fit Decreasing (FFD) [121,127], Best Fit [88,107], and Best Fit Decreasing [10]. As explained in the next section, VM consolidation is in fact an NP-hard combinatorial optimization problem [17] and greedy approaches are not guaranteed to generate near-optimal solutions. Moreover, most of these approaches use inappropriate mean estimators that fail to capture the multi-dimensional aspect of resource utilization. Other studies [60, 117] utilize *Constraint Programming* (CP) to achieve high VM packing efficiency; however, by the use of CP, the proposed frameworks effectively restrict the domain of possible values for the total number of servers and VMs, and thus

[2]Amazon EC2 Instance Types, 2016. `https://aws.amazon.com/ec2/instance-types/`

Table 4.1: Typical Amazon EC2 Instance (VM) Types

Instance Category	Model	# of vCPU	Memory (GiB)
General Purpose	m4.large	2	8
	m4.xlarge	2	16
	m4.2xlarge	8	32
	m4.4xlarge	16	64
	m4.10xlarge	40	160
Compute Optimized	c4.large	2	3.75
	c4.xlarge	4	7.5
	c4.2xlarge	8	15
	c4.4xlarge	16	30
	c4.8xlarge	36	60
Memory Optimized	r3.large	2	15.25
	r3.xlarge	4	30.5
	r3.2xlarge	8	61
	r3.4xlarge	16	122
	r3.8xlarge	32	244

can address only limited search space. Other studies [11, 12] ignore the multi-dimensional aspect of VM resource demand and consider only one type of resource for consolidation (primarily CPU demand), ignoring other crucial resources, such as main memory and network I/O; whereas over-commitment of these resources, specially memory can effectively degrade the performance of hosted applications. Furthermore, simple mean estimators for deriving scalar form of the multi-dimensional resource utilization (e.g., L1-norm) fail to achieve balanced resource utilization, and in effect, degrade the performance of consolidation techniques [46, 50, 127].

In contrast, the approach presented in this chapter considers multi-dimensional server resource capacities and VM resource demands in the system model, and focuses on balanced resource utilization of servers for different resource types in order to increase overall server resource utilization. The consolidated VM cluster placement is modeled as an instance of the *Multi-dimensional Vector Packing Problem* (MDVPP) and the ACO [35] metaheuristic is adapted to address the problem, incorporating an extended version of the vector algebra-based multi-dimensional server resource utilization capture method [88]. Simulation-based evaluation shows that the proposed multi-objective consolidated VM placement algorithm outperforms four state-of-the-art VM placement approaches on several performance metrics.

The proposed multi-objective consolidated VM cluster placement approach can be applied in several practical scenarios including the following:

1. During the initial VM deployment phase when Cloud providers handle customers' requests to create VMs in the data center.

2. Intra-data center VM cluster migration. Such situations can arise during data center maintenance or upgrade when a group of active VMs needs to be moved from one part of a data center to another (using either cold or live VM migration).

3. Inter-data center VM cluster migration. Such situations can arise when Cloud consumers want to move VM clusters from one Cloud provider to another (inter-Cloud VM migration). Other applications of inter-data center VM migration are situations like replications and disaster management.

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

1. The *Multi-objective Consolidated VM Placement Problem* (MCVPP) is formally defined as a discrete combinatorial optimization problem with the aims of minimizing the power consumption and resource wastage.

2. A balanced server resource utilization capture technique across multiple resource dimensions based on vector algebra. This utilization capture technique is generic that helps in utilizing complementary resource demand patterns among VMs and can be readily integrated to any online or offline VM management strategies.

3. Adaptation of the ACO metaheuristic to apply in the problem domain of VM placement, incorporation of balanced resource utilization through heuristic information, and eventually, formulation of a novel *ACO- and Vector algebra-based VM Placement* (AVVMP) algorithm as a solution to the proposed MCVPP problem.

4. Simulation-based experimentation and performance evaluation are conducted of the proposed VM placement algorithm taking into account multiple scaling factors and performance metrics. The results indicate that the proposed consolidated VM placement approach outperforms the competitor VM placement techniques across all performance metrics.

The remainder of this chapter is organized as follows. The next section introduces the mathematical frameworks formulated to define the multi-objective VM placement problem and associated models used in the proposed placement approach. Section 4.3 provides a

brief background on ACO metaheuristics. The proposed AVVMP multi-objective VM placement algorithm is presented in Section 4.4, followed by a performance evaluation and analysis of experiemental results in Section 4.5. Finally, Section 4.6 concludes the chapter with a summary of the contributions and results.

4.2 Multi-objective Consolidated VM Placement Problem

This section begins by presenting the mathematical framework modeled to define the multi-objective consolidated VM placement problem (MCVPP). Next, it presents the proposed vector algebra-based mean estimation technique to capture multi-dimensional resource utilization of physical machines. Furthermore, it provides relevant models to estimate resource utilization and wastage, and power consumption of physical machines, which are utilized later in the proposed ACO- and vector algebra based VM placement (AVVMP) algorithm.

4.2.1 Modeling Multi-objective VM Placement as a MDVPP

In computational complexity theory, MDVPP is categorized as an $\mathcal{NP}$–*hard* combinatorial optimization problem [17] where m number of items, each item j having d weights $w_j^1, w_j^2, \ldots, w_j^d \geq 0$ ($j = 1, \ldots, m$ and $\sum_{l=1}^{d} w_j^l > 0$), have to be packed into a minimum number of bins, each bin i having d capacities $W_i^1, W_i^2, \ldots, W_i^d > 0$ ($i = 1, \ldots, n$), in such a way that the capacity constraints of the bins for each capacity dimension are not violated [22]. The bin capacity constraint for any particular dimension l means that the combined weight of the items packed in a bin in dimension l is less than or equal to the bin capacity in that dimension. In the research literature, the consolidated VM placement problem is often referred to as an instance of the *Multi-dimensional Bin Packing Problem* (MDBPP), which has different capacity constraints than that of MDVPP. As an illustration, for a 2-dimensional bin of length A and width B containing s number of items each having length a_j and width b_j, the capacity constraints of MDBPP can be expressed by the following equation:

$$\sum_{j=1}^{s} a_j \times b_j \leq A \times B. \tag{4.1}$$

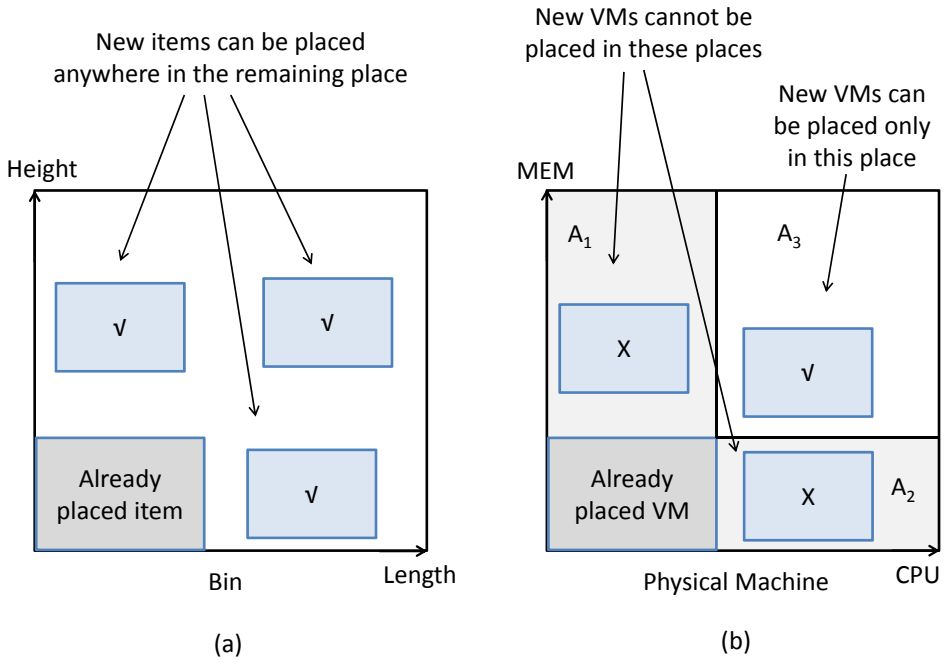

Figure 4.1: (a) 2-Dimensional Bin Packing Problem, and (b) 2-Dimensional VM Packing Problem.

On the other hand, in the case of MDVPP, the capacity constraints would be as follows:

$$\sum_{j=1}^{s} a_j \leq A \text{ and } \sum_{j=1}^{s} b_j \leq B. \tag{4.2}$$

The MCVPP problem is in fact an instance of MDVPP, as defined in the later part of this section. The difference is further illustrated in Figure 4.1, which shows the constraints of the packing problems for two dimensions. In the case of 2-dimensional bin packing in Figure 4.1(a), any unused 2-dimensional space is available for placing new items. However in the case of 2-dimensional VM packing, modeled as 2-dimensional vector packing, in Figure 4.1(b), areas A_1 and A_2 cannot be used for placing new VMs, since for these areas the CPU and memory capacities of the physical machine, respectively, are used up by the VM that is already placed. Any new VM placement request must be fulfilled by using area A_3, for which both CPU and memory capacities are available.

Table 4.2: Notations and their meanings

Notation	*Meaning*
VM	Virtual Machine
VMS	Set of VMs in a cluster
VM_i	An individual VM in set VMS
N_v	Total number of VMs in a cluster
$vmList$	Ordered list of VMs in a cluster
PM	Physical Machine
PMS	Set of PMs in a data center
PM_p	An individual PM in set PMS
N_p	Total number of PMs in a data center
$pmList$	Ordered list of PMs in a data center
RC	Single computing resource in PM (CPU, memory, network I/O)
RCS	Set of computing resources available in PMs
RC_l	An individual resource in set RCS
N_r	Number of resource types available in PM
C_p	Resource Capacity Vector (RCV) of PM_p
U_p	Resource Utilization Vector (RUV) of PM_p
D_i	Resource Demand Vector (RDV) of VM_i
x	VM-to-PM Placement Matrix
y	PM Allocation Vector
f_1	MCVPP Objective Function
τ	Pheromone matrix
τ_0	Initial pheromone amount
η	Heuristic value
δ	Global pheromone decay parameter
$\Delta\tau$	Pheromone reinforcement

Problem Definition

The *Physical Machines* (PMs) or servers in the data center are modeled as bins and the VMs as items to pack into the bins. Let PMS denote the set of N_p homogeneous PMs and VMS denote the set of N_v VMs to be deployed in the data center. The set of N_r types of resources available in the PMs is represented by RCS. Table 4.2 provides a list of important notations used throughout this chapter.

Each PM_p ($PM_p \in PMS$) has a N_r-dimensional *Resource Capacity Vector* (RCV) $C_p = \langle C_p^1, \ldots, C_p^l, \ldots, C_p^{N_r} \rangle$, where C_p^l denotes the total capacity of resource RC_l in PM_p. Similarly, each VM_i ($VM_i \in VMS$) is represented by its N_r-dimensional *Resource Demand Vector* (RDV) $D_i = \langle D_i^1, \ldots, D_i^l, \ldots, D_i^{N_r} \rangle$, where D_i^l denotes the demand of resource RC_l by VM_i. The *Resource Utilization Vector* (RUV) $U_p = \langle U_p^1, \ldots, U_p^l, \ldots, U_p^{N_r} \rangle$ of PM_p is

computed as the sum of the RDVs of the hosted VMs:

$$U_p^l = \sum_{\forall i:\, x_{i,p}=1} D_i^l \tag{4.3}$$

where x is the *Placement Matrix* that models the VM-to-PM placements and is defined as follows:

$$x_{i,p} = \begin{cases} 1, & \text{if } VM_i \text{ is placed in } PM_p; \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

Another decision variable *PM Allocation Vector* y is introduced, where each element y_p equals 1 if PM_p is hosting at least 1 VM, or 0 otherwise:

$$y_p = \begin{cases} 1, & \text{if } \sum_{i=1}^{N_v} x_{i,p} > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

The goal of the AVVMP algorithm is to place the VMs in the available PMs in such a way that: 1) resource utilization of active PMs is maximized across all dimensions; and 2) power consumption of active PMs is minimized. A careful analysis of the above two objectives reveals that both can be captured by a single *Objective Function* (OF)—minimization of the number of active PMs. Any placement decision that results in the least number of active PMs has the highest resource utilization across all dimensions compared to other placement decisions that require greater numbers of active PMs. Moreover, since available models for server power consumption primarily focus on CPU utilization [42], with a given number of VMs having specific CPU demands, it can be concluded that any placement decision that needs the least number of active PMs will have minimum energy consumption compared to others. Therefore, the OF f_1 is formulated as a single minimization function on y:

$$\textbf{minimize}\, f_1(y) = \sum_{p=1}^{N_p} y_p. \tag{4.6}$$

Finally, the PM resource capacity constraint is expressed as follows:

$$\sum_{i=1}^{N_v} D_i^l x_{i,p} \leq C_p^l, \forall p \in \{1, \ldots, N_p\}, \forall l \in \{1, \ldots, N_r\}. \tag{4.7}$$

Table 4.3: Example multi-dimensional VM resource demands and corresponding scalar values using mean estimator based on L1-norm

	VM_1	VM_2	VM_3	VM_4	VM_5	VM_6	VM_7	VM_8
CPU	10	61	5	14	49	17	13	19
MEM	19	7	54	18	37	18	25	20
L1-norm	29	68	59	32	86	35	38	39

The above constraint (4.7) ensures that the resource demands of all the VMs assigned to any PM do not exceed its resource capacity for each individual resource. The following constraint guarantees that a VM is assigned to at most one PM:

$$\sum_{p=1}^{N_p} x_{i,p} \leq 1, \forall i \in \{1, \ldots, N_v\}. \tag{4.8}$$

4.2.2 Modeling Multi-dimensional Resource Utilization based on Vector Algebra

Capturing the net effect of placing VMs with multiple resource demands in PMs with multiple resource capacities is one of the most important factors for any VM consolidation algorithm, since saturation of only one resource type can lead to no further improvement in utilization while leaving other types of resources underutilized. Existing solutions either use some sort of mathematical modeling to convert the multi-dimensional resource into scalar form [46,127], or consider only one type of resource while ignoring others [10]. Flaws in capturing the net effect of multi-dimensional resources lead consolidation algorithms (especially deterministic approaches) to solutions which fail to optimize overall resource utilization and as a result, they are energy-inefficient. Mean estimator based on L1-norm used in several recent studies on VM placement and consolidation [45, 46], and the flaw of using L1-norm to convert multiple resource demand vector into scalar form is demonstrated next using an example placement. Anomalies in other mathematical approaches can be found in [88].

Table 4.3 shows 8 VMs with their corresponding CPU and memory demands in terms of percentage of CPU and memory capacities of the PMs, along with the values of L1-norm (2 types of resources are used to simplify the demonstration). Traditionally, vector packing problems are solved using fast heuristic algorithms [76] and FFD is among the most efficient [97]. Using mean estimator based on L1-norm, the FFD heuristic

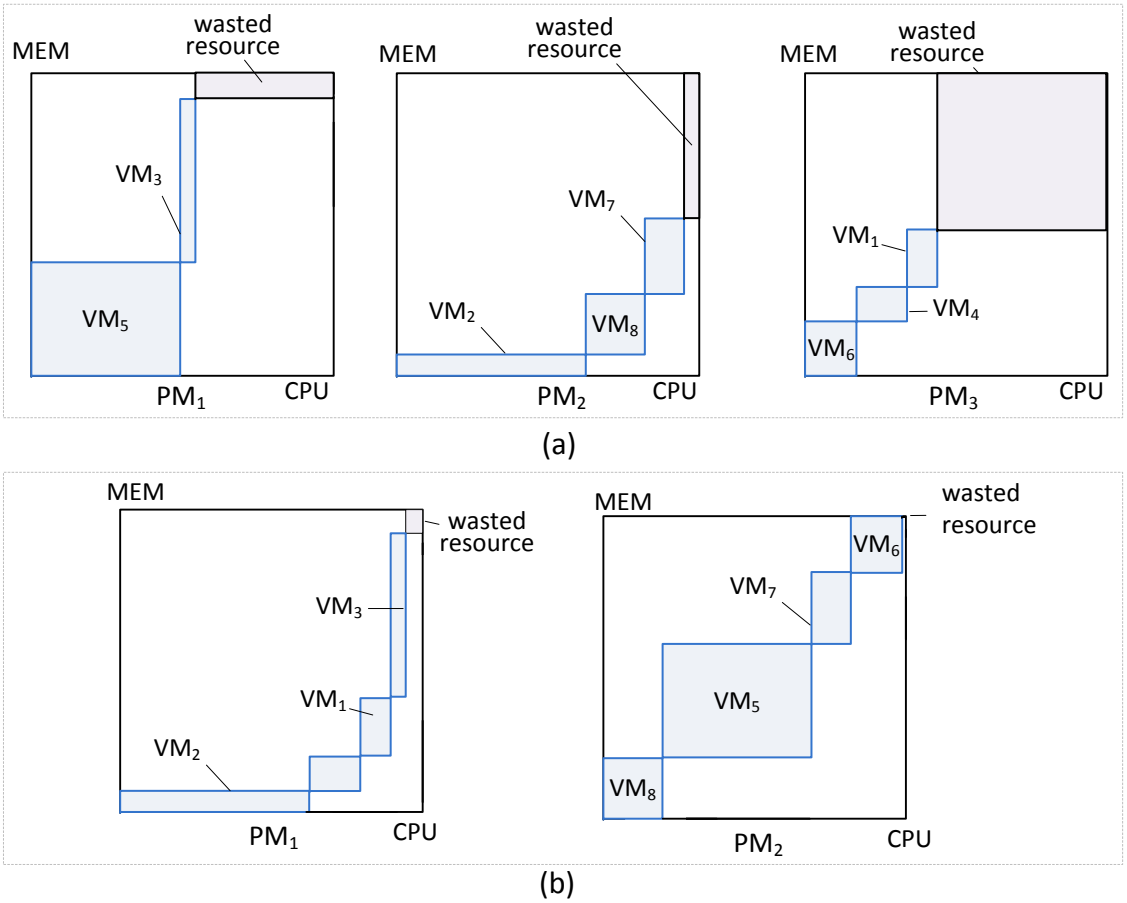

Figure 4.2: (a) FFD placement based on L1-norm mean estimator and (b) Alternative placement with less resource wastage.

first sorts the VMs into decreasing order of their L1-norm values, producing the list $\langle VM_5, VM_2, VM_3, VM_8, VM_7, VM_6, VM_4, VM_1 \rangle$. Then, it takes out the first VM from the list (VM_5) and tries to place it in the first PM PM_1 that can accommodate VM_5 (Figure 4.2(a)). Next, FFD takes out the second VM from the list (VM_2) and tries to place it in the first PM again. However, this time PM_1 cannot accommodate VM_2, since the CPU demand of VM_2 is 61, which is greater than the available CPU capacity of 51 of PM_1. As a result, FFD tries to place VM_2 in the second PM that can accommodate VM_2. Subsequently, FFD takes the next VM from the sorted list and tries to place in a PM starting from the first one. Figure 4.2(a) shows the consolidated placement of the 8 VMs produced by the FFD heuristic using mean estimator based on L1-norm, which requires 3 PMs with significant resource wastage. In contrast, an an alternative placement focusing on complementary resource utilization is shown in Figure 4.2(b), which requires only 2 PMs with much less resource wastage. This example reveals one feature of efficient consolidation—balanced utilization of resources across all dimensions.

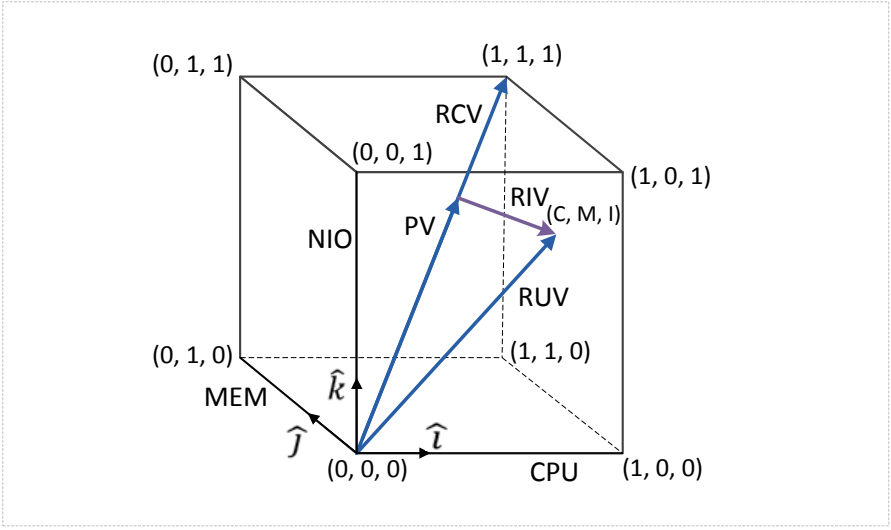

Figure 4.3: Balanced resource utilization using vector algebra.

In order to capture both balanced and overall resource utilization, the vector algebra-based complementary resource utilization capture technique [88] is augmented and integrated in the proposed ACO-based solution. Three main resources are considered in the model in the context of consolidated VM placement in Cloud data centers: CPU, memory (MEM), and network I/O (NIO). Storage resources are considered to be provided on-demand through storage backbones such as Network Attached Storage (NAS) and Storage Area Network (SAN) (e.g., Amazon EBS[3]).

PM's normalized resource capacity is expressed as a unit cube (*Resource Cube*), with the three dimensions representing three types of resources (Figure 4.3). Resource-related information are expressed as vectors within the resource cube: The resource capacity vector (RCV) represents the total capacity of PM and the resource utilization vector (RUV) represents the current resource utilization of PM, which is computed by vector addition of the normalized resource demand vectors (RDVs) of the hosted VMs (4.3). After placing a VM in the PM, the RUV is represented by $RUV = C\hat{i} + M\hat{j} + N\hat{k}$, where C, M, and N are the combined resource demands (normalized) of the hosted VMs in the dimension of CPU, MEM, and NIO, and $\hat{i}$, $\hat{j}$, and $\hat{k}$ are the unit vectors along those three resource dimensions, respectively.

The *Projection Vector* (PV) of RUV on RVC is computed by multiplying the unit vector along RCV (right term) with the magnitude of projection of RUV on RVC (left

[3]Amazon Elastic Block Store (EBS), 2016. `https://aws.amazon.com/ebs/`

term):

$$\begin{aligned} PV &= \frac{1}{\sqrt{3}}(C+M+N)\left(\frac{1}{\sqrt{3}}\hat{i}+\frac{1}{\sqrt{3}}\hat{j}+\frac{1}{\sqrt{3}}\hat{k}\right) \\ &= \left(\frac{C+M+N}{3}\right)\hat{i}+\left(\frac{C+M+N}{3}\right)\hat{j}+\left(\frac{C+M+N}{3}\right)\hat{k}. \end{aligned} \tag{4.9}$$

To capture the degree of imbalance in current resource utilization of a PM, *Resource Imbalance Vector* (RIV) is used, which is computed as the vector difference between RUV and PV:

$$RIV = (C-H)\,\hat{i}+(M-H)\,\hat{j}+(I-H)\,\hat{k} \tag{4.10}$$

where $H = (C+M+I)/3$. When selecting among VMs for placement in a PM, the VM that shortens the magnitude of RIV most is the VM that balances the resource utilization of the PM maximum across different dimensions. The magnitude of RIV is given by the following equation:

$$\|RIV\| = \sqrt{(C-H)^2+(M-H)^2+(I-H)^2}. \tag{4.11}$$

For normalized resources utilization, C, M, and N fall in the range of $[0,1]$, and therefore, $\|RIV\|$ has the range $[0.0, 0.82]$. This $\|RIV\|$ is used to define the heuristic information for the proposed AVVMP algorithm along with the overall resource utilization of PM (4.18).

4.2.3 Modeling Resource Utilization and Wastage

The overall resource utilization of PM p is modeled as the summation of the normalized resource utilization of each individual resource type:

$$Utilization_p = \sum_{l \in RCS} U_p^l \tag{4.12}$$

where RCS is the set of available resources (in this case, $RCS = \{CPU, MEM, NIO\}$) and U_p^l is the utilization of resource $l \in RCS$ (4.3).

Similarly, resource wastage is modeled as the summation of the remaining (unused) resources (normalized) of each individual resource type:

$$Wastage_p = \sum_{l \in RCS} (1 - U_p^l). \tag{4.13}$$

4.2.4 Modeling Power Consumption

Power consumption of physical servers is dominated by their CPU usage and can be expressed as a linear expression of current CPU utilization [42]. Therefore, the electricity energy drawn by a PM p is modeled as a linear function of its CPU utilization $U_p^{CPU} \in [0, 1]$:

$$E(p) = \begin{cases} E_{idle} + (E_{full} - E_{idle}) \times U_p^{CPU}, & \text{if } U_p^{CPU} > 0; \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

where E_{full} and E_{idle} are the average energy drawn when a PM is fully utilized (i.e., 100% CPU busy) and idle, respectively.

Due to the non-proportional power usage (i.e., high idle power) of commodity physical servers, servers that do not host any active VM are considered to be turned to power save mode (e.g., suspended or turned off) after the VM deployment. Therefore, these servers are not considered in this energy consumption model. Therefore, the estimate of total energy consumed by a VM placement decision x is computed as the sum of the individual energy consumption of the active PMs:

$$E(x) = \sum_{p \in PMS} E(p). \tag{4.15}$$

4.3 Ant Colony Optimization Metaheuristics

In the last two decades, ants have inspired a number of methods and techniques, among which the most studied and the most successful is the general purpose optimization technique known as *Ant Colony Optimization* (ACO) [38]. ACO takes inspiration from the foraging behavior of some ant species. These ants deposit a chemical substance named pheromone on the ground in order to mark some favorable paths. Other ants perceive the presence of pheromone and tend to follow paths where pheromone concentration is higher. This is a colony-level behavior of the ants that exploits positive feedback, termed *autocatalysis*, can be utilized by the ants in order to find the shortest path between a food source and their nest [32]. Similar to the behaviors of natural ants, in ACO a number of artificial ants build solutions to the optimization problem at hand and share information on the quality of these solutions via a communication mechanism that is similar to that used by real ants [35].

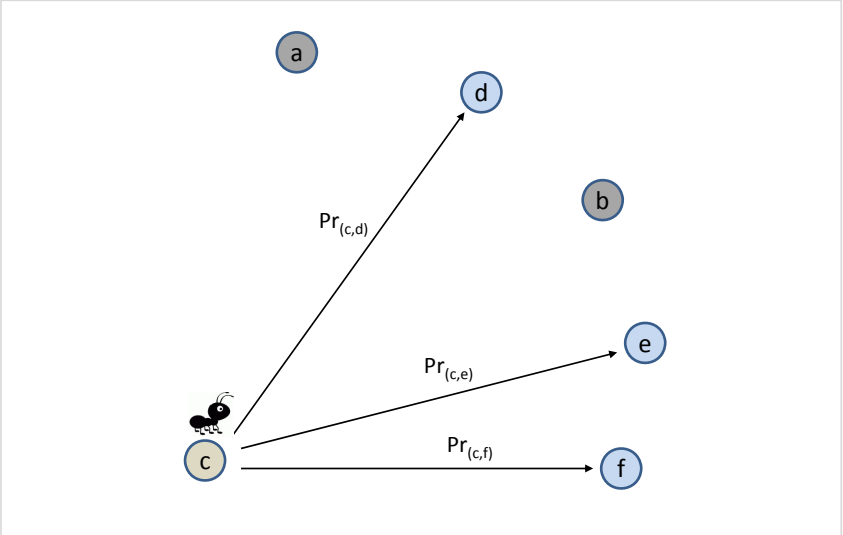

Figure 4.4: After visiting cities a and b, an ant is currently in city c and selects the next city to visit among unvisited cities d, e, and f based stochastically on the associated pheromone levels and the distances of edges (c, d), (c, e), and (c, f).

Since the ACO metaheuristics are computational methods rather than specific concrete algorithms, the ACO approach is best understood by using an appropriate example problem, such as the classical Traveling Salesman Problem (TSP). The problem statement of TSP is as follows— a set of cities is given where the inter-city distances for all the cities are known *a priori* and the objective is to find the shortest tour by visiting each city once and only once. Generally, the problem is represented using a graph, where each vertex denotes a city and each edge denotes a connection between two cities.

When applied to TSP, a number of artificial ants are put in different cities randomly and each ant walks through the TSP graph to build its individual TSP solution. Each edge of the graph is associated with a pheromone variable that stores the pheromone amount for that connection, and the ants can read and modify the value of this variable. ACO is an iterative algorithm that incrementally refines previously-built solutions. In every iteration, each ant builds a solution by simulating a walk from the initial vertex to other vertices following the condition of visiting each vertex exactly once. At each step of the solution-building process, the ant selects the next vertex to visit using a probabilistic decision rule based on the associated pheromone concentration and the distance between cities. For example, in Figure 4.4, the ant is currently in city c and cities a and b are already visited. The next city to visit is chosen from the cities d, e, and f. Among these cities, the ant can select any city, say city d, with a probability proportional to the

pheromone level of the edge (c, d) and inversely proportional to the distance between cities c and d. Each such edge of the graph that the ant chooses in every step in its tour denotes a *solution component* and all the solution components that the ant selects to complete its tour make up a solution. When all the ants finish building their solutions (i.e., tours), the pheromone levels are updated on the basis of the quality of the solutions, with the intention of influencing ants in future iterations to build solutions similar to the best ones previously built.

The first ACO algorithm is known as the *Ant System* (AS) [37] and was proposed in the early 90s. Since then, a number of other ACO algorithms have been introduced. The main differing characteristic of the AS algorithm compared to later ACO algorithms is that in AS, at the end of each cycle, each ant that has built a solution deposits an amount of pheromone on the path depending on the quality of its solution. Later, Dorigo *et al.* [36] proposed *Ant Colony System* (ACS), where the random-proportional rule is updated and a local pheromone update rule is added to diversify the search performed by the subsequent ants during an iteration. Stützle *et al.* [111] presented *Max-Min Ant System* (MMAS), an improvement over the original AS. Its characterizing elements are that only the best ant updates the pheromone trails and that the value of the pheromone is bounded within a predefined range $[\tau_{min}, \tau_{max}]$.

4.4 Proposed Solution

This section starts by presenting the motivation for using an ACO metaheuristic-based approach for solving the MCVPP problem. Then, it presents the adaptation of the algorithmic features of the ACO so that it can be applied in the context of VM cluster placement in a data center. Finally, a detailed description of the proposed AVVMP algorithm is provided.

4.4.1 Motivation for Applying ACO for Consolidated VM Placement

ACO metaheuristics have been proven to be efficient in various problem domains and to date have been tested on more than one hundred different $\mathcal{NP}$*–hard* problems, including discrete optimization problems [35]. The overall empirical results that emerge from the tests show that, for many problems, ACO algorithms produce results that are very close to those of the best-performing algorithms, while on others they are the state-of-the-art [38].

In [76] and [17], the authors have shown, based on experimental results, that for the one-dimensional Bin Packing Problem adapted versions of the ACO algorithms can outperform the best performing Evolutionary Algorithms (EAs) on this problem, especially for large problem instances. As presented in Section 4.2, the MCVPP is in fact an instance of the MDVPP, which is also an $\mathcal{NP}-hard$ combinatorial optimization problem. For $\mathcal{NP}-hard$ problems, the best-known algorithms that guarantee to find an optimal solution have exponential time worst-case complexity and, as a result, applications of such algorithms are infeasible for large problem instances, such as consolidated VM cluster placement in Cloud data centers. In such cases, ACO algorithms offer to produce high-quality solutions in polynomial time complexity.

4.4.2 Adaptation of ACO Metaheuristic for Consolidated VM Placement

Since ACO metaheuristics are computational methods rather than specific concrete algorithms, the application of these metaheuristics requires appropriate representation of the problem at hand to match the ACO scenario and appropriate adaptation of ACO features to address the specific problem. In the original ACO metaheuristics [36], the authors proposed the use of pheromone values and heuristic information for each edge of the graph in the TSP and ants walking on the graph to complete their tours guided by the pheromone and heuristic values converging towards the optimal path.

Since consolidated VM placement modeled as MDVPP does not incorporate the notion of graph and path in the graph, each VM-to-PM assignment is considered as an individual solution component in place of an edge in the graph in its TSP counterpart. Thus, each artificial ant of the AVVMP algorithm produces a solution composed of a list of VM-to-PM assignments instead of a sub-graph. Pheromone levels are associated with each VM-to-PM assignment representing the desirability of assigning a VM to a PM ((4.16) and (4.23)) instead of each edge of the graph, and heuristic values are computed dynamically for each VM-to-PM assignment, representing the preference of assigning a VM to a PM in terms of both overall and balanced resource utilization of the PM (4.18).

Figure 4.5 illustrates the AVVMP solution construction process for a single ant using an example where four VMs need to be deployed in a data center. The ant starts with the first PM and computes probabilities for the placement of each of the four VMs using a probabilistic decision rule presented in (4.20) (Figure 4.5(a)). When the ant has selected

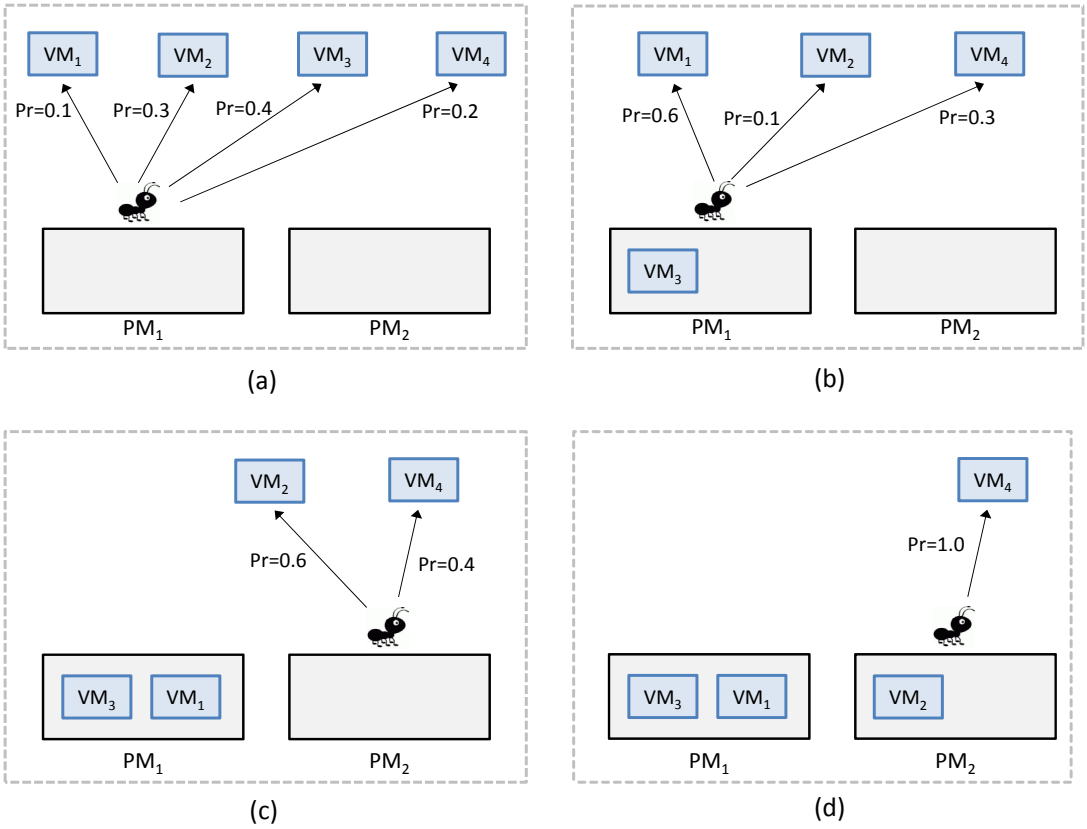

Figure 4.5: Illustration of an ant's VM selection process through example.

the first VM to place in the PM, it recomputes the probabilities of selecting each of the remaining VMs using the probabilistic decision rule (Figure 4.5(b)). The probabilities can differ in this step since this time PM_1 is not empty and the remaining VMs utilize the multi-dimensional PM resources differently compared to the empty PM case in Figure 4.5(a). When the first PM cannot accommodate any of the remaining VMs (Figure 4.5(c)), the ant considers the next PM and starts placing a VM from the set of remaining VMs using the same approach. This process continues until all the VMs are placed in PMs (Figure 4.5(d)).

After all the ants have finished building complete solutions, the best solution is identified based on the OF f_1 (4.6). The whole process is repeated multiple times until a predefined terminating condition is met. In order to increase the extent of exploration of the search space and avoid early stagnation to a sub-optimum, after each cycle the best solution found so far is identified and the pheromone levels of the solution components are reinforced.

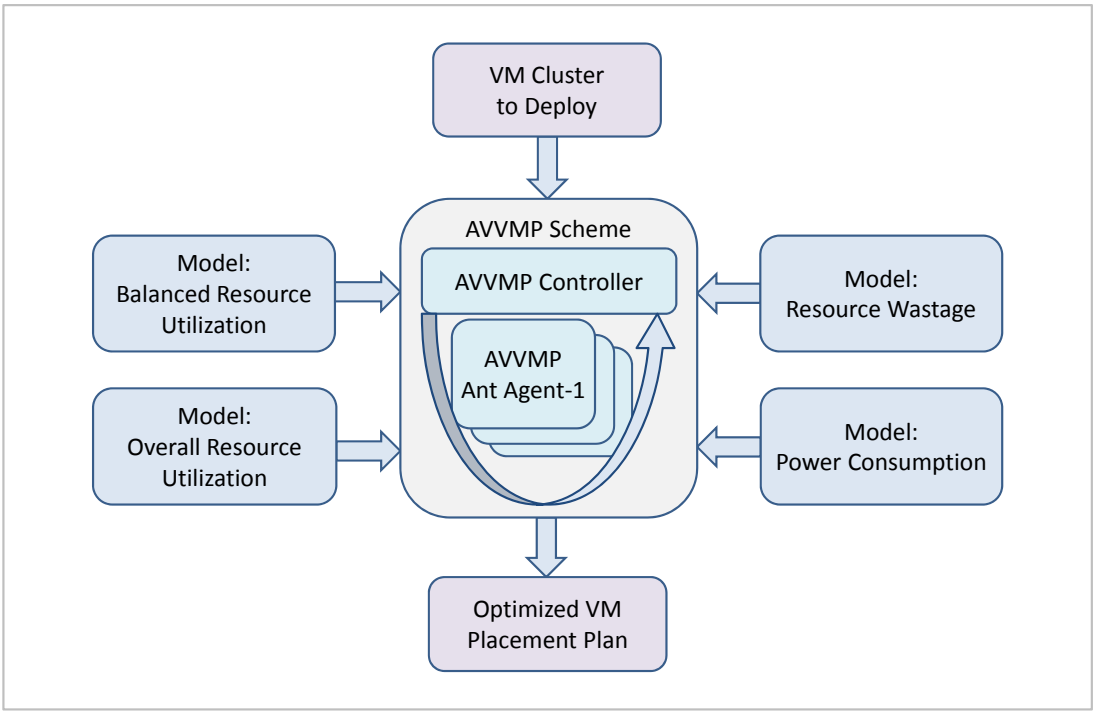

Figure 4.6: AVVMP algorithm with associated models.

4.4.3 AVVMP Algorithm

Figure 4.6 shows the main components of the proposed AVVMP VM placement algorithm. Taking the VM cluster to deploy in the data center as input, the controller component spawns multiple ant agents and passes each ant a copy of the input. The overall AVVMP scheme utilizes the various resource- and energy-related models formulated in Section 4.2. The ant agents run in parallel and produce VM placement plans (solutions) and pass them to the controller. The controller identifies the best solution, performs necessary updates on shared data, and activates the ant agents for the next iteration. Finally, when the terminating condition is met, the controller outputs the so-far-found best VM placement plan.

The pseudocode of the AVVMP algorithm is shown in Algorithm 4.1. Ants depositing pheromone on solution components is implemented using a $N_v \times N_p$ pheromone matrix τ. At the beginning of each cycle, each ant starts with an empty solution, a set of PMs, and a randomly shuffled set of VMs [lines 6-12]. The VM set is shuffled for each ant to randomize the search in the solution space. From lines 15-28, all the ants build their solutions based on a modified ACS rule.

In every iteration of the while loop, an ant is chosen randomly [line 16] and is allowed to choose a VM to assign next to its current PM among all the feasible VMs (4.21). In this way, parallel behavior among ants is implemented using sequential code. If the current PM is fully utilized or there is no feasible VM left to assign to the PM, another PM is taken to fill in [lines 18-20]. In lines 21-23, the chosen ant uses a probabilistic decision rule termed *pseudo-random-proportional rule* (4.20) that is based on the current pheromone concentration ($\tau_{i,p}$) on the $\langle VM, PM \rangle$ pair and heuristic information ($\eta_{i,p}$) that guides the ant to select the VMs that lead to better PM resource utilization and in the long run, a lower value of the OF f_1 (4.6) for the complete solution. Thus, the $\langle VM, PM \rangle$ pairs that have higher pheromone concentrations and heuristic values have higher probability of being chosen by the ant. When an ant is finished with all the VMs in its VM list, the number of PMs used for VM placement is set as the OF f_1 for its solution and the ant is removed from the temporary list of active ants ($antList$) [lines 25-26].

When all the ants have finished building their solutions (i.e., a cycle is complete), all the solutions computed by the ants in the current cycle are compared to the so far found *global-best-solution* (GBS) against their achieved OF (f_1) values (4.6). The solution that results in the minimum value for f_1 is chosen as the current GBS [lines 29-34].

At line 35, the pheromone reinforcement amount is computed based on (4.24). The amount of pheromone associated with each $\langle VM, PM \rangle$ pair is updated to simulate the pheromone evaporation and deposition according to (4.23) [lines 36-40]. The algorithm reinforces the pheromone values only on the $\langle VM, PM \rangle$ pairs that belong to the GBS.

After the global pheromone update, the whole process of searching new solutions is repeated. The algorithm terminates when no further improvement in the solution quality is observed for the last $nCycleTerm$ cycles [line 41]. The various parts of the AVVMP algorithm are formally defined in the remaining part of this section.

Algorithm 4.1 AVVMP Algorithm

Input: Set of PMs PMS and their RCV C_p, set of VMs VMS and their RDV D_i, set of ants $antSet$. Set of parameters $\{nAnts, nCycleTerm, \beta, \omega, \delta, q_0\}$.
Output: Global-best-solution GBS.
Initialization: Set parameters, set pheromone value for each $\langle VM, PM\rangle$ pair ($\tau_{i,p}$) to τ_0 [4.16], $GBS \leftarrow \emptyset$, $nCycle \leftarrow 0$.

1: **repeat**
2: **for each** $ant \in antSet$ **do** {Initialize data structures for each ant}
3: $ant.solution \leftarrow \emptyset$
4: $ant.pmList \leftarrow PMS$
5: $ant.p \leftarrow 1$
6: $ant.vmList \leftarrow VMS$
7: Shuffle $ant.vmList$ {Shuffle VM set to randomize search}
8: **end for**
9: $antList \leftarrow antSet$
10: $nCycle \leftarrow nCycle + 1$
11: **while** $antList \neq \emptyset$ **do**
12: Pick an ant randomly from $antList$
13: **if** $ant.vmList \neq \emptyset$ **then**
14: **if** $FV_{ant}(ant.p) = \emptyset$ **then** {Take new PM if current one is unable to host another VM}
15: $ant.p \leftarrow ant.p + 1$
16: **end if**
17: Choose a VM i from $FV_{ant}(ant.p)$ using probabilistic rule in (4.20) and place in PM p
18: $ant.solution.x_{i,p} \leftarrow 1$
19: $ant.vmList.remove(i)$
20: **else**{When all VMs are placed, then ant completes a solution and stops for this cycle}
21: $ant.solution.f \leftarrow p$
22: $antList.remove(ant)$
23: **end if**
24: **end while**
25: **for each** $ant \in antSet$ **do** {Find global-best-solution for this cycle}
26: **if** $ant.solution.f < GBS.f$ **then**
27: $GBS \leftarrow ant.solution$
28: $nCycle \leftarrow 0$
29: **end if**
30: **end for**
31: Compute $\Delta\tau$ {Compute pheromone reinforcement amount for this cycle}
32: **for each** $p \in PMS$ **do** {Simulate pheromone evaporation and deposition}
33: **for each** $i \in VMS$ **do**
34: $\tau_{i,p} \leftarrow (1 - \delta) \times \tau_{i,p} + \delta \times \Delta\tau_{i,p}$
35: **end for**
36: **end for**
37: **until** $nCycle = nCycleTerm$ {AVVMP ends if it sees no improvement for $nCycleTerm$ cycles}

Definition of Pheromone and Initial Pheromone Amount

At the beginning of any ACO algorithm, the ants start with a fixed amount of initial pheromone for each solution component. In the original proposal for the ACS metaheuristic [36], the initial pheromone amount for each edge is set to the inverse of the tour length of the TSP solution produced by a baseline heuristic (namely the nearest neighborhood heuristic) divided by the number of cities in the problem. This effectively captures a measure of the quality of the solution of the referenced baseline algorithm. Following

a similar approach, the initial pheromone amount for AVVMP is set to the quality of the solution produced by a reference baseline algorithm FFDL1 (FFD heuristic based on L1-norm mean estimator):

$$\tau_0 \leftarrow PE_{FFDL1} \tag{4.16}$$

where PE_{FFDL1} is the *Packing Efficiency* (PE) of the solution produced by the FFDL1 heuristic. The PE of any solution *sol* produced by an algorithm is given by:

$$PE_{sol} = \frac{N_v}{nActivePM}. \tag{4.17}$$

Definition of Heuristic Information

During a solution-building process, the heuristic value $\eta_{i,p}$ represents the measure of benefit of selecting a solution component $\langle i, p\rangle$. This is effectively the "greedy" part of the solution-building process that each ant exercises to improve the overall solution quality by choosing one solution component among all the feasible solution components. As the goal of AVVMP is to reduce the number of active PMs by packing VMs in a balanced way, the heuristic value $\eta_{i,p}$ is defined to favor both balanced resource utilization in all dimensions and higher overall resource utilization:

$$\eta_{i,p} = \omega \times (-\log_{10}\|RIV_p(i)\|) + (1-\omega) \times Utilization_p(i) \tag{4.18}$$

where $\|RIV_p(i)\|$ is the magnitude of RIV of PM p after assigning VM i to it (4.11). The negative of the logarithm of $\|RIV_p(i)\|$ is taken to give higher heuristic values to the $\langle i, p\rangle$ pairs that result in smaller magnitudes of RIV. $Utilization_p(i)$ is the overall resource utilization of PM p if VM i is assigned to it and is computed as the summation of the normalized resource utilization across all dimensions after assigning VM i:

$$Utilization_p(i) = \sum_{l \in RCS} (U_p^l + D_i^l). \tag{4.19}$$

Finally, $\omega \in [0, 1]$ is a parameter that trades off the relative importance of balanced versus overall resource utilization as per the definition.

For normalized resource capacities ($C^{CPU} = C^{MEM} = C^{NIO} = 1$), it has already been shown that $\|RIV\|$ falls in the interval $[0.0, 0.82]$ (Section 4.2.2). However, since the logarithm of zero is undefined, the interval $[0.001, 0.82]$ is used in the experimental

evaluation. Given this interval of $\|RIV\|$, the expression $-log_{10}\|RIV\|$ results in the interval $[0.086, 3.0]$ which is compatible to $Utilization_p$ in terms of metric, which results in the interval $[0.0, 3.0]$.

Pseudo-random Proportional Rule

When constructing a solution (Algorithm 4.1, line 21), an ant k selects a VM s to be assigned to PM p using the following *pseudo-random proportional rule* [36]:

$$s = \begin{cases} \arg\max_{i \in FV_k(p)}\{\tau_{i,p} \times [\eta_{i,p}]^\beta\}, & \text{if } q \leq q_0; \\ S, & \text{otherwise} \end{cases} \tag{4.20}$$

where q is a random number uniformly distributed in $[0, 1]$, q_0 is a parameter in interval $[0, 1]$, $\eta_{i,p}$ is the heuristic value for assigning VM i to PM p (4.18), $\tau_{i,p}$ is the current pheromone value associated with the $\langle i, p\rangle$ pair (4.23), β is a non-negative parameter that determines the relative importance of pheromone amount versus heuristic value in the decision rule, and S is a random variable selected according to the probability distribution given by (4.22) below. $FV_k(p)$ defines the list of feasible VMs for ant k to assign to PM p (i.e., VMs that are not assigned to any PM yet and do not violate the resource capacity constraint of p given by (4.7):

$$FV_k(p) = \left\{ i \middle| \sum_{p \in PMS} x_{i,p} = 0 \bigwedge U_p^l + D_i^l \leq C_p^l \text{ for } \forall l \in RCS \right\}. \tag{4.21}$$

The pseudo-random proportional rule works as follows: when $q \leq q_0$, then the $\langle i, p\rangle$ pair that results in the highest $\tau_{i,p} \times [\eta_{i,p}]^\beta$ value is chosen as the solution component (termed exploitation), otherwise a VM i is chosen with probability $P_k(i, p)$ using the following *random-proportional rule* (termed exploration):

$$P_k(i,p) = \begin{cases} \frac{\tau_{i,p} \times [\eta_{i,p}]^\beta}{\sum_{u \in FV_k(p)} \tau_{u,p} \times [\eta_{u,p}]^\beta}, & \text{if } i \in FV_k(p); \\ 0, & \text{otherwise.} \end{cases} \tag{4.22}$$

In the above random-proportional rule, the pheromone value for a VM-PM pair ($\tau_{i,p}$) is multiplied by the corresponding heuristic value ($\eta_{i,p}$) in order to favor the VMs which cause higher PM resource utilization (both for balanced and overall utilization) and which have greater values of pheromone.

Global Pheromone Update

In order to favor the solution components of the GBS for subsequent iterations and simulate pheromone evaporation, the global pheromone update rule is applied on the pheromone values for each $\langle i, p \rangle$ pair according to the following equation:

$$\tau_{i,p} \leftarrow (1 - \delta) \times \tau_{i,p} + \delta \times \Delta\tau_{i,p} \tag{4.23}$$

where δ is the global pheromone decay parameter ($0 < \delta < 1$) and $\Delta\tau_{i,p}$ is the pheromone reinforcement applied to each of the $\langle i, p \rangle$ pairs that constitute the GBS solution, and its value depends on the quality of the solution in terms of PE:

$$\Delta\tau_{i,p} = \begin{cases} PE_{GBS}, & \text{if } \langle i, p \rangle \in GBS; \\ 0, & \text{otherwise.} \end{cases} \tag{4.24}$$

4.5 Performance Evaluation

This section presents the performance evaluation of the proposed AVVMP VM placement algorithm compared to other energy and utilization-aware VM cluster placement approaches from the existing literature. Because of the lack of access to large-scale testbeds and real Cloud infrastructures, as well as the ease of reproducibility, the evaluation is conducted on simulation-based experimentation. The simulated environment models the data center as a cluster of homogeneous PMs with three physical resource capacities: CPU, main memory, and network I/O. The VM cluster to deploy is modeled as a collection of VMs with synthetically-generated resource demands for CPU, memory, and network I/O.

4.5.1 Algorithms Compared

The following existing studies from the literature are compared to the proposed AVVMP algorithm:

- *Max-Min Ant System-based VM Consolidation (MMVMC)*: An adapted version of Max-Min Ant System (MMAS) metaheuristic for solving consolidated VM cluster placement problems [46].

- *Vector Algebra-based Greedy Algorithm (VecGrdy)*: A greedy algorithm for solving consolidation that uses vector algebra for mean estimation of multi-dimensional resources [88].

- *Volume-based First Fit Decreasing (FFFVol)*: A modified version of the FFD algorithm that uses volume-based mean estimator [127].

- *First Fit Decreasing (FFFL1) based on L1-norm*: Another version of the modified FFD algorithm often used as a baseline algorithm for packing problems [44].

Both MMVMC and FFDL1 use mean estimator based on L1-norm to capture the multi-dimensional aspect of the VM resource demands to a single scalar value. VecGrdy, on the other hand, although it captures the aspect of balanced resource utilization in multiple dimensions, employs a greedy approach, and therefore, is not guaranteed to explore the search space effectively and approach close to near-global optimal solutions.

4.5.2 Simulation Setup

The simulated data center consists of a cluster of homogeneous PMs and the VM resource demand for each resource type is expressed in percentage of the total resource capacity of PM. VM resource demands are generated using reference values, where $Ref = z\%$ indicates that each randomly-generated VM resource demand D^l falls in the interval $[0, 2z]$ for $l \in \{CPU, MEM, NIO\}$ [3]. As explained in Section 4.1, Clouds host various types of applications ranging from CPU-intensive to memory-bound and bandwidth-hungry, and offer different categories of VMs with different amounts for available resource types such as high-CPU, high-memory, etc. In order to capture these factors in the experiments, random VM resource demands are generated across three dimensions: CPU, memory, and network I/O. This approach can generate VMs with uniform resource demands, as well as VMs with complementary resource demands in different dimensions, and thus is aligned with the VM resource pattern presented in Table 4.1.

Since Clouds deploy high-end servers and try to host as many VMs as possible in each active server to increase resource utilization, the simulation is conducted for the scenarios where expected average PE would be more than 2, otherwise there would not be much scope for VM consolidation, which will result in little benefit of using specialized algorithms. Therefore, reference values of $Ref = 5\%$, 10%, 15%, 20%, 25%, and 30% are

considered, with their corresponding expected average PE of 20, 10, 6.7, 5, 4, and 3.3, respectively.

The AVVMP parameters used for the performance evaluation are determined by sensitivity analysis presented in the remaining part of this section. Parameters for the other algorithms are taken as reported in the respective papers. For the purpose of simulation, E_{idle} and E_{full} were set to 162 watts and 215 watts, respectively, as used by Gao *et al.* [50]. The simulation was conducted through 100 independent simulation runs and each run was repeated 100 times. The results were generated after taking their average.

AVVMP Parameter Sensitivity Analysis

The optimal values of the AVVMP algorithm parameters used for the performance evaluation were measured by rigorous parameter sensitivity analysis in the preliminary phases of the experiment and the results are summarized in Table 4.4. Figure 4.7 presents changes of OF value (f_1) (4.6) across different AVVMP parameters.

As can be seen from Figure 4.7(a), the number of ant agents ($nAnts$) for AVVMP is gradually increased from 1 to a maximum of 10 and the minimum value of the OF f_1 is first achieved when $nAnts = 5$ and further increase of the number of ants does not improve the solution quality. Although it may apparently seem that with the increase in the number of ants for the search process, the solution quality would improve; this may not necessarily be the case for ACO-based algorithms [35]. A natural explanation for this behavior can be derived from the different aspects of the ACO metaheuristic itself. Due to the *Stigmergy* [39] property of this multi-agent-based scheme (i.e., stimulation of the agents by their combined performance), ACO-based algorithms have the added advantage of *Synergistic Effect* [36] (i.e., an effect arising from multiple agents or entities that produces an effect greater than the sum of their individual effects) on the quality of the solutions, which is achieved by the shared pheromone information in AVVMP algorithm. As a consequence, a small number of ants can produce high quality solutions. Having said that, the search process for the ant agents is guided by the heuristic component of the ACO-based algorithms. The phenomenon that AVVMP algorithm finds best solution with only 5 ants and further increase in the number of ants does not improve the solution quality persuasively indicates the effectiveness of the proposed vector algebra-based balanced resource utilization capturing technique (4.11), which is incorporated into the proposed heuristic formulation (4.18).

Table 4.4: ACS parameters used in AVVMP algorithm evaluation

$nAnts$	$nCycleTerm$	β	δ	q_0	ω
5	5	2	0.5	0.8	0.5

A similar result has been found for $nCycleTerm$ (Figure 4.7(b)), where starting from 1, increasing the number of tries in cases of no improvement in solution quality effectively enables AVVMP to find a better solution. However, increasing $nCycleTerm$ beyond 5 does not provide any further improvement. Figure 4.7(c) shows how β influences the OF by increasing the relative importance of heuristic value over pheromone level in the probabilistic decision rule (4.20). For $\beta < 2$, pheromone level has more influence compared to heuristics value and, therefore, ants in AVVMP suffer from early stagnation. AVVMP achieves the best solution for $\beta = 2$ and for higher values, the heuristic part dominates the probabilistic decision rule and as a result, AVVMP is affected more by its greedy component, and, as a consequence, the solution quality degrades.

The OF values achieved for different values of global pheromone decay parameter δ are shown in Figure 4.7(d). For $\delta = 0.5$, AVVMP achieves the best result for the OF f_1. However, for $\delta < 0.5$, both the global pheromone delay is slower and the pheromone deposition for the solution components of the so-far-best solution is lower, and, therefore, ants fail to converge to a better solution. On the other hand, for $\delta > 0.5$, the global pheromone delay is quicker and the solution components of the so-far-best solution have higher deposition of pheromone, and as a result, the ants experience early stagnation. Figure 4.7(e) presents the solution quality for various values of q_0 parameter which determines the relative importance of the exploitation of already-found good solution components versus the exploration of new solution components. As the figure suggests, for $q_0 < 0.8$, a greater amount of exploration impedes timely convergence to good solutions and AVVMP has the best result for $q_0 = 0.8$. However, going beyond 0.8 results in too much exploitation and, as a result, causes stagnation. Finally, Figure 4.7(f) shows the sensitivity of parameter ω that determines the relative importance between balanced and overall resource utilization of PMs for computing the heuristic value in (4.18). As the result implies, the best OF value is achieved for $\omega = 0.5$ when equal weight is given to balanced and overall utilization, and the solution quality worsens when more importance is given to either of the two factors compared to the other.

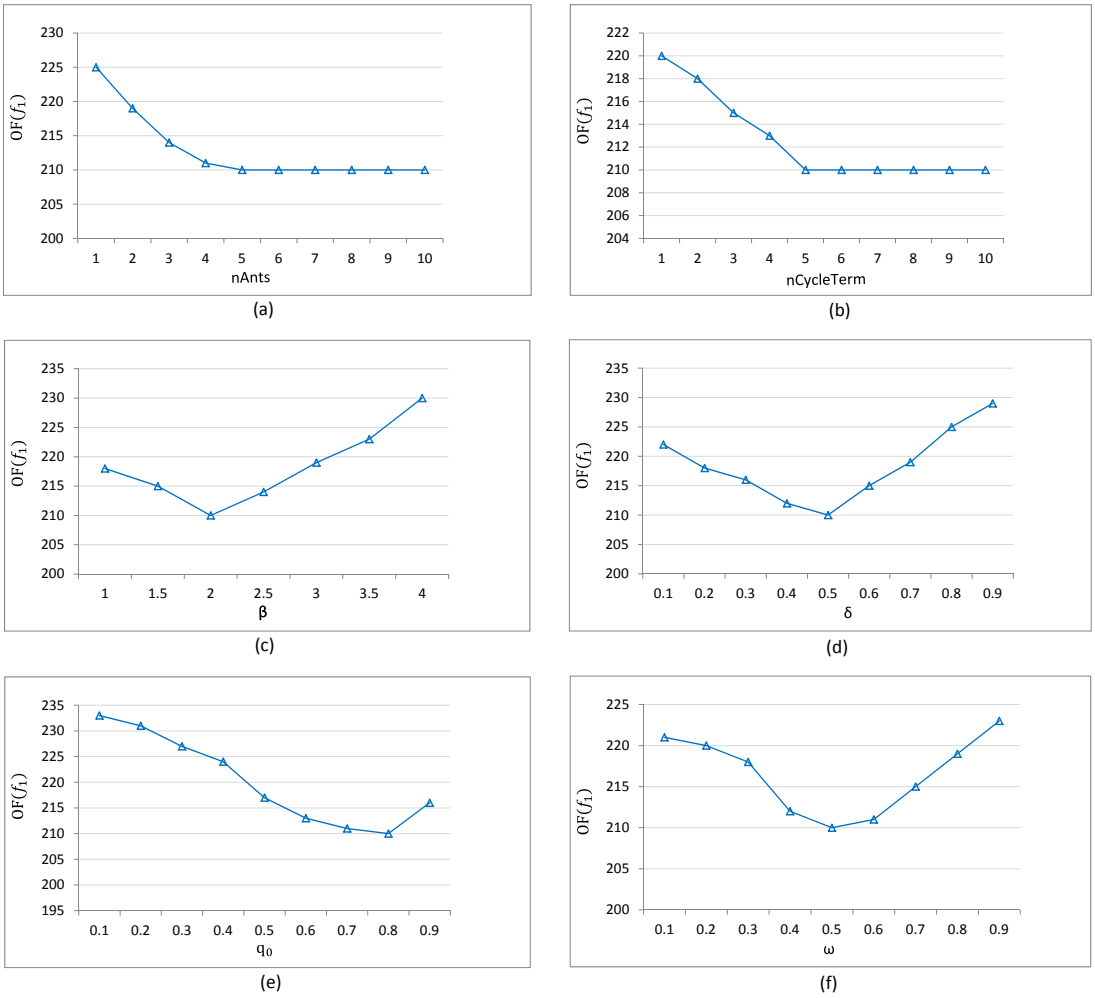

Figure 4.7: AVVMP parameter sensitivity analysis.

Performance Evaluation Metrics

In order to assess the quality of the consolidated VM placement decisions produced by the evaluated algorithms, the number of PMs needed (i.e., active PMs) to place the VM cluster is considered as one of the performance metrics— the reduced number of PMs results in reduced capital cost as well as reduced physical space to host the VM cluster. VM packing efficiency PE (4.17) is another performance metric that indicates the VM consolidation capability of each placement algorithm. The power consumption of the active PMs hosting the VMs computed according to the overall power consumption model (4.15) is another important performance metric that represents the energy-related cost incurred by the placement algorithms and hence, the level of energy efficiency for each placement approach. High utilization of server resources is another goal of multi-objective placement algorithms and VM placement that reduces the total residual (unused) resources in active

servers across CPU, memory, and network I/O effectively increases server resource utilization. The overall resource wastage metric is measured as the summation of resource wastage for all active PMs (4.13). Finally, the solution computation time for a VM placement algorithm is another important performance metric that determines the feasibility and scalability of the algorithm, specially for online decision making scenarios.

Simulation Environment

The algorithms are implemented in Java (JDK and JRE version 1.7.0) and the simulation is conducted on a Dell Workstation (Intel Core i5-2400 3.10 GHz CPU (4 cores), 4 GB of RAM, and 240 GB storage) hosting Windows 7 Professional Edition.

4.5.3 Scaling VM Cluster Size

In order to assess the performance of the placement algorithms with increasing size of the problem, in this part of the experiment the VM cluster size (N_v) was initially set to 100 and gradually increased up to 2100, each time adding 400 more VMs to the cluster. The Ref value for VM resource demand was set to 15%, so that on average 5 to 6 VM could be hosted by a single PM.

Table 4.5 details the values of various performance metrics for the simulated VM placement algorithms for each of the six VM cluster sizes. The results clearly indicate the superior performance of AVVMP compared to other placement algorithms for all cluster sizes: it requires the lowest number of PMs to host the VMs in the clusters, achieves the highest VM packing efficiency, and incurs the least power consumption in the data center.

In order to better visualize the improvement achieved by AVVMP compared to other approaches in terms of energy efficiency, Figure 4.8 depicts a bar chart representation of the percentage of improvement in overall power consumption of AVVMP over its competitors. As the chart shows, VM placement decisions produced by AVVMP result in 11-12% less power consumption compared to FFDL1 and FFDVol across all cluster sizes, whereas it is 4-8% for VecGrdy and 3-4% for MMVMC. This is because both FFDL1 and FFDVol are simple greedy heuristics, whereas VecGrdy places VMs focusing on balanced resource utilization and MMVMC utilizes the ACO metaheuristic to refine the solution through multiple iterations. As a result, VecGrdy and MMVMC produce placement plans with fewer PMs and thus, achieve higher energy efficiency.

Table 4.5: Placement performance metrics across various VM cluster sizes (N_v)

# of VM	Algorithm	# Active PM	Achieved PE	Power Con. (Watt)
100	MMVMC	18	5.56	3769.66
	VecGrdy	19	5.26	3931.66
	FFDVol	20	5.00	4093.66
	FFDL1	20	5.00	4093.66
	AVVMP	17	5.88	3607.66
500	MMVMC	85	5.88	17835.35
	VecGrdy	87	5.75	18159.35
	FFDVol	95	5.26	19455.35
	FFDL1	96	5.21	19617.35
	AVVMP	82	6.10	17349.35
900	MMVMC	153	5.88	32112.65
	VecGrdy	156	5.77	32598.65
	FFDVol	172	5.23	35190.65
	FFDL1	173	5.20	35352.65
	AVVMP	148	6.08	31302.65
1300	MMVMC	220	5.91	46035.23
	VecGrdy	230	5.65	47655.23
	FFDVol	246	5.28	50247.23
	FFDL1	248	5.24	50571.23
	AVVMP	210	6.19	44415.23
1700	MMVMC	288	5.90	60141.85
	VecGrdy	301	5.65	62247.85
	FFDVol	321	5.30	65487.85
	FFDL1	319	5.33	65163.85
	AVVMP	273	6.23	57711.85
2100	MMVMC	353	5.95	73770.52
	VecGrdy	370	5.68	76524.52
	FFDVol	394	5.33	80412.52
	FFDL1	397	5.29	80898.52
	AVVMP	340	6.18	71664.52

Figure 4.9 shows a bar chart representation of the overall normalized resource wastage (4.13) of the active PMs needed by each placement algorithm for different VM cluster sizes. It is evident from the chart that AVVMP significantly reduces the resource wastage compared to other algorithms: 57-71% over FFDL1, 57-72% over FFDVol, 36-59% over VecGrdy, and 26-44% over MMVMC. This is because AVVMP tries to improve the overall resource utilization with preference to consolidating VMs with complementary resource demands in each server, and thus reduces resource wastage across different resource dimensions. Another pattern can be observed from the figure: the resource wastage reduction of AVVMP over other algorithms improves with larger cluster sizes. This is again attributed to the fact that with a higher number of VMs, AVVMP has more flexibility to match VMs

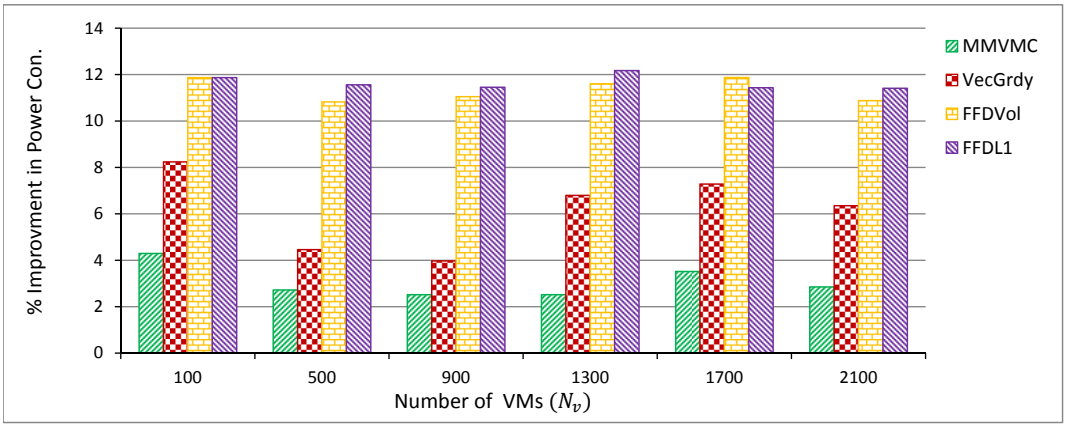

Figure 4.8: Percentage of improvement of AVVMP in power consumption over other approaches across different VM cluster sizes (best viewed in color).

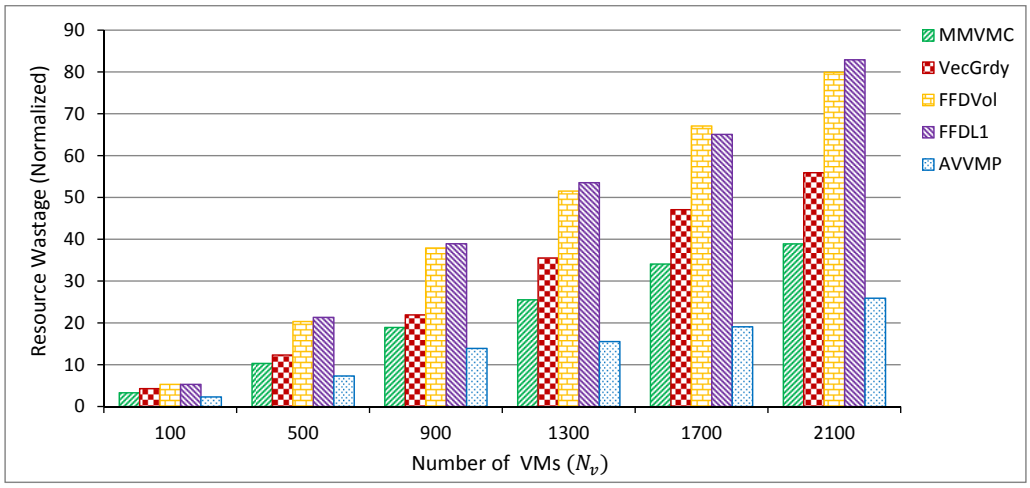

Figure 4.9: Total resource (normalized) wastage of active PMs for placement algorithms across different VM cluster sizes (best viewed in color).

with complementary resource demands to pack them more efficiently in order to reduce residual server resources across multiple resource dimensions.

4.5.4 Scaling VM Resource Demands

In this part of the experiment, the reference value for the VM resource demands (Ref) was initially set to 5% and gradually increased up to 30%, each time with an increase of 5. Increase of Ref broadens the range of randomly-generated VM resource demands, and therefore results in more diverse VMs in terms of resource demands across multiple dimensions. Therefore, larger values of Ref will cause VM clusters to have larger as well as smaller VMs. For all Ref values, the VM cluster size was fixed to 1300.

Table 4.6: Placement performance metrics across various resource demands (Ref)

Ref	Algorithm	# Active PM	Achieved PE	Power Con. (Watt)
5%	MMVMC	69	18.84	14643.08
	VecGrdy	77	16.88	15939.08
	FFDVol	84	15.48	17073.08
	FFDL1	84	15.48	17073.08
	AVVMP	67	19.40	14319.08
10%	MMVMC	142	9.15	29934.15
	VecGrdy	151	8.61	31392.15
	FFDVol	165	7.88	33660.15
	FFDL1	165	7.88	33660.15
	AVVMP	137	9.49	29124.15
15%	MMVMC	220	5.91	46035.23
	VecGrdy	230	5.65	47655.23
	FFDVol	246	5.28	50247.23
	FFDL1	248	5.24	50571.23
	AVVMP	210	6.19	44415.23
20%	MMVMC	303	4.29	62946.31
	VecGrdy	319	4.08	65538.31
	FFDVol	330	3.94	67320.31
	FFDL1	328	3.96	66996.31
	AVVMP	291	4.47	61002.31
25%	MMVMC	380	3.42	78885.38
	VecGrdy	403	3.23	82611.38
	FFDVol	400	3.25	82125.38
	FFDL1	401	3.24	82287.38
	AVVMP	361	3.60	75807.38
30%	MMVMC	486	2.67	99522.46
	VecGrdy	543	2.39	108756.46
	FFDVol	520	2.50	105030.46
	FFDL1	512	2.54	103734.46
	AVVMP	467	2.78	96444.46

Table 4.6 shows various performance metrics for AVVMP and competitor placement policies for six Ref values. It is obvious from the data that AVVMP outperforms other algorithms for all the cases. It also shows that AVVMP achieves PE near the expected average values. Furthermore, it can be observed that packing efficiency drops with the increase of Ref value for all algorithms; this is because higher Ref values cause greater number, of larger VMs to be generated on average, which reduces the packing efficiency of the PMs.

Figure 4.10 shows the improvements in overall power consumption of AVVMP over other approaches for different Ref values. VM placement decisions produced by AVVMP

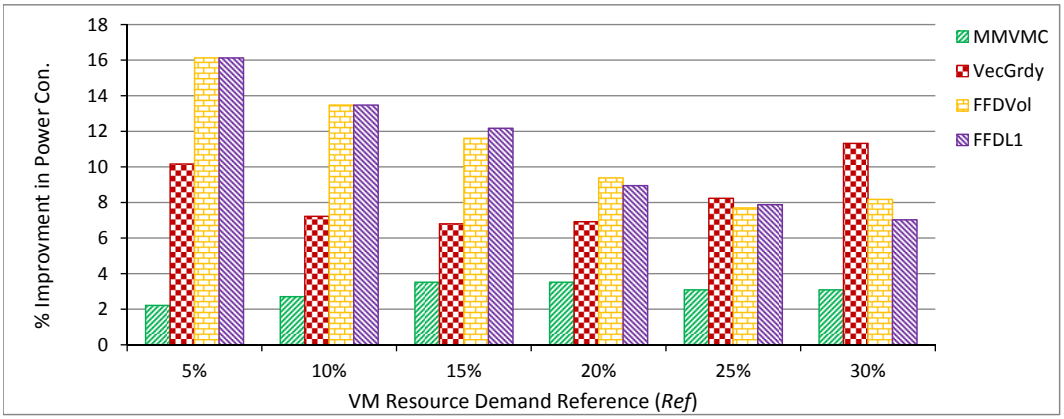

Figure 4.10: Percentage of improvement of AVVMP in power consumption over other approaches across different demand levels of VMs (best viewed in color).

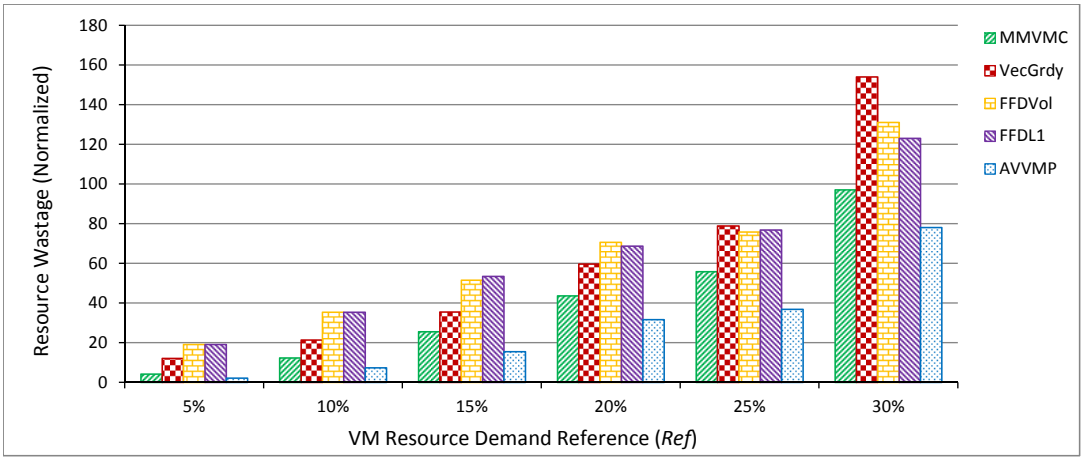

Figure 4.11: Total resource (normalized) wastage of active PMs for placement algorithms across different demand levels of VMs (best viewed in color).

result in 7-16% less power consumption compared to FFDL1 and FFDVol across different VM resource demand levels, whereas the reduction is 7-11% for VecGrdy and 2-4% for MMVMC. One interesting observation from this chart is that AVVMP achieves comparatively better performance over MMVMC and VecGrdy for larger reference values (i.e., larger VM sizes), whereas it achieves comparatively better performance over FFD-based algorithms for smaller reference values (i.e., smaller VM sizes). The reason is that metaheuristic-based solutions have more flexibility to refine the solutions for smaller VM sizes (i.e., when higher numbers of VMs can be packed in a single PM) compared to larger VM sizes. On the other hand, for larger reference values, FFD-based greedy solutions achieve comparatively higher overall resource utilization and need relatively fewer active PMs.

The overall resource wastage (normalized) for the placement algorithms for different VM sizes is shown in Figure 4.11. From the figure it can be seen that, AVVMP incurs much less resource wastage compared to other approaches: 37-89% over FFDL1, 40-89% over FFDVol, 47-82% over VecGrdy, and 20-48% over MMVMC. This is because AVVMP tries to improve the overall resource utilization with preference to consolidating VMs with complementary resource demands in each server, and thus reduces resource wastage across different resource dimensions. Another pattern which can be observed from the figure is that the resource wastage reduction of AVVMP over other algorithms is higher for smaller VM sizes. This is attributed to the fact that when VMs are of smaller sizes (i.e., for smaller Ref values), each PM can accommodate a higher number of VMs and, therefore, AVVMP has more flexibility to choose VMs with complementary resource demands to consolidate them more efficiently with the goal of minimizing residual server resources across different resource dimensions.

4.5.5 AVVMP Decision Time

In order to assess AVVMP for time complexity, a simulation was conducted to measure VM cluster placement computation time for larger cluster sizes and various VM sizes, and the results are plotted in Figure 4.12.

It can be observed that the computation time increases non-linearly with the number of VMs in the cluster and the growth is smooth for each of the different Ref values, even though the search space for the problem grows exponentially with the number of VMs. Moreover, for the same number of VMs, AVVMP requires relatively more time to compute placement decisions for larger Ref values. This is to be expected, since for larger Ref values, higher numbers of larger VMs are generated and to accommodate the larger VMs more PMs are needed. As a result, this increases $nActivePM$, which contributes to solution computation time.

Furthermore, as the figure suggests, for a cluster of 4000 VMs AVVMP requires a maximum of 30 seconds to compute optimized VM placement decisions and this time is much less for smaller clusters, such as 1.4 seconds for 1000 VMs. In addition, since AVVMP utilizes ACO, a multi-agent-based computational method, there is potential for parallel implementation [98] of AVVMP, where the ant agents can run in parallel in multiple Cloud nodes in order to reduce the solution computation time significantly.

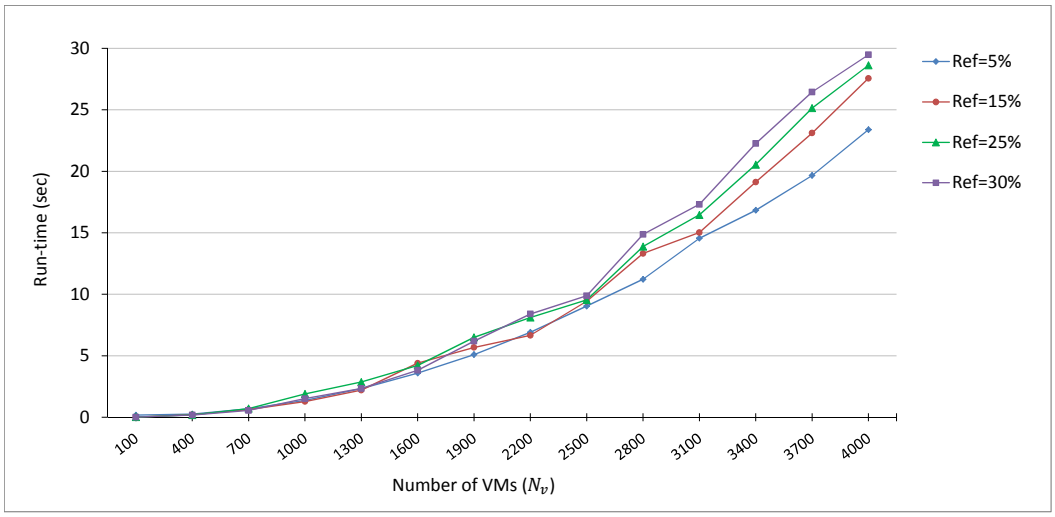

Figure 4.12: AVVMP's placement decision time for large problem instances (best viewed in color).

4.6 Summary and Conclusions

The rapidly increasing energy costs of data centers is emerging as a great challenge for infrastructure management, specially for large-scale data centers such as the Clouds. Virtualization technologies provide efficient methods to provision computing resources in the form of VMs so that multiple VMs can share physical resources in a time-sharing (e.g., CPU) and space-sharing (e.g., memory) manner, and thus consolidate VMs to ensure higher resource utilization and lower energy consumption. However, the consolidation of VMs with single resource demands is already an $\mathcal{NP}$–*hard* problem and multiple resource demands increase the complexity of the solution approaches. This chapter has presented several motivating factors for consolidated VM placement in large-scale virtualized data centers and several aspects of server resource utilization and consolidation. It has proposed mathematical models to formally define the consolidated VM cluster placement problem and techniques for capturing balanced server resource utilization across multiple resource dimensions. It has further proposed a metaheuristic-based consolidated VM cluster placement algorithm that optimizes both server energy consumption and resource utilization.

Simulation-based performance evaluation has been presented by comparing the proposed technique with some of the techniques proposed in the recent literature. The results suggest that the proposed method outperforms other methods by significantly reducing

both data center energy consumption and server resource wastage. Finally, evaluation of time complexity of solution computation and arguments on the feasibility and effectiveness of the algorithm for large data centers have also been presented.

The VM placement approach proposed in this chapter does not take into consideration the inter-VM communication traffic while making placement decisions, because it is assumed that the VMs in the cluster do not have communication dependency among themselves. The next chapter addresses the problem of online, network-aware VM cluster placement where VMs have communication correlations among each other. In particular, the VMs are considered to be part of composite applications accompanied by their associated data components with defined communication relationships with the VMs. The overall placement decisions consider the simultaneous placement of VMs and data components with the objective of localizing the data traffic in order to reduce network overhead on the data center network.

Chapter 5

Network-aware Virtual Machine Placement

This chapter addresses the problem of online, network-aware placement of Virtual Machines (VMs) and associated data blocks, comprising composite Cloud applications, in virtualized data centers. The placement problem is formally defined as an optimization problem which is shown to be $\mathcal{NP}-hard$. As a solution, a fast greedy heuristic is proposed for network-efficient, simultaneous placement of VMs and data blocks of a multi-component application with the goal of minimizing the network traffic incurred due to the placement decision, while respecting the computing, network, and storage capacity constraints of data center resources. The proposed placement scheme strives to reduce the distance that data packets need to travel in the data center network and eventually help in localizing network traffic and reducing communication overhead in upper-layer network switches. Extensive performance evaluation across several scaling factors reveals that the proposed approach outperforms the competitor algorithms in all performance metrics by reducing the network cost by up to 67%, and network usage of core and aggregation switches by up to 84% and 50%, respectively.

5.1 Introduction

The previous chapter has presented an online *Virtual Machine* (VM) cluster placement approach with the goal of minimizing resource wastage and energy consumption. The VMs in the cluster are considered to be independent from each other in terms of mutual traffic communication. Complementary to the previous approach, this chapter addresses

the problem of online, network-aware VM cluster placement in virtualized data centers along with associated data components where VMs are interrelated to each other and to the data components based on mutual communication requirements. Such VM clusters and their data components are modeled as composite applications and the placement of such multi-component applications is formally defined as an optimization problem. The proposed greedy heuristic performs simultaneous placement of VMs and data blocks with the aim of reducing the network overhead on the data center network infrastructure due the placement decision, while at the same time respecting the computing, network, and storage capacity constraints of the data center resources. The application model and solution scheme are developed as generic and not restricted to any particular application type or data center architecture.

As presented in Chapter 1, after the emergence of Cloud Computing, data centers are facing rapid growth of network traffic and a large portion of this traffic is constituted of the data communication within the data center. Cisco's Global Cloud Index [25], an annual assessment and future projection of global network traffic trends, shows that the Cloud traffic will dominate the global data center traffic flow in the near future. It forecasts that, while data center network traffic will triple from 2014 to 2019, global Cloud traffic will more than quadruple within the same timeframe. Moreover, it projects that the total volume of global data center traffic will grow steadily from 3.4 zettabytes in 2014 to 10.4 zettabytes by 2019 and three-quarters of this traffic will be generated due to the data communication within the data centers (Figure 5.1).

This huge amount of intra-data center traffic is primarily generated by the application components that are coupled with each other, for example, the computing components of a composite application (e.g., MapReduce) writing data to the storage array after it has processed the data. This large growth of data center traffic may pose serious scalability problems for the wide adoption of Cloud Computing. Moreover, as a result of the continuously rising popularity of social networking sites, e-commerce, and Internet-based gaming applications, large amounts of data processing have become an integral part of Cloud applications. Furthermore, scientific processing, multimedia rendering, workflow, and other massive parallel processing and business applications are being migrated to the Clouds due to the unique advantages of their high scalability, reliability, and pay-per-use business model. Furthermore, the recent trend in Big Data Computing using Cloud

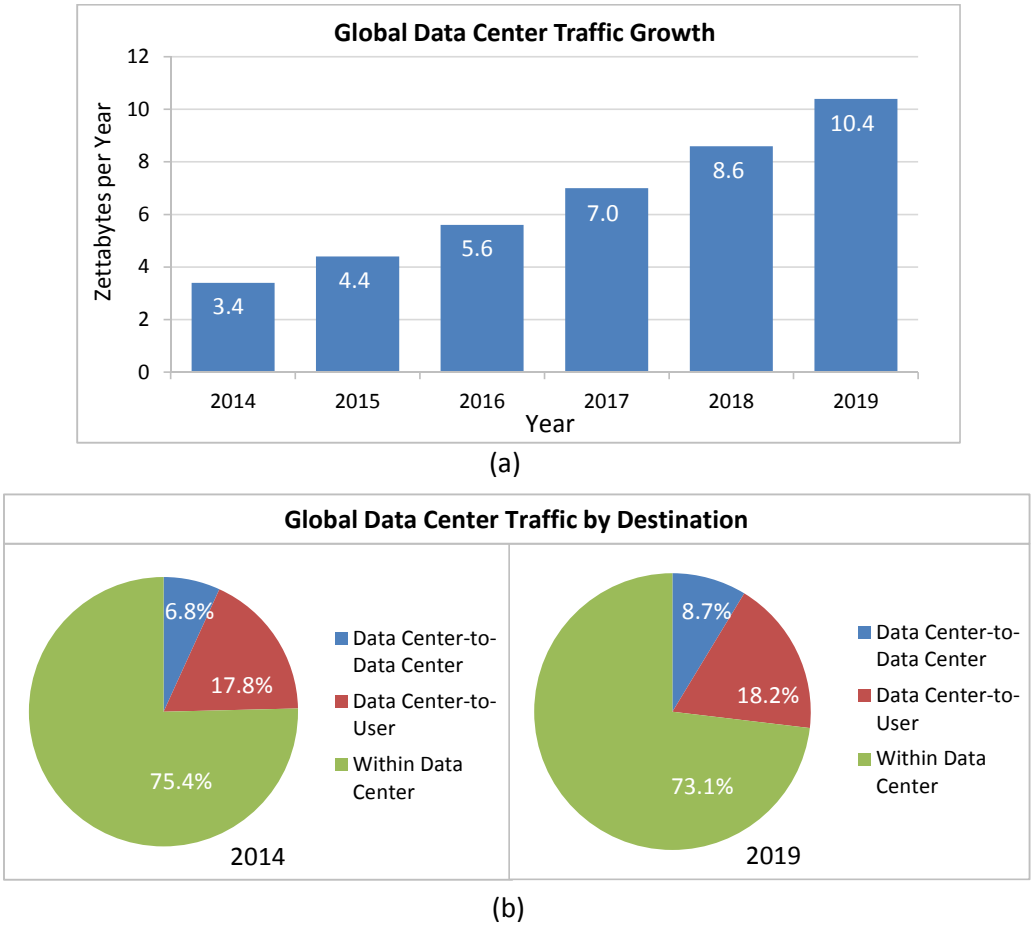

Figure 5.1: Global data center traffic growth: (a) by year, (b) by destination (source: Cisco Inc., 2015) (best viewed in color).

resources [8] is emerging as a rapidly growing factor contributing to the rise of network traffic in Cloud data centers.

One of the key technological elements that has paved the way for the extreme success of Cloud Computing is virtualization. Modern data centers leverage various virtualization technologies (e.g., machine, network, and storage virtualization) to provide users with an abstraction layer that delivers a uniform and seamless computing platform by hiding the underlying hardware heterogeneity, geographic boundaries, and internal management complexities [133]. By the use of virtualization, physical server resources are abstracted and shared through partial or full machine simulation by time-sharing, and hardware and software partitioning into multiple execution environments, known as *Virtual Machines* (VMs), each of which runs as a complete and isolated system. This allows dynamic sharing and reconfiguration of physical resources in Cloud infrastructures that make it possible to run multiple applications in separate VMs with different performance metrics. It also facilitates Cloud providers to improve utilization of physical servers through VM

multiplexing [84] and multi-tenancy, i.e., simultaneous sharing of physical resources of the same server by multiple Cloud customers. Furthermore, it enables on-demand resource pooling through which computing (e.g., CPU and memory), network, and storage resources are provisioned to customers only when needed [73]. By utilizing these flexible features of virtualization for provisioning physical resources, the scalability of data center network can be improved through the minimization of network load imposed due to the deployment of customer applications.

On the other hand, modern Cloud applications are dominated by multi-component applications such as multi-tier applications, massive parallel processing applications, scientific and business workflows, content delivery networks, and so on. These applications usually have multiple computing and associated data components. The computing components are usually delivered to customers in the form of VMs, such as Amazon EC2 Instances[1], where the data components are delivered as data blocks, such as Amazon EBS[2]. The computing components of such applications have specific service roles and are arranged in layers in the overall structural design of the application. For example, large enterprise applications are often modeled as 3-tier applications: the presentation tier (e.g., web server), the logic tier (e.g., application server), and the data tier (e.g., relational database) [115]. The computing components (VMs) of such applications have specific communication requirements among themselves, as well as with the data blocks that are associated with these VMs (Figure 5.2). As a consequence, the overall performance of such applications heavily depends on the communication delays among the computing and data components. From the Cloud providers' perspective, optimization of network utilization of data center resources is tantamount to profit maximization. Moreover, efficient bandwidth allocation and reduction of data packet hopping through network devices (e.g., switches or routers) reduce the overall energy consumption of network infrastructure. On the other hand, Cloud consumers' concern is to receive guaranteed Quality of Service (QoS) of the delivered virtual resources, which can be assured through appropriate provisioning of requested resources.

Given the issue of the sharp rise in network traffic in data centers, this chapter deals with the scalability of data center networks using a traffic-aware placement strategy of

[1]Amazon EC2 - Virtual Server Hosting, 2016. https://aws.amazon.com/ec2/
[2]Amazon Elastic Block Store (EBS), 2016. https://aws.amazon.com/ebs/

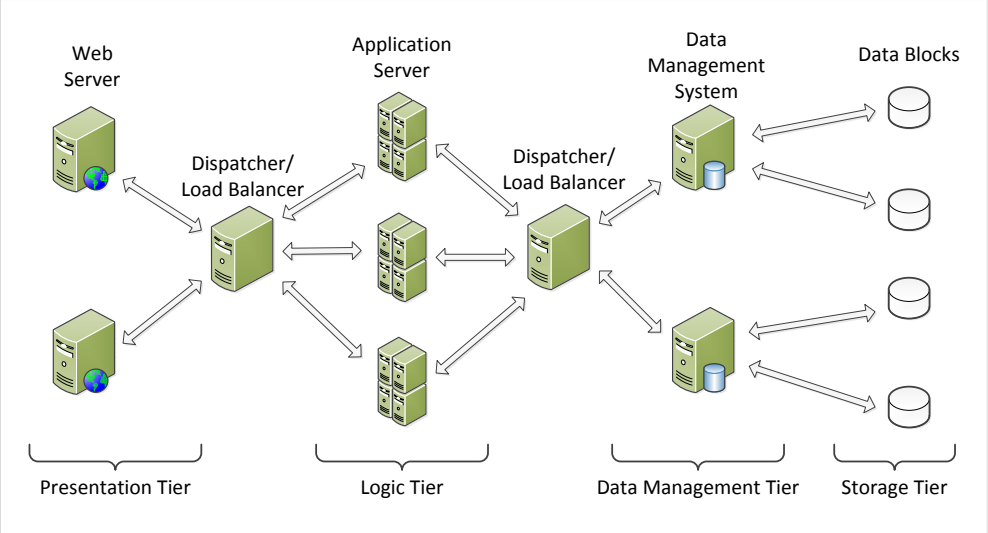

Figure 5.2: Multi-tier application architecture.

multi-component, composite application (in particular, VMs and data blocks) in a virtualized data center in order to optimize the network traffic load incurred due to placement decisions. The placement decisions can be made during the application deployment phase in the data center. VM placement decisions focusing on other goals than network efficiency, such as energy consumption reduction [12, 46], and server resource utilization [48, 50], often result in placement decisions where VMs with high mutual traffic are placed in host servers with high mutual network cost. In contrast, this chapter focuses on placing mutually communicating components of applications (such as VMs and data blocks) in data center components (such as physical servers and storage devices) with lower network cost so that the overall network overhead due to the placement is minimal. With this placement goal, the best placement for two communicating VMs would be in the same server, where they can communicate through memory copy, rather than using the physical network links.

Moreover, advanced hardware devices with combined capabilities are opening new opportunities for efficient resource allocation focusing on application needs. For example, Dell PowerEdge C8000 servers are equipped with CPU, GPU, and storage components that can work as multi-function devices. Combined placement of application components with high mutual traffic (e.g., VMs and their associated data components) in such multi-function servers will effectively reduce data transfer delay, since the data accessed by the VMs reside in the same devices. Similar trends are found in high-end network switches (e.g., Cisco MDS 9200 Multiservice switches) that are equipped with additional built-in

processing and storage capabilities. Reflecting these technological development and multi-purpose devices, this research work considers a generic approach in modeling computing, network, and storage elements in a data center so that placement algorithms can make efficient decisions for application component placement in order to achieve the ultimate goal of network cost reduction.

Most of the existing studies on network-aware VM placement and relocation primarily focus on run-time reconfiguration of VMs in the data center with the purpose of reducing the traffic overhead [33, 85, 106, 114, 126]. These works suggest the use of the VM live migration technique in order to achieve the intended optimization. However, given the fact that VM live migrations are costly operations [78], the above-mentioned VM relocation strategies overlook the impact of necessary VM migrations and reconfiguration on hosted applications, physical servers and network devices. Complementary to these studies, the research presented in this chapter tackles the problem of network-efficient, online placement of composite applications consisting of multiple VMs and associated data components along with inter-component communication patterns in a data center consisting of both computing servers and storage devices. The proposed solution does not involve VM migration since the placement decision is taken during the application deployment phase.

An online VM placement problem, particularly focusing on a data center designed based on the PortLand network topology [91], is presented in [51] and two heuristics are proposed for reducing network utilization at the physical layer. However, this work does not involve any data component for VM-cluster specification, which is a rapidly increasing trend in modern, multi-component Cloud applications. In contrast, the composite application and data center models, as well as the application placement strategy proposed in this chapter, are generic and not restricted to any particular application or data center topology. Some data location-aware VM placement studies can be found in [100] and [72], however, these studies model the applications as a single instance of VM, which is an oversimplified view of today's Cloud or Internet applications, that are mostly composed of multiple computing and storage entities in multi-tier structure with strong communication correlations among the components. Based on these insights, this chapter considers a much wider VM communication model by considering the placement of application

environments, each involving a number of VMs and associated data blocks with sparse communication links between them.

This research addresses the allocation, specifically the online placement of composite application components (modeled as an *Application Environment*), requested by the customers to be deployed in Cloud data center focusing on network utilization, with consideration of the computing, network, and storage resources capacity constraints of the data center. In particular, this chapter makes the following **key contributions**:

1. A *Network-aware Application environment Placement Problem* (NAPP) is formally defined as a combinatorial optimization problem with the objective of network cost minimization due to placement. The proposed data center and application environment models are generic and not restricted to any specific data center topology and application type or structure.

2. Given the resource requirements and structure of the application environment to be deployed, and the information on the current resource state of the data center, this research work proposes a *Network- and Data location-aware Application environment Placement* (NDAP) scheme, a greedy heuristic that generates mappings for simultaneous placement of the computing and data components of the application into the computing and storage nodes of the data center, respectively, focusing on the minimization of network traffic, while respecting the computing, network, and storage capacity constraints of data center resources. While making placement decisions, NDAP strives to reduce the distance that data packets need to travel in the data center network, which in turn, helps to localize network traffic and reduces communication overhead in the upper-layer network switches.

3. Performance evaluation of the proposed approach is conducted through extensive simulation-based experimentation across multiple performance metrics and several scaling factors. The results suggest that NDAP successfully improves network resource utilization though the efficient placement of application components and significantly outperforms the algorithms compared across all performance metrics.

The remainder of this chapter is organized as follows. Section 5.2 formally defines the multi-component application placement problem (NAPP) as an optimization problem, along with the associated mathematical models. The proposed network-aware, application

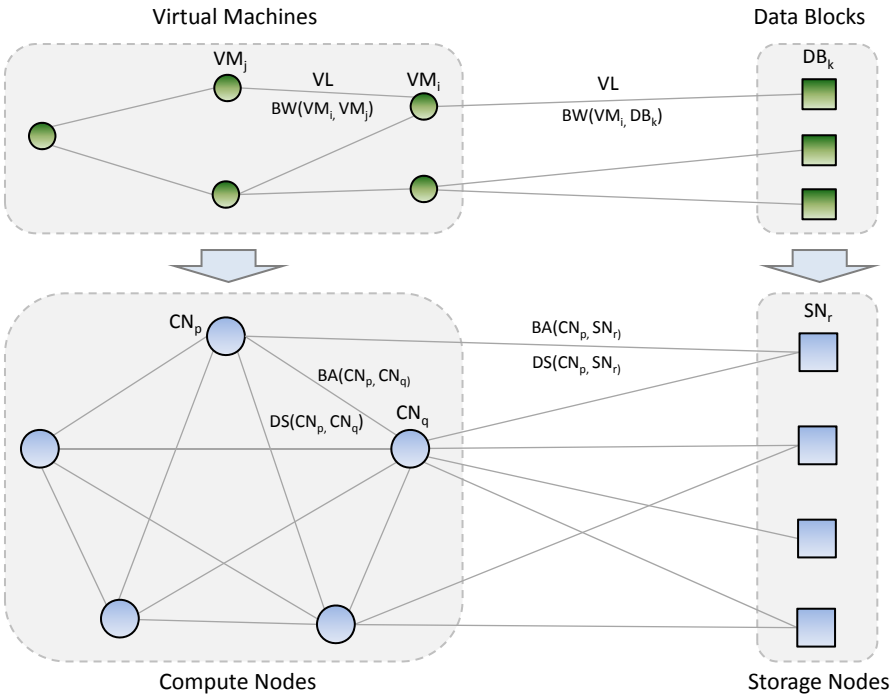

Figure 5.3: Application environment placement in data center (best viewed in color).

placement approach (NDAP) and its associated algorithms are explicated in Section 5.3. Section 5.4 details the experiments performed and shows the results, together with their analysis. Finally, Section 5.5 concludes the chapter with a summary of the contributions and results.

5.2 Network-aware VM Cluster Placement Problem

While deploying composite applications in Cloud data centers, such as multi-tier or workflow applications, customers request multiple computing VMs in the form of a VM cluster or a Virtual Private Cloud (VPC) and multiple Data Blocks (DBs). These computing VMs have specific traffic flow requirements among themselves, as well as with the data blocks. Such traffic flow measures can be supplied as user-provided hints or expected bandwidth requirements, depending on the application type and its characteristics. The remainder of this section formally defines this composite application environment placement as an optimization problem. Figure 5.3 presents a visual representation of the application placement in a data center and Table 5.1 provides the various notations used in the problem definition and proposed solution.

Table 5.1: Notations and their meanings

Notation	*Meaning*
VM	Virtual Machine
DB	Data Block
AN	AE node (either a VM or a DB)
VMS	Set of VMs in an AE
DBS	Set of DBs in an AE
ANS	Set of ANs ($ANS = \{VMS \cup DBS\}$) in an AE
N_v	Total number of VMs in an AE
N_d	Total number of DBs in an AE
VL	Virtual Link
VCL	Virtual Computing Link that connects two VMs
VDL	Virtual Data Link that connects a VM and a DB
$vclList$	Ordered list of VCLs in an AE
$vdlList$	Ordered list of VDLs in an AE
N_{vc}	Total number of VCLs in an AE
N_{vd}	Total number of VDLs in an AE
N_{vn}	Average number of NTPP VLs of a VM or a DB
$BW(VM_i, VM_j)$	Bandwidth demand between VM_i and VM_j
$BW(VM_i, DB_k)$	Bandwidth demand between VM_i and DB_k
CN	Computing Node
SN	Storage Node
$DN(AN)$	DC node where AN is placed
CNS	Set of CNs in a DC
SNS	Set of SNs in a DC
N_c	Total number of CNs in a DC
N_s	Total number of SNs in a DC
$cnList$	Ordered list of CNs in a DC
$snList$	Ordered list of SNs in a DC
PL	Physical Link
PCL	Physical Computing Link that connects two CNs
PDL	Physical Data Link that connects a CN and a SN
$DS(CN_p, CN_q)$	Network distance between CN_p and CN_q
$DS(CN_p, SN_r)$	Network distance between CN_p and SN_r
$BA(CN_p, CN_q)$	Available bandwidth between CN_p and CN_q
$BA(CN_p, SN_r)$	Available bandwidth between CN_p and SN_r
f_2	NAPP Objective Function

5.2.1 Formal Definition

An *Application Environment* is defined as $AE = \{VMS, DBS\}$, where VMS is the set of requested VMs: $VMS = \{VM_i : 1 \leq i \leq N_v\}$ and DBS is the set of requested DBs: $DBS = \{DB_k : 1 \leq k \leq N_d\}$. Each VM VM_i has specification of its CPU and memory demands represented by VM_i^{CPU} and VM_i^{MEM}, respectively, and each DB DB_k has the specification of its storage resource demand denoted by DB_k^{STR}.

Data communication requirements between any two VMs and between a VM and a DB are specified as *Virtual Links* (VLs) between $\langle VM, VM \rangle$ pairs and $\langle VM, DB \rangle$ pairs, respectively, during AE specification and deployment. The bandwidth demand or traffic load between VM_i and VM_j is represented by $BW(VM_i, VM_j)$. Similarly, the bandwidth demand between VM_i and DB_k is represented by $BW(VM_i, DB_k)$. These bandwidth requirements are provided as user input along with the VM and DB specifications.

A *Data Center* is defined as $DC = \{CNS, SNS\}$, where CNS is the set of computing nodes (e.g., physical servers or computing components of a multi-function storage device) in DC: $CNS = \{CN_p : 1 \leq p \leq N_c\}$ and SNS is the set of storage nodes: $SNS = \{SN_r : 1 \leq r \leq N_s\}$. For each computing node CN_p, the available CPU and memory resource capacities are represented by CN_p^{CPU} and CN_p^{MEM}, respectively. Here available resource indicates the remaining usable resource of a CN that may have already hosted other VMs that are consuming the rest of the resources. Similarly, for each storage node SN_r, the available storage resource capacity is represented by SN_r^{STR}.

Computing nodes and storage nodes are interconnected by *Physical Links* (PLs) in the data center communication network. PL distance and available bandwidth between two computing nodes CN_p and CN_q are denoted by $DS(CN_p, CN_q)$ and $BA(CN_p, CN_q)$, respectively. Similarly, PL distance and available bandwidth between a computing node CN_p and a storage node SN_r are represented by $DS(CN_p, SN_r)$ and $BA(CN_p, SN_r)$, respectively. PL distance can be any practical measure, such as link latency, number of hops or switches, and so on. Furthermore, this data center model is not restricted to any fixed network topology. Therefore, the network distance DS and available bandwidth BA models are generic and different model formulations focusing on any particular network topology or architecture can be readily applied in the optimization framework and proposed solution. In the experiments, the number of hops or switches between any two data center nodes is used as the only input parameter for the DS function in order to measure the PL distance. Although, singular distances between $\langle CN, CN \rangle$ and $\langle CN, SN \rangle$ pairs are considered in the experiments, link redundancy and multiple communication paths in data centers can be incorporated in the proposed model and placement algorithm by the appropriate definition of distance function (DS) and available bandwidth function (BA), respectively.

Furthermore, $DN(VM_i)$ denotes the computing node where VM_i is currently placed, otherwise if VM_i is not already placed, $DN(VM_i) = null$. Similarly, $DN(DB_k)$ denotes the storage node where DB_k is currently placed.

The network cost of placing VM_i in CN_p and VM_j in CN_q is defined as follows:

$$Cost(VM_i, CN_p, VM_j, CN_q) = BW(VM_i, VM_j) \times DS(CN_p, CN_q). \tag{5.1}$$

Likewise, the network cost of placing VM_i in CN_p and DB_k in SN_r is defined as:

$$Cost(VM_i, CN_p, DB_k, SN_r) = BW(VM_i, DB_k) \times DS(CN_p, SN_r). \tag{5.2}$$

Given the AE to deploy in the DC, the objective of the NAPP problem is to find placements for VMs and DBs in CNs and SNs, respectively, in such a way that the overall network cost or communication overhead due to the AE deployment is minimized. Hence, the *Objective Function* (OF) f_2 is formulated as follows:

$$\begin{aligned}\underset{\substack{\forall i: DN(VM_i) \\ \forall k: DN(VM_k)}}{\textbf{minimize}}\ f_2(AE, DC) = \sum_{i=1}^{N_v} \Bigg(\sum_{j=1}^{N_v} Cost(VM_i, DN(VM_i), VM_j, DN(VM_j)) + \\ \sum_{k=1}^{N_d} Cost(VM_i, DN(VM_i), DB_k, DN(DB_k)) \Bigg).\end{aligned} \tag{5.3}$$

The above AE placement is subject to the constraints that the available resource capacities of any CN and SN are not violated:

$$\forall p: \sum_{\forall i: DN(VM_i) = CN_p} VM_i^{CPU} \leq CN_p^{CPU}. \tag{5.4}$$

$$\forall p: \sum_{\forall i: DN(VM_i) = CN_p} VM_i^{MEM} \leq CN_p^{MEM}. \tag{5.5}$$

$$\forall r: \sum_{\forall k: DN(DB_k) = SN_r} DB_k^{STR} \leq SN_r^{STR}. \tag{5.6}$$

Furthermore, the sum of the bandwidth demands of the VLs that are placed on each PL must be less than or equal to the available bandwidth of the PL:

$$\forall p \forall q: \; BA(CN_p, CN_q) \geq \sum_{\forall i: DN(VM_i)=CN_p} \; \sum_{\forall j: DN(VM_j)=CN_q} BW(VM_i, VM_j). \tag{5.7}$$

$$\forall p \forall r: \; BA(CN_p, SN_r) \geq \sum_{\forall i: DN(VM_i)=CN_p} \; \sum_{\forall k: DN(DB_k)=SN_r} BW(VM_i, DB_k). \tag{5.8}$$

Given that every VM and DB placement fulfills the above-mentioned constraints (5.4 - 5.8), the NAPP problem defined by the OF f_2 (5.3) is explained as follows: among all possible feasible placements of VMs and DBs in AE, the placement that has the minimum cost is the optimal solution. Therefore, NAPP falls in the category of combinatorial optimization problems. In particular, it is an extended form of the *Quadratic Assignment Problem* (QAP) [80], which is proven to be computationally $\mathcal{NP}-hard$ [18].

5.3 Proposed Solution

The proposed network-aware VM and DB placement approach (NDAP) tries to place the VLs in such a way that network packets need to travel short distances. For better explanation of the solution approach, the above models of AE and DC are extended by addition of some other notations.

Every AE node is represented by *AN*, which can either be a VM or a DB, and the set of all ANs in an AE is represented by *ANS*. Every *VL* can be either a *Virtual Computing Link (VCL)*, i.e., *VL* between two VMs or a *Virtual Data Link (VDL)*, i.e., VL between a VM and a DB. The total number of VCLs and VDLs in an AE is represented by N_{vc} and N_{vd}, respectively. All the VCLs and VDLs are maintained in two ordered lists, *vclList* and *vdlList*, respectively. While VM-VM communication (VCL) and VM-DB communication (VDL) may be considered closely related, they differ in terms of actor and size. As only a VM can initiate communications, VCL supports an "active" duplex link, while VDL supports a "passive" duplex link. More precisely, the bandwidth demands of VDLs are multiple orders larger than those of VCLs.

Every DC node is represented by *DN*, which can either be a *CN* or a *SN*. All the CNs and SNs in a DC are maintained in two ordered lists, *cnList* and *snList*, respectively.

Every *PL* can be either a *Physical Computing Link (PCL)*, i.e., *PL* between two CNs or a *Physical Data Link (PDL)* i.e., *PL* between a CN and a SN.

The proposed NDAP algorithm is a greedy heuristic that first sorts the *vdlList* and *vclList* in decreasing order of the bandwidth demand of VDLs and VCLs. It then tries to places all the VDLs from the *vdlList*, along with any associated VCLs to fulfill placement dependency, on the feasible PDLs and PCLs, and their associated VMs and DBs in CNs and SNs, respectively, focusing on the goal of minimizing the network cost incurred due to placement of all the VDLs and associated VCLs. Finally, NDAP tries to place the remaining VCLs from the *vclList* on PCLs, along with their associated VMs and DBs in CNs and SNs, respectively, again with the aim of reducing the network cost incurred.

As mentioned in Section 5.2, NAPP is in fact an $\mathcal{NP}$*–hard*, combinatorial optimization problem similar to QAP and [103] have shown that even finding an approximate solution for QAP within some constant factor from the optimal solution cannot be done in polynomial time unless P=NP. Since greedy heuristics are relatively fast, easy to understand and implement, and very often used as an effective solution approach for NP-complete problems, a greedy heuristic (NDAP) is proposed as a solution for the NAPP problem.

The straightforward placement of an individual VL (either VDL or VCL) on a preferred PL is not always possible, since one or both of its ANs can have *Peer ANs* connected by *Peer VLs* (Figure 5.4(a)). At any point during an AE placement process, a VL can have Peer ANs that are already placed. The peer VLs that have already-placed peer ANs are termed *need-to-place peer VLs* (NTPP VLs), indicating the condition that placement of any VL also requires the simultaneous placement of its NTPP VLs and the average number of NTPP VLs for any VM or DB is denoted by N_{vn}. The maximum value of N_{vn} can be $N_v + N_d - 1$, which indicates that the corresponding VM or DB has VLs with all the other VMs and DBs in the AE. Since, for any VL placement, the corresponding placement of its NTPP VLs is an integral part of the NDAP placement strategy, first the VL placement feasibility part of the NDAP algorithm is described in the following subsection and the remaining four subsections describe other constituent components of the NDAP algorithm. Finally, a detailed description of the final NDAP algorithm is provided, along with the pseudocode.

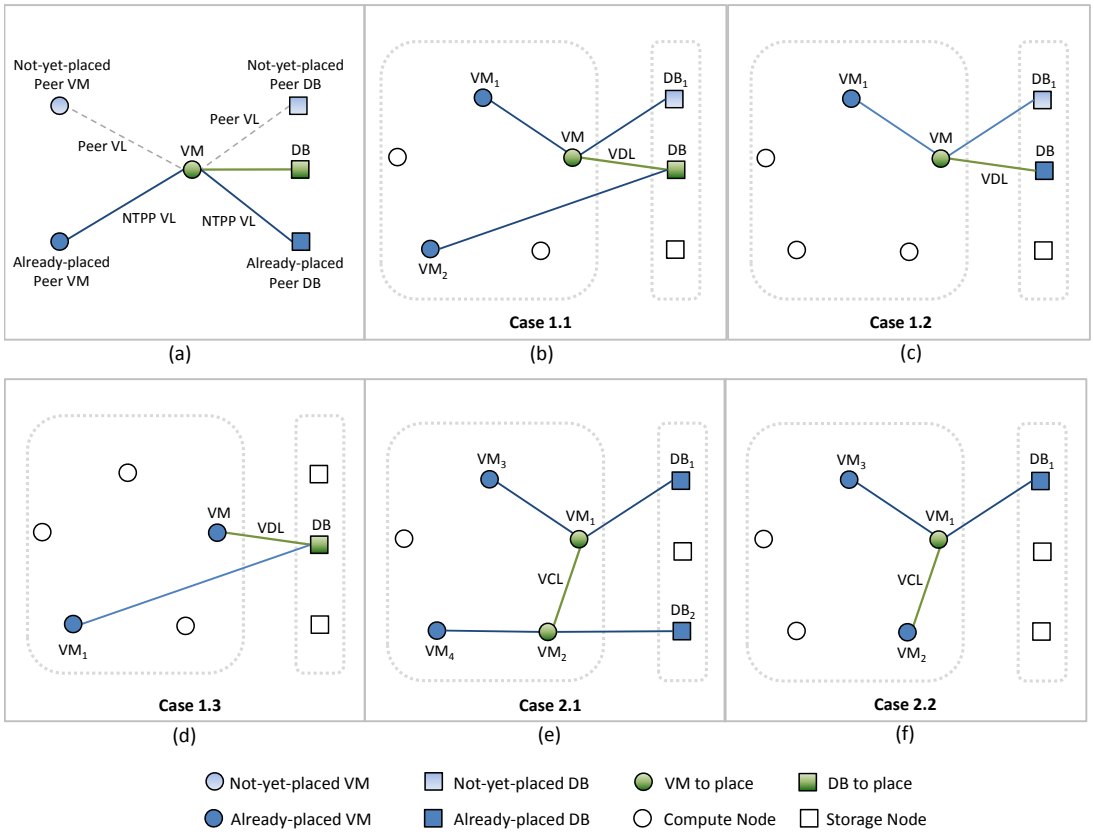

Figure 5.4: (a) Peer VL and NTPP VL, and (b-f) Five possible VL placement scenarios (best viewed in color).

5.3.1 VL Placement Feasibility

During the course of AE placement, when NDAP tries to place a *VL* that has one or both of its ANs not placed yet (i.e., $DN(AN) = null$), then a feasible placement for the *VL* needs to ensure that (1) the *VL* itself is placed on a feasible PL, (2) its ANs are placed on feasible DNs, and (3) all the NTPP VLs are placed on feasible PLs.

Depending on the type of *VL* and the current placement status of its ANs, five different cases may arise that are presented below. The NDAP placement algorithm handles these five cases separately. Figure 5.4(b)-(f) provide a visual representation of the five cases, where the *VL* to place is shown as a solid green line and its NTPP VLs are shown as solid blue lines.

VDL Placement: When trying to place a *VDL*, any of the following three cases may arise:

Case 1.1: Both the *VM* and *DB* are not placed yet and their peers VM_1, DB_1, and VM_2 are already placed (Figure 5.4(b)).

Case 1.2: DB is placed but *VM* is not placed yet and *VM*'s peers VM_1 and DB_1 are already placed (Figure 5.4(c)).

Case 1.3: VM is placed but *DB* is not placed yet and *DB*'s peer VM_1 is already placed (Figure 5.4(d)).

VCL Placement: In the case of *VCL* placement, any of the following two cases may arise:

Case 2.1: Both the VMs (VM_1 and VM_2) are not placed yet and their peers VM_3, DB_1, VM_4, and DB_2 are already placed (Figure 5.4(e)).

Case 2.2: Only one of the VMs is already placed and its peers VM_3 and DB_1 are already placed (Figure 5.4(f)).

In all the above cases, the placement feasibility of the NTPP VDLs and VCLs of the not-yet-placed VMs and DBs must be checked against the corresponding PDLs and PCLs, respectively (5.7 & 5.8).

5.3.2 Feasibility and Network Cost of VM and Peer VLs Placement

When NDAP tries to place a VM in a CN, it is feasible when (1) the computing and memory resource demands of the VM can be fulfilled by the remaining computing and memory resource capacities of the CN, and (2) the bandwidth demands of all the NTPP VLs can be satisfied by the available bandwidth capacities of the corresponding underlying PLs (Figure 5.5(a)–(b)):

$$VMPeerFeas(VM, CN) = \begin{cases} 1, & \text{if Eq. 5.4 \& 5.5 holds and, } DN(AN) \neq \text{null and} \\ & BW(VM, AN) \leq BA(CN, DN(AN)) \text{ for } \forall AN; \\ 0, & \text{otherwise.} \end{cases} \tag{5.9}$$

When NDAP tries to place two VMs (VM_1 and VM_2) in a single CN, it is feasible when (1) the combined computing and memory resource demands of the two VMs can be fulfilled by the remaining computing and memory resource capacities of the CN, and (2) the bandwidth demands of all the NTPP VLs of both the VMs can be satisfied by the

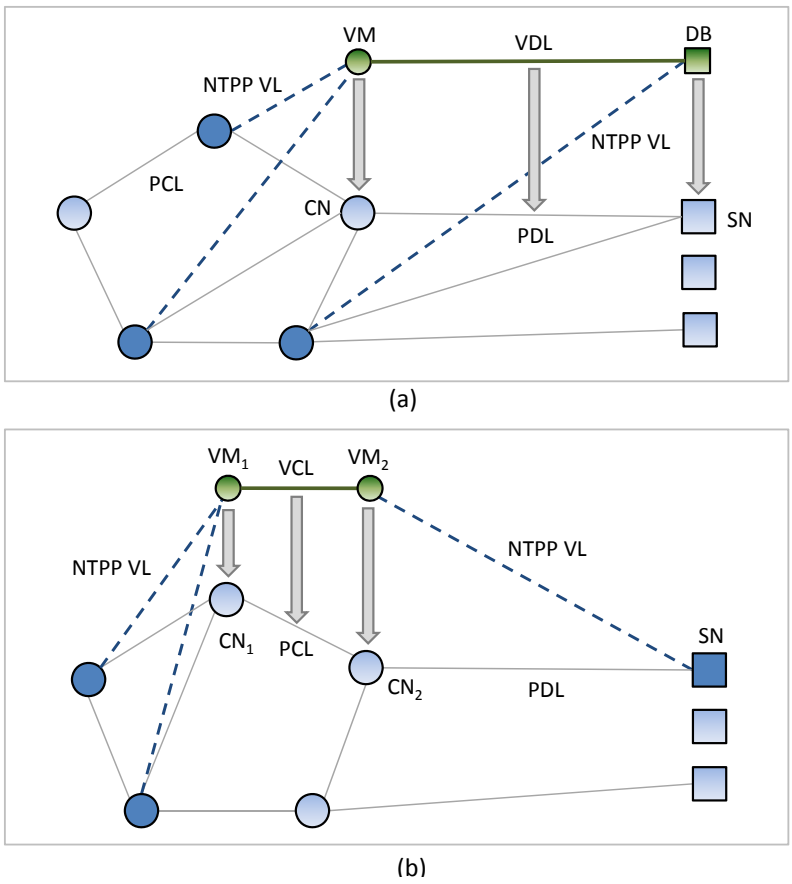

Figure 5.5: Placement of (a) VDL and (b) VCL along with NTPP VLs (best viewed in color).

available bandwidth capacities of the corresponding underlying PLs:

$$VMPeerFeas(VM_1, VM_2, CN) = \begin{cases} 1, & \text{if Eq. 5.4 \& 5.5 holds for } (VM_1 + VM_2) \text{ and,} \\ & \forall AN: DN(AN) \neq \text{null and,} \\ & BW(VM_1, AN) + BW(VM_2, AN) \leq BA(CN, DN(AN)); \\ 0, & \text{otherwise.} \end{cases} \tag{5.10}$$

The network cost of a VM placement is measured as the accumulated cost of placing all of its NTPP VLs:

$$VMPeerCost(VM, CN) = \sum_{\forall AN:\, DN(AN) \neq null \wedge BW(VM,AN) > 0} Cost(VM, CN, AN, DN(AN)). \tag{5.11}$$

5.3.3 Feasibility and Network Cost of DB and Peer VLs Placement

When trying to place a DB in a SN, it is feasible when (1) the storage resource demand of the DB can be fulfilled by the remaining storage resource capacity of the SN, and (2) the bandwidth demands of the NTPP VLs can be satisfied by the available bandwidth

capacities of the corresponding underlying PLs (Figure 5.5(a)):

$$DBPeerFeas(DB, SN) = \begin{cases} 1, & \text{if Eq. 5.6 holds and, } DN(AN) \neq \text{null and} \\ & BW(AN, DB) \leq BA(DN(AN), SN) \text{ for } \forall AN; \\ 0, & \text{otherwise.} \end{cases} \tag{5.12}$$

The network cost of any DB placement is measured as the total cost of placing all of its NTPP VLs:

$$DBPeerCost(DB, SN) = \sum_{\forall AN:\, DN(AN) \neq null \wedge BW(AN,DB) > 0} Cost(AN, DN(AN), DB, SN). \tag{5.13}$$

5.3.4 VM and Peer VLs Placement

Algorithm 5.1 shows the subroutine for placing a *VM* and its associated NTPP VLs. First, the *VM*-to-*CN* placement is accomplished by reducing the available CPU and memory resource capacities of the *CN* by the amount of CPU and memory resource requirements of the *VM* and setting the *CN* as the DC node of the *VM* [lines 1–3]. Then, for each already-placed peer *AN* of *VM* (i.e., any *AN* that has non-zero traffic load with *VM* and $DN(AN) \neq null$), it is checked if the selected *CN* is different from the computing node where the peer *AN* is placed, in which case the available bandwidth capacity of the PL that connects the selected *CN* and $DN(AN)$ is reduced by the amount of the bandwidth demand of the corresponding NTPP VL [lines 4–8]. In those cases where the selected *CN* is the computing node where the peer *AN* is placed, the *VM* can communicate with the peer *AN* through memory copy instead of passing packets through physical network links. Afterwards, the NTPP *VL* is removed from the $vclList$ or $vdlList$, depending on whether it is a VCL or VDL, respectively, in order to indicate that it is now placed [lines 9–13].

5.3.5 DB and Peer VLs Placement

Algorithm 5.2 shows the subroutine for placing a *DB* in a *SN* and its associated NTPP VLs. First, the *DB*-to-*SN* placement is performed by reducing the available storage capacity of the *SN* by the amount of the storage requirements of the *DB* and by setting the *SN* as the DC node of *DB* [lines 1–2]. Then, for every already-placed peer *AN* of *DB* (i.e., any *AN* that has non-zero traffic load with *DB* and $DN(AN) \neq null$), the available bandwidth capacity of the PDL that connects the selected *SN* and $DN(AN)$ is reduced by the amount of the NTPP *VL*'s bandwidth requirement [lines 3–5], and lastly, the NTPP *VL* is removed from the $vdlList$ to mark that it is now placed [lines 6–7].

Algorithm 5.1 PlaceVMandPeerVLs

Input: VM to place, CN where VM is being placed, set of all ANs ANS, $vclList$, and $vdlList$.

```
1: CN^CPU ← CN^CPU − VM^CPU
2: CN^MEM ← CN^MEM − VM^MEM
3: DN(VM) ← CN
4: for each AN ∈ ANS do
5:    if BW(VM, AN) > 0 ∧ DN(AN) ≠ null then
6:       if DN(AN) ≠ CN then
7:          BA(CN, DN(AN)) ← BA(CN, DN(AN)) − BW(VM, AN)
8:       end if
9:       VL ← virtualLink(VM, AN)
10:      if VL is a VCL then
11:         vclList.remove(VL)
12:      else
13:         vdlList.remove(VL)
14:      end if
15:   end if
16: end for
```

Algorithm 5.2 PlaceDBandPeerVLs

Input: DB to place, SN where DB is being placed, set of all ANs ANS, and $vdlList$.

```
1: SN^STR ← SN^STR − DB^STR
2: DN(DB) ← SN
3: for each AN ∈ ANS do
4:    if BW(AN, DB) > 0 ∧ DN(AN) ≠ null then
5:       BA(DN(AN), SN) ← BA(DN(AN), SN) − BW(AN, DB)
6:       VL ← virtualLink(AN, DB)
7:       vdlList.remove(VL)
8:    end if
9: end for
```

5.3.6 NDAP Algorithm

The pseudocode of the final NDAP algorithm is presented in Algorithm 5.3. It receives the DC and AE as input and returns the network cost incurred due to the AE placement. NDAP begins by performing necessary initialization and sorting the $vdlList$ and $vclList$ in decreasing order of their VLs' bandwidth demands [lines 1–2]. Afterwards, it iteratively takes the first VDL from the $vdlList$ (i.e., the VDL with the highest bandwidth demand) and tries to place it (along with its VM and DB, and all NTPP VLs) in a PDL among the feasible PDLs so that the total network cost incurred due to the placement is minimum [lines 3–34] (Figure 5.5(a)). As explained in Section 5.3.1, there can be three cases for this placement, depending on the current placement status of the VDL's VM and DB.

When the VDL matches Case 1.1 (both VM and DB are not placed), for each feasible CN and SN in DC (5.9 & 5.12), it is checked if the bandwidth demand of the VDL can be satisfied by the available bandwidth of the corresponding PDL connecting the CN and

Algorithm 5.3 NDAP Algorithm

Input: DC and AE.
Output: Total network cost of AE placement.

1: $totCost \leftarrow 0$
2: Sort $vdlList$ and $vclList$ in decreasing order of VL's bandwidth demands
3: **while** $vdlList \neq \emptyset$ **do** {NDAP tries to place all VDLs in vdlList}
4: $VDL \leftarrow vdlList[0]; minCost \leftarrow \infty; VM \leftarrow VDL.VM; DB \leftarrow VDL.DB; selCN \leftarrow null; selSN \leftarrow null$
5:
6: **if** $DN(VM) = null \wedge DN(DB) = null$ **then** {Case 1.1: Both VM and DB are not placed}
7: **for** each $CN \in cnList \wedge VMPeerFeas(VM, CN) = 1$ **do**
8: **for** each $SN \in snList \wedge DBPeerFeas(DB, SN) = 1$ **do**
9: **if** $BW(VM, DB) \leq BA(CN, SN)$ **then**
10: $cost \leftarrow BW(VM, DB) \times DS(CN, SN) + VMPeerCost(VM, CN) + DBPeerCost(DB, SN)$
11: **if** $cost < minCost$ **then** $minCost \leftarrow cost; selCN \leftarrow CN; selSN \leftarrow SN$ **endif**
12: **end if**
13: **end for**
14: **end for**
15: **if** $minCost \neq \infty$ **then** $BA(selCN, selSN) \leftarrow BA(selCN, selSN) - BW(VM, DB)$ **endif**
16:
17: **else if** $DN(VM) = null \wedge DN(DB) \neq null$ **then** {Case 1.2: VM is not placed and DB is already placed}
18: **for** each $CN \in cnList \wedge VMPeerFeas(VM, CN) = 1$ **do**
19: $cost \leftarrow VMPeerCost(VM, CN)$
20: **if** $cost < minCost$ **then** $minCost \leftarrow cost; selCN \leftarrow CN$ **endif**
21: **end for**
22:
23: **else if** $DN(VM) \neq null \wedge DN(DB) = null$ **then** {Case 1.3: VM is already placed and DB is not placed}
24: **for** each $SN \in snList \wedge DBPeerFeas(DB, SN) = 1$ **do**
25: $cost \leftarrow DBPeerCost(DB, SN)$
26: **if** $cost < minCost$ **then** $minCost \leftarrow cost; selSN \leftarrow SN$ **endif**
27: **end for**
28: **end if**
29:
30: **if** $minCost = \infty$ **then return** -1 **endif** {Feasible placement not found}
31: **if** $selCN \neq null$ **then** $PlaceVMandPeerVLs(VM, selCN)$ **endif** {For Case 1.1 and Case 1.2}
32: **if** $selSN \neq null$ **then** $PlaceDBandPeerVLs(DB, selSN)$ **endif** {For Case 1.1 and Case 1.3}
33: $totCost \leftarrow totCost + minCost; vdlList.remove(0)$
34: **end while**
35:
36: **while** $vclList \neq \emptyset$ **do** {NDAP tries to place remaining VCLs in vclList}
37: $VCL \leftarrow vclList[0]; minCost \leftarrow \infty$
38: $VM_1 \leftarrow VCL.VM_1; VM_2 \leftarrow VCL.VM_2; selCN_1 \leftarrow null; selCN_2 \leftarrow null$
39:
40: **if** $DN(VM_1) = null \wedge DN(VM_2) = null$ **then** {Case 2.1: Both VMs are not placed}
41: **for** each $CN_1 \in cnList \wedge VMPeerFeas(VM_1, CN_1) = 1$ **do**
42: **for** each $CN_2 \in cnList \wedge VMPeerFeas(VM_2, CN_2) = 1$ **do**
43: **if** $CN_1 = CN_2 \wedge VMPeerFeas(VM_1, VM_2, CN) = 0$ **then continue endif**
44: **if** $BW(VM_1, VM_2) \leq BA(CN_1, CN_2)$ **then**
45: $cost \leftarrow BW(VM_1, VM_2) \times DS(CN_1, CN_2)$
46: $cost \leftarrow cost + VMPeerCost(VM_1, CN_1) + VMPeerCost(VM_2, CN_2)$
47: **if** $cost < minCost$ **then** $minCost \leftarrow cost; selCN_1 \leftarrow CN_1; selCN_2 \leftarrow CN_2$ **endif**
48: **end if**
49: **end for**
50: **end for**
51: **if** $minCost \neq \infty$ **then** $BA(selCN_1, selCN_2) \leftarrow BA(selCN_1, selCN_2) - BW(VM_1, VM_2)$ **endif**
52:
53: **else if** $DN(VM_1) \neq null \vee DN(VM_2) \neq null$ **then** {Case 2.2: One of the VMs is not placed}
54: **if** $DN(VM_1) \neq null$ **then** swap values of VM_1 and VM_2 **endif** {Now VM_1 denotes the not-yet-placed VM}
55: **for** each $CN_1 \in cnList \wedge VMPeerFeas(VM_1, CN_1) = 1$ **do**
56: $cost \leftarrow VMPeerCost(VM_1, CN_1)$
57: **if** $cost < minCost$ **then** $minCost \leftarrow cost; selCN_1 \leftarrow CN_1$ **endif**
58: **end for**
59: **end if**
60:
61: **if** $minCost = \infty$ **then return** -1 **endif** {Feasible placement not found}
62: $PlaceVMandPeerVLs(VM_1, selCN_1)$ {For Case 2.1 and Case 2.2}
63: **if** $selCN_2 \neq null$ **then** $PlaceVMandPeerVLs(VM_2, selCN_2)$ **endif** {For Case 2.1}
64: $totCost \leftarrow totCost + minCost; vclList.remove(0)$
65: **end while**
66:
67: **return** $totCost$

SN. If it can be satisfied, the total cost of placing the VDL and its associated NTPP VLs is measured (5.11 & 5.13). The $\langle CN, SN \rangle$ pair that offers the minimum cost is selected for placing the $\langle VM, DB \rangle$ pair and the available bandwidth capacity of the PDL that connects the selected $\langle CN, SN \rangle$ pair is updated to reflect the VDL placement [lines 6–15]. When the VDL matches Case 1.2 (VM is not placed, but DB is placed), the feasible CN that offers minimum cost placement is selected for the VM and the total cost is measured [lines 17–21]. In a similar way, Case 1.3 (VM is placed, but DB is not placed) is handled in lines 23–28 and the best SN is selected for the DB placement.

If NDAP fails to find a feasible CN or SN, it returns -1 to indicate failure in finding a feasible placement for the AE [line 30]. Otherwise, it activates the placements of the VM and DB along with their NTPP VLs by using subroutines $PlaceVMandPeerVLs$ (Algorithm 5.1) and $PlaceDBandPeerVLs$ (Algorithm 5.2), accumulates the measured cost in variable $totCost$, and removes the VDL from $vdlList$ [lines 31–33]. In this way, by picking the VDLs from a list that is already sorted based on bandwidth demand, and trying to place each VDL, along with its NTPP VLs, in such a way that the incurred network cost is minimum in the current context of the DC resource state, NDAP strives to minimize the total network cost of placing the AE, as formulated by the OF f_2 (5.3) of the proposed optimization. In particular, in each iteration of the first while loop (lines 3–34), NDAP picks the next highest bandwidth demanding VDL from the $vdlList$ and finds the best placement (i.e., minimum cost) for it along with its NTPP VLs. Moreover, the placement of the VDLs is performed before the placement of the VCLs, since the average VDL bandwidth demand is expected to be higher than the average VCL bandwidth demand considering the fact that the average traffic volume for the $\langle VM, DB \rangle$ pairs is expected to be higher than that for the $\langle VM, VM \rangle$ pairs.

After NDAP has successfully placed all the VDLs, then it starts placing the remaining VCLs in the $vclList$ (i.e., VCLs that were not NTPP VLs during the VDLs placement). For this part of the placement, NDAP applies a similar approach by repeatedly taking the first VCL from the $vclList$ and trying to place it on a feasible PCL so that the network cost incurred is minimum [lines 36–65] (Figure 5.5(b)). This time, there can be two cases, depending on the placement status of the two VMs of the VCL (Section 5.3.1).

When the VCL matches Case 2.1 (both VMs are not placed), for each feasible CN in DC (5.9), it is first checked if both the VMs (VM_1 and VM_2) are being tried for placement

in the same CN. In such cases, if the combined placement of both the VMs along with their NTPP VLs is not feasible (5.10), NDAP continues checking feasibility for different CNs [line 43]. When both VMs placement feasibility passes and the bandwidth demand of the VCL can be satisfied by the available bandwidth of the corresponding PCL connecting the CNs, the total cost of placing the VCL and its associated NTPP VLs is measured (5.11 & 5.13) [lines 44–48]. When both the VMs are being tried for the same CN, they can communicate with each other using memory copy rather going through physical network links and the available bandwidth check in line 44 works correctly, since the intra-CN available bandwidth is considered to be unlimited. The $\langle CN_1, CN_2 \rangle$ pair that offers the minimum cost is selected for placing the $\langle VM_1, VM_2 \rangle$ pair and the available bandwidth capacity of the PCL connecting the selected $\langle CN_1, CN_2 \rangle$ pair is updated to reflect the VCL placement [lines 47–51]. When the VCL matches Case 2.2 (one of the VMs is not placed), the feasible CN that offers the minimum cost placement is selected for the not-yet-placed VM (VM_1) and the total cost is measured [lines 53–59].

Similar to VDL placement, if NDAP fails to find feasible CNs for any VCL placement, it returns –1 to indicate failure [line 61]. Otherwise, it activates the placements of the VMs along with their NTPP VLs by using subroutine $PlaceVMandPeerVLs$ (Algorithm 5.1), accumulates the measured cost in $totCost$, and removes the VCL from the $vclList$ [lines 61–65]. For the same reason as for VDL placement, the VCL placement part of the NDAP algorithm fosters the reduction of the OF (f_2) value (5.3).

Finally, NDAP returns the total cost of the AE placement, which also indicates a successful placement [line 67].

5.4 Performance Evaluation

This section describes the performance of the proposed NDAP algorithm compared to other algorithms, based on a set of simulation-based experiments. Section 5.4.1 gives a brief description of the evaluated algorithms, Section 5.4.2 describes the various aspects of the simulation environment, and the results are discussed in the subsequent subsections.

5.4.1 Algorithms Compared

The following algorithms are evaluated and compared in this research:

Network-aware VM Allocation (NVA)

This is an extended version of the network-aware VM placement approach proposed by [100], where the authors consider already-placed data blocks. In this version, each $DB \in DBS$ is placed randomly in a $SN \in SNS$. Each VM that has one or more VDL is then placed according to the VM allocation algorithm presented by the authors, provided that all of its NTPP VLs are placed on feasible PLs. For any remaining $VM \in VMS$, it is placed randomly. All the above placements are subject to the constraints presented in (5.4 - 5.8). In order to increase the probability of feasible placements, DB and VM placements are tried multiple times and the maximum number of tries (N_{mt}) is parameterized by a constant which is set to 100 in the simulation.

Time Complexity: For the above-mentioned implementation, the worst-case time complexity of NVA algorithm is given by:

$$T_{NVA} = \mathcal{O}(N_d N_{mt}) + \mathcal{O}(N_v N_c N_{vn}) + \mathcal{O}(N_v N_{mt}). \tag{5.14}$$

Since N_{mt} is a constant and the maximum number of VMs (N_v) and DBs (N_d) in an AE is generally much less than the number of computing nodes (N_c) in DC, the above time complexity reduces to:

$$T_{NVA} = \mathcal{O}(N_v N_c N_{vn}). \tag{5.15}$$

Memory Overhead: Given that NVA starts with already-placed DBs, and VM placements are done in-place using no auxiliary data structure, the NVA algorithm itself does not have any memory overhead.

First Fit Decreasing (FFD)

This algorithm begins by sorting the CNs in the *cnList* and the SNs in the *snList* in decreasing order based on their remaining resource capacities. Since CNs have two different types of resource capacity (CPU and memory), L1-norm mean estimator is used to convert the vector representation of multi-dimensional resources into scalar form. Similarly, all the VMs in the *vmList* and the DBs in the *dbList* are sorted in decreasing order of their resource demands. FFD then places each DB from the *dbList* in the first feasible SN of the *snList* according to the *First First* (FF) algorithm. Next, it places each VM from the *vmList* in the first feasible CN of the *cnList* along with any associated NTPP VLs. All the above placements are subject to the constraints presented in (5.4 - 5.8).

Time Complexity: For the above implementation of the FFD, the worst-case time complexity of FFD algorithm is given by:

$$T_{FFD} = \mathcal{O}(N_c \lg N_c) + \mathcal{O}(N_s \lg N_s) + \mathcal{O}(N_v \lg N_v) + \mathcal{O}(N_d \lg N_d) + \mathcal{O}(N_d N_s) + \mathcal{O}(N_v N_c). \quad (5.16)$$

Since in a typical setting the number of VMs (N_v) and DBs (N_d) in an AE is much less than the number of CNs (N_c) and SNs (N_s) in DC, the above term reduces to:

$$T_{FFD} = \mathcal{O}(N_c \lg N_c) + \mathcal{O}(N_s \lg N_s) + \mathcal{O}(N_d N_s) + \mathcal{O}(N_v N_c). \quad (5.17)$$

Memory Overhead: Since merge sort [28] is used in FFD to sort *cnList*, *snList*, *vmList*, and *dbList*, and N_c is usually greater than each of N_s, N_v, and N_d in a typical setting, it can be concluded that the memory overhead for the sorting operation is $\mathcal{O}(N_c)$. Apart from sorting, the placement decision part of FFD works in-place without using any additional data structure. Therefore, the memory overhead of FFD algorithm is given by:

$$M_{FFD} = \mathcal{O}(N_c). \quad (5.18)$$

Network- and Data-aware Application Placement (NDAP)

The NDAP algorithm is implemented primarily based on the the description presented in Section 5.3 and follows the execution flow presented in Algorithm 5.3. The final NDAP algorithm utilizes the feasibility check (5.9, 5.10, & 5.12), network cost computation (5.11 & 5.13), and the placement subroutines (Algorithm 5.1 & 5.2).

Time Complexity: All of these NDAP components need to go through a list of NTPP VLs for the corresponding VM or DB, and in the implementation, this list is stored in an array. Therefore, the time complexity for each of these NDAP components is $\mathcal{O}(N_{vn})$. So, for the above-mentioned implementation, the running time of the NDAP algorithm (referring to pseudocode in Algorithm 5.3) is the sum of the time needed for sorting *vdlList* and *vdlList* (T_2), the time needed for placing all the VDLs in *vdlList* ($T_{3\text{-}34}$), and the time needed for placing all the remaining VCLs in *vclList* ($T_{36\text{-}65}$). The time complexity for placing a single VDL (considering three cases) is given by:

$$\begin{aligned} T_{6\text{-}33} &= \mathcal{O}(N_c N_s N_{vn}) + \mathcal{O}(N_c N_{vn}) + \mathcal{O}(N_s N_{vn}) + \mathcal{O}(N_{vn}) \\ &= \mathcal{O}(N_c N_s N_{vn}). \end{aligned} \tag{5.19}$$

Therefore, the time complexity for placing all the VDLs is:

$$T_{3\text{-}34} = \mathcal{O}(N_{vd} N_c N_s N_{vn}). \tag{5.20}$$

Similarly, the time complexity for placing all the remaining VCLs is:

$$T_{36\text{-}65} = \mathcal{O}(N_{vc} N_c^2 N_{vn}). \tag{5.21}$$

Therefore, the worst-case time complexity of NDAP algorithm is given by:

$$\begin{aligned} T_{NDAP} &= T_2 + T_{3\text{-}34} + T_{36\text{-}65} \\ &= \mathcal{O}(N_{vd} \lg N_{vd}) + \mathcal{O}(N_{vc} \lg N_{vc}) + \mathcal{O}(N_{vd} N_c N_s N_{vn}) + \mathcal{O}(N_{vc} N_c^2 N_{vn}). \end{aligned} \tag{5.22}$$

Memory Overhead: For this implementation of the NDAP algorithm, merge sort is used in order to sort *vdlList* and *vclList* [line 2, Algorithms 5.3]. Given that AEs are typically constituted of a number of VMs and DBs with sparse communication links between them, it is assumed that $N_{vd} = N_{vc} = \mathcal{O}(N_v)$ since N_{vd} and N_{vc} are of the same order. Therefore, the memory overhead for this sorting operation is $\mathcal{O}(N_v)$. Apart from sorting, the placement decision part of NDAP [lines 3–67] works in-place and no additional data structure is needed. Therefore, the memory overhead of NDAP algorithm is given by:

$$M_{NDAP} = \mathcal{O}(N_v). \tag{5.23}$$

The detailed computational time complexity analyses presented above may be further simplified as follows. While the number of computing nodes outweighs the number of storage nodes in a typical DC, they may be assumed to be of the same order, i.e., $N_s = \mathcal{O}(N_c)$. Moreover, the size of a typical DC is at least a multiple order higher than that of an AE. Hence, it may also be assumed that $N_v, N_d, N_{vc}, N_{vd}, N_{vn} = o(N_c)$. From (5.15, 5.17, &

5.22), it can be concluded that the running times of the NVA, FFD, and NDAP algorithms are $\mathcal{O}(N_c)$, $\mathcal{O}(N_c \lg N_c)$, and $\mathcal{O}(N_c^2)$, respectively, i.e., these are linear, linearithmic, and quadratic time algorithms, respectively. Regarding the overhead of the above-mentioned algorithms, although there are variations in the run-time memory overhead, considering that the input optimization problem (i.e., AE placement in DC) itself has $\mathcal{O}(N_c)$ memory overhead, it can be concluded that, overall, all the compared algorithms have an equal memory overhead of $\mathcal{O}(N_c)$.

For all the above algorithms, if any feasible placement is not found for a VM or DB, the corresponding algorithm terminates with failure status.

5.4.2 Simulation Setup

Data Center Setup

In order to address the increasing complexity of large-scale Cloud data centers, network vendors are developing network architecture models focusing on the resource usage patterns of Cloud applications. For example, Juniper Networks Inc. in their "Cloud-ready data center reference architecture" suggest the use of Storage Area Networks (SANs) interconnected to the computing network with converged access switches [65], as shown in Figure 5.6. The simulated data center is generated following this reference architecture with a three-tier computing network topology (core-aggregation-access) [71] and a SAN-based storage network. Following the approach presented in [72]), the number of parameters is limited in simulating the data center by using the number of physical computing servers as the only parameter denoted by N. The quantity of other data center nodes are derived from N as follows: $5N/36$ high-end storage devices with built-in spare computing resources that work as multi-function devices for storage and computing, $4N/36 (= N/9)$ regular storage devices without additional computing resources, $N/36$ high-end core switches with built-in spare computing resources that work as multi-function devices for switching and computing, $N/18$ mid-level aggregation switches, and $5N/12$ $(= N/3 + N/12)$ access switches. Following the three-tier network topology [71], $N/3$ access switches provide connectivity between N computing servers and $N/18$ aggregation switches, whereas the $N/18$ aggregation switches connects $N/3$ access switches and $N/36$ core switches in the computing network. The remaining $N/12$ access switches provide connectivity between $N/4$ storage devices and $N/36$ core switches in the storage network. In such a data center

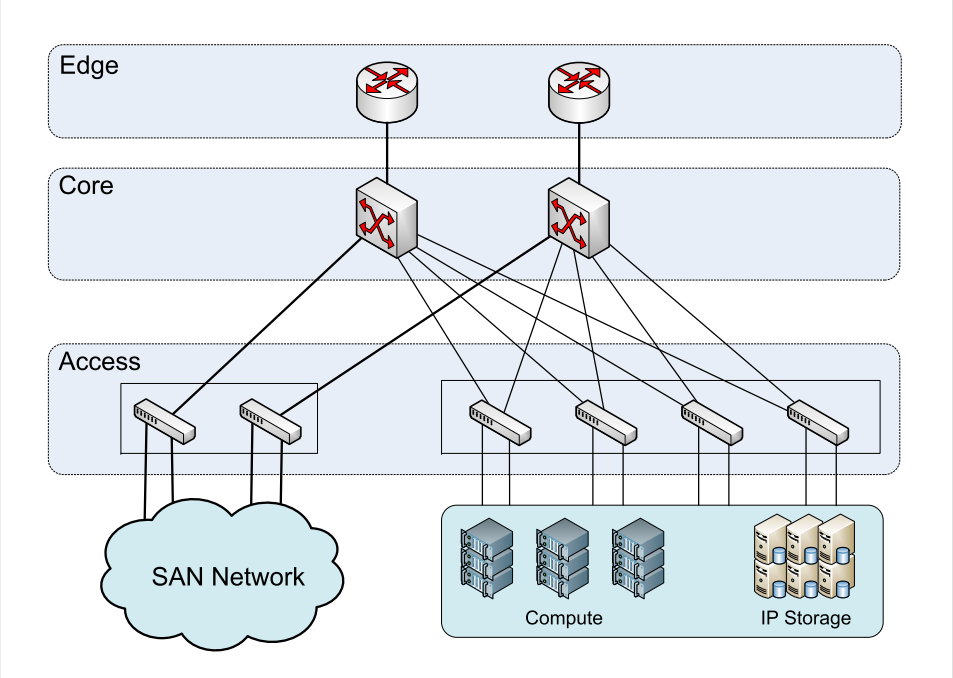

Figure 5.6: Cloud-ready data center network architecture (source: Juniper Networks Inc.).

setup, the total number of computing nodes (CNs) $N_c = N + 5N/36 + N/36 = 7N/6$ and the total number of storage nodes (SNs) $N_s = 5N/36 + 4N/36 = N/4$.

Network distances between $\langle CN, CN \rangle$ pairs and between $\langle CN, SN \rangle$ pairs are measured as $DS = h \times DF$, where h is the number of physical hops between two DC nodes (CN or SN) in the simulated data center architecture as defined above, and DF is the *Distance Factor* that implies the physical inter-hop distance. The value of h is computed using the analytical expression for tree topology as presented in [85], and DF is fed as a parameter into the simulation. the network distance of a node with itself is 0, which implies that data communication is done using memory copy without going through the network. A higher value of DF indicates greater relative communication distance between any two data center nodes.

Application Environment Setup

In order to model composite application environments for the simulation, multi-tier enterprise applications and scientific workflows are considered as representatives of the dominant Cloud applications. According to the analytical model for multi-tier Internet applications presented in [115], three-tier applications are modeled as comprised of 5 VMs ($N_v = 5$) and 3 DBs ($N_d = 3$) interconnected through 4 VCLs ($N_{vc} = 4$) and 5 VDLs

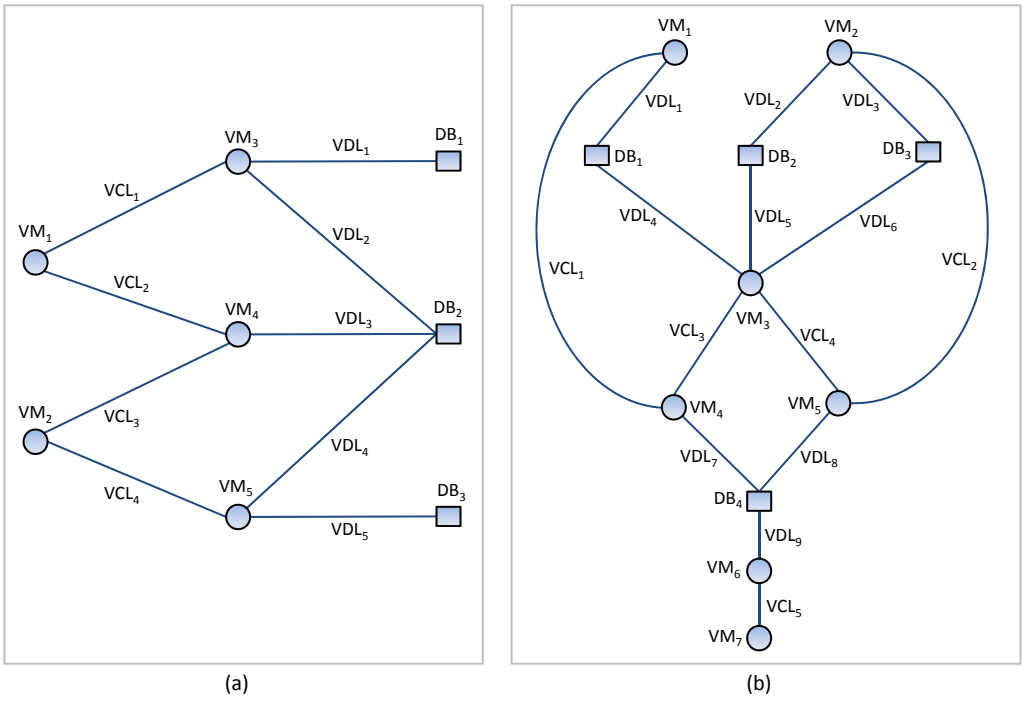

Figure 5.7: Application environment models for (a) Multi-tier application and (b) Scientific (Montage) workflow.

($N_{vd} = 5$) as shown in Figure 5.7(a). In order to model scientific applications, Montage workflow composed of 7 VMs ($N_v = 7$) and 4 DBs ($N_d = 4$) interconnected through 5 VCLs ($N_{vc} = 5$) and 9 VDLs ($N_{vd} = 9$) is simulated following the structure presented in [67] (Figure 5.7(b)). While deploying an application in the data center, user-provided hints on estimated resource demands are parameterized during the course of the experimentation. Extending the approaches presented in [85] and [106], computing resource demands (CPU and memory) for VMs, storage resource demands for DBs, and bandwidth demands for VLs are stochastically generated based on normal distribution with parameter means ($meanCom$, $meanStr$, and $meanVLBW$, respectively) and standard deviation (sd) against the normalized total resource capacities of CNs and SNs, and the bandwidth capacities of PLs, respectively.

Simulated Scenarios

For each of the experiments, all the algorithms started with their own empty data centers. In order to represent the dynamics of the real Cloud data centers, two types of events are simulated: (1) AE deployment and (2) AE termination. With the purpose of assessing the relative performance of the various placement algorithms in states of both higher and lower resource availability of data center nodes (CNs and SNs) and physical

links (PCLs and PDLs), this experiment simulated scenarios where the average number of AE deployments doubles the average number of AE terminations. Since during the initial phase of the experiments the data centers are empty, algorithms enjoy more freedom for the placement of AE components. Gradually, the data centers become loaded due to the higher number of AE deployments compared to the number of AE terminations. In order to reflect upon the reality of application deployment dynamics in real Clouds where the majority of the Cloud application spectrum is composed of multi-tier enterprise applications, in the simulated scenarios, 80% of the AE deployments were considered to be enterprise applications (three-tier application models) and 20% were considered as scientific applications (Montage workflow models). Overall, the following two scenarios were considered:

Group Scenario: For all the placement algorithms, AE deployments and terminations were continued until any of them failed to place an AE due to the lack of feasible placement. In order to maintain fairness among algorithms, the total number of AE deployments and terminations for each of the placement algorithms were equal and the same instances of AEs were deployed or terminated for each simulated event.

Individual Scenario: For each of the algorithms, AE deployment and termination were continued separately until it failed to place an AE due to the failure to find a feasible placement. Similar to the group scenario, all the algorithms drew AEs from same pools so that all the algorithms worked with the exactly same instances of AE for each event.

All the experiments presented in this paper are repeated 1000 times and the average results were reported.

Performance Evaluation Metrics

In order to assess the network load imposed due the placement decisions, the average network cost of AE deployment was computed (using OF f_2 according to (5.3)) for each of the algorithms in the group scenario. Since the cost functions (5.1 & 5.2) are defined based on the network distance between DC nodes and the expected amount of traffic flow, they effectively provide measures of the network packet transfer delays, and imposed packet forwarding load and power consumption for the network devices (e.g., switches and routers) and communication links. With the aim of maintaining a fair comparison among the algorithms, the average cost metric was computed and compared in the group scenario where all the algorithms terminated when any of them failed to place an AE due to the

feasible resource constraints (5.4 - 5.8) in DC. As a consequence, each algorithm worked with the same instances of AE at each deployment and termination event, and the average cost was computed over the same number of AEs.

In order to measure how effectively each of the algorithms utilized the network bandwidth during AE placements, the total number of AE deployments in empty DC was measured until the data center was saturated in the individual scenario. Using this performance metric, the effective capacity of the DC resources utilized by each of the placement algorithms was captured and compared. Moreover, this performance metric also captures the degree of convergence with solutions (i.e., successful placements of AEs) of the placement algorithms in situations when networking and computing resources within the data center components (e.g., servers and switches) are strained. This is due to the fact that the placement algorithm that deploys higher number of AEs compared to other algorithms demonstrates higher degree of convergence, even at times of resource scarcity.

In order to assess how effectively the placement algorithms localized network traffic and, eventually, optimized network performance, the average network utilization of access, aggregation, and core switches were measured in the group scenario. In this part of the evaluation, the group scenario was chosen so that when any of the algorithms failed to place an AE, all the algorithms halted their placements with the purpose of keeping the total network loads imposed on the respective data centers for each of the algorithms the same. This switch-level network usage assessment was performed by scaling the mean and standard deviation of the VLs' bandwidth demands.

Finally, the average placement decision computation time for AE deployment was measured for the individual scenario. Average placement decision time is an important performance metric to assess the efficacy of NDAP as an online AE placement algorithm and its scalability across various factors.

All the above performance metrics were measured against the following scaling factors: (1) DC size, (2) mean resource demands of VMs, DBs, and VLs, (3) diversification of workloads, and (4) network distance factor. The following subsections present the experimental results and analysis for each of the experiments conducted.

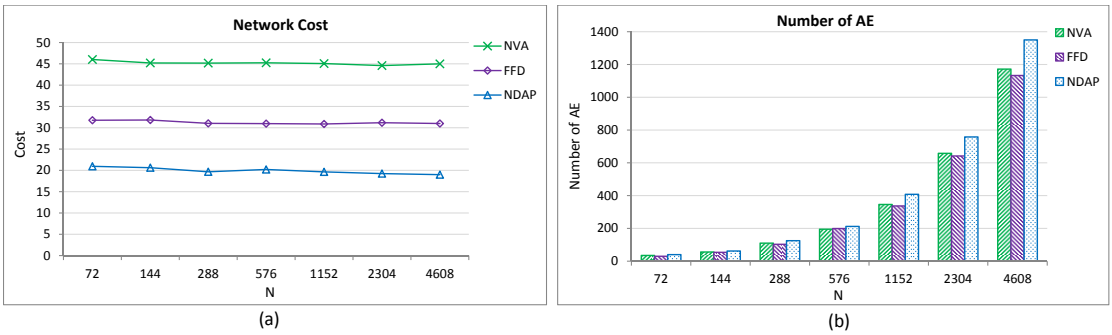

Figure 5.8: Performance with increasing N: (a) Network cost and (b) Number of AEs deployed in DC (best viewed in color).

Simulation Environment

The algorithms are implemented in Java (JDK and JRE version 1.7.0) and the simulation was conducted on a Dell Workstation (Intel Core i5-2400 3.10 GHz CPU (4 cores), 4 GB of RAM, and 240 GB storage) hosting Windows 7 Professional Edition.

5.4.3 Scaling Data Center Size

In this part of the experiment, the placement qualities of the algorithms with increasing size of the DC were evaluated and compared. As mentioned in Section 5.4.2, N was used as the only parameter to denote DC size, and its minimum and maximum values were set to 72 and 4608, respectively, doubling for each subsequent simulation phase. Therefore, in the largest DC there were a total of 5376 CNs and 1152 SNs. The other parameters $meanCom$, $meanStr$, $meanVLBW$, sd, and DF were set to 0.3, 0.4, 0.35, 0.5, and 2, respectively.

Figure 5.8(a) shows the average cost of AE placement incurred by each of the three algorithms in the group scenario for different values of N. From the chart, it is quite evident that NDAP consistently outperforms the other placement algorithms at a much higher level for the different DC sizes and its average AE placement cost is 56% and 36% less than NVA and FFD, respectively. Being network-aware, NDAP checks the feasible placements with the goal of minimizing the network cost. FFD, on the other hand, tries to place the ANs in DNs with maximum available resource capacities and, as a result, has the possibility of placing VLs on shorter PLs. Finally, NVA has random components in placement decisions and, thus, incurs higher average cost.

From Figure 5.8(b), it can be seen that the average number of successful AE deployments in the individual scenario by the algorithms increases non-linearly with the DC

size as more DNs and PLs (i.e., resources) are available for AE deployments. It is also evident that NDAP deploys a larger number of AEs in the data center compared to other algorithms until the data center is saturated with resource demands. The relative performance of NDAP remains almost steady across different data center sizes— it deploys around 13-17% and 18-21% more AEs compared to NVA and FFD, respectively. This demonstrates the fact that NDAP's effectiveness in utilizing the data center resources is not affected by the scale of the data center.

5.4.4 Variation of Mean Resource Demands

This experiment assessed the solution qualities of the placement algorithms when the mean resource demands of the AEs increased. Since the AE is composed of different components, the mean resource demands were varied in the two different approaches presented in the rest of this subsection. The other parameters, N, sd, and DF, were set to 1152, 0.4, and 2, respectively.

Homogeneous Mean Resource Demands

The same *mean* (i.e., $meanCom = meanStr = meanVLBW = mean$) was used to generate the computing (CPU and memory) resource demands of VMs, storage resource demands of DBs, and bandwidth demands of VLs under normal distribution. The experiment started with a small *mean* of 0.1 and increased it up to 0.7, adding 0.1 at each subsequent phase.

The average cost for AE placement is shown in Figure 5.9(a) for the group scenario. It is obvious from the chart that NDAP achieves much better performance compared to other placement algorithms— on average it incurs 55% and 35% less cost compared to NVA and FFD, respectively. With the increase of mean resource demands, the cost incurred for each algorithm increases almost at a constant rate. The reason for this performance pattern is that when the mean resource demands of the AE components (VMs, DBs, and VLs) increase with respect to the available resource capacities of the DC components (CNs, SNs, and PLs), the domain of feasible placements is reduced, which causes the rise in the average network cost.

Figure 5.9(b) shows the average number of AEs deployed in empty data center with increasing *mean* for the individual scenario. It can be seen from the chart that the number of AEs deployed by the algorithms constantly reduces as higher mean values are used to

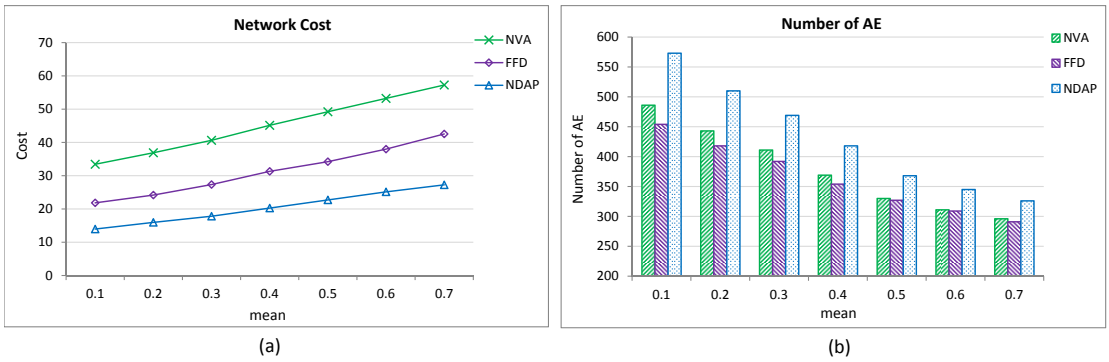

Figure 5.9: Performance with increasing mean (homogeneous): (a) Network cost and (b) Number of AE deployed in DC (best viewed in color).

generate the resource demands. This is due to the fact that when resource demands are increased compared to the available resource capacities, the DC nodes and PLs can accommodate fewer AE nodes and VLs. One interesting observation from this figure is that FFD was able to deploy fewer AEs compared to NVA when the *mean* was small. This can be attributed to the multiple random tries during AN placement by NVA, which helps it to find feasible placements, although at a higher average cost. Overall, NDAP was able to place larger numbers of AEs compared to other algorithms across all mean values: 10-18% and 12-26% more AEs than NVA and FFD, respectively.

Heterogeneous Mean Resource Demands

In order to assess the performance variations across different mean levels of resource demands of AE components, this experiment set two different mean levels L (low) and H (high) for mean VM computing resource demands ($meanCom$ for both CPU and memory), mean DB storage resource demands ($meanStr$), and mean VL bandwidth demands ($meanVLBW$). L and H levels were set to 0.2 and 0.7 for this simulation. Given the two levels for the three types of resource demands, there are eight possible combinations.

Figure 5.10(a) shows the average network costs of the three algorithms for the eight different mean levels (x axis of the chart). The three different positions of the labels are set as follows: the left-most, the middle, and the right-most positions are for $meanCom$, $meanStr$, and $meanVLBW$, respectively. As the chart shows, NDAP performs much better in terms of incurred cost than the other algorithms for each of the mean combinations. Its relative performance is highest for combinations *LHL* and *LHH*, incurring on average 67% and 52% lower costs compared to NVA and FFD, whereas its performance is lowest for combinations *HLL* and *HLH* incurring on average 42% and 25% lower costs

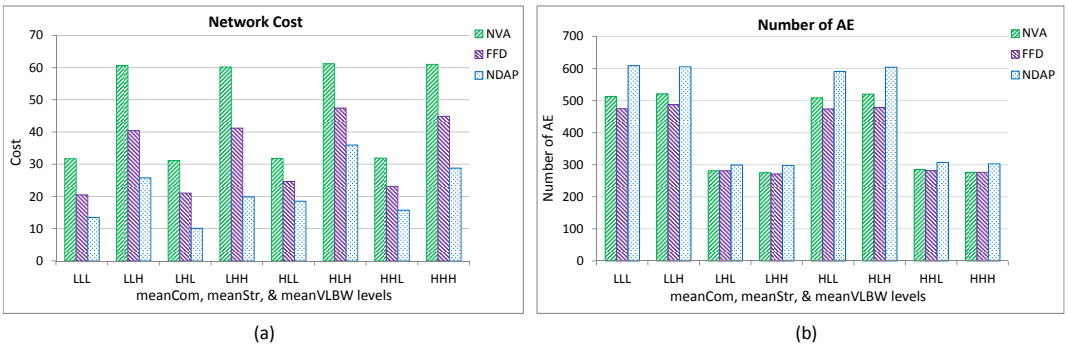

Figure 5.10: Performance with mixed levels of means (heterogeneous): (a) Network cost and (b) Number of AEs deployed in DC (best viewed in color).

compared to NVA and FFD, respectively. The reason for this pattern is the algorithmic flow of NDAP as it starts with the VDLs placement and finishes with the remaining VCLs placement. As a consequence, for relatively higher means of DB storage demands, NDAP performs relatively better.

A similar performance trait can be seen in Figure 5.10(b), which shows that NDAP places more AEs in DC compared to other algorithms. An overall pattern demonstrated by the figure is that when the $meanStr$ is high (H), the number of AEs deployed is reduced for all algorithms compared to the cases when $meanStr$ is low (L). This is because the simulated storage resources are fewer compared to the computing and network resources of DC with respect to the storage, computing, and bandwidth demands of AEs. Since NDAP starts AE deployment with the efficient placement of DBs and VDLs, on average it deploys 17% and 26% more AEs compared to NVA and FFD, respectively, when $meanStr = H$; whereas this improvement is 9% for both NVA and FFD when $meanStr = L$.

5.4.5 Diversification of Workloads

The degree of workload diversification of the deployed AEs was simulated by varying the standard deviation of the random (normal) number generator used to generate the resource demands of the components of AEs. For this purpose, initially the sd parameter was set to 0.05 and gradually increased by adding 0.05 at each simulation phase until a maximum of 0.5 was reached. The other parameters, N, $meanCom$, $meanStr$, $meanVLBW$, and DF, were set to 1152, 0.3, 0.4, 0.35, and 2, respectively.

As shown in Figure 5.11(a), the average network cost for NDAP is much lower than that for the other algorithms when the same number of AEs is deployed (as the simulation terminates when any of the algorithms fail to deploy an AE in the group scenario) as,

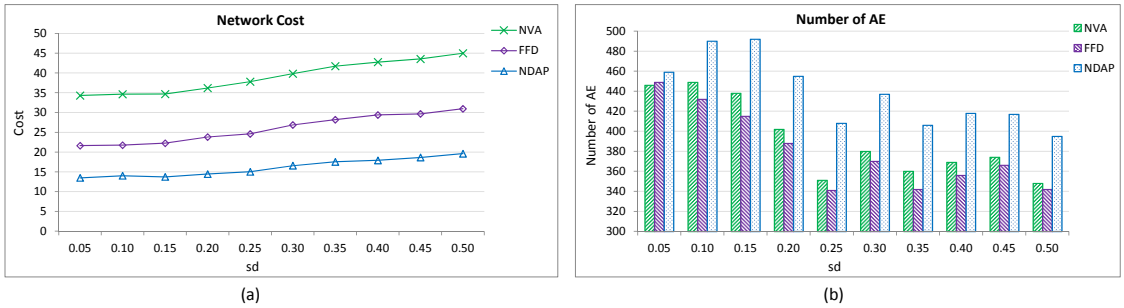

Figure 5.11: Performance with increasing standard deviation of resource demands: (a) Network cost and (b) Number of AEs deployed in DC (best viewed in color).

on average, it incurs 61% and 38% less cost compared to NVA and FFD, respectively. Moreover, for each algorithm, the cost increases with the increase of workload variations. This is due to the fact that for higher variation in resource demands, the algorithms experience reduced scope in the data center for AE component placement as the feasibility domain is shrunk. As a consequence, feasible placements incur increasingly higher network cost with the increase of the *sd* parameter.

In the individual scenario, NDAP outperforms other algorithms in terms of the number of AEs deployed across various workload variations (Figure 5.11(b)) by successfully placing on average 12% and 15% more AEs compared to NVA and FFD, respectively. Due to the random placement component, overall NVA performs better than FFD, which is deterministic by nature. Another general pattern noticeable from the chart is that all the algorithms deploy more AEs for lower values of *sd*. This is due the fact that for higher values of *sd*, resource demands of the AE components demonstrate higher variations, and as a consequence, the resources of the data center components become more fragmented during the AE placements, and thus, the utilization of these resources is reduced.

5.4.6 Scaling Network Distances

This experiment varies the relative network distance between any two data center nodes by scaling the *DF* parameter defined in Subsection 5.4.2. As the definition implies, the inter-node network distance increases with *DF* and such situations can arise due to higher delays in network switches or due to geographical distances. Initially, the *DF* value was set to 2 and increased to 16. Other parameters, N, $meanCom$, $meanStr$, $meanVLBW$, and *sd*, are set to 1152, 0.3, 0.4, 0.35, and 0.5, respectively.

Since network distance directly contributes to the cost function, it is evident from Figure 5.12(a) that the placement cost rises with the increase of the *DF* parameter in a linear

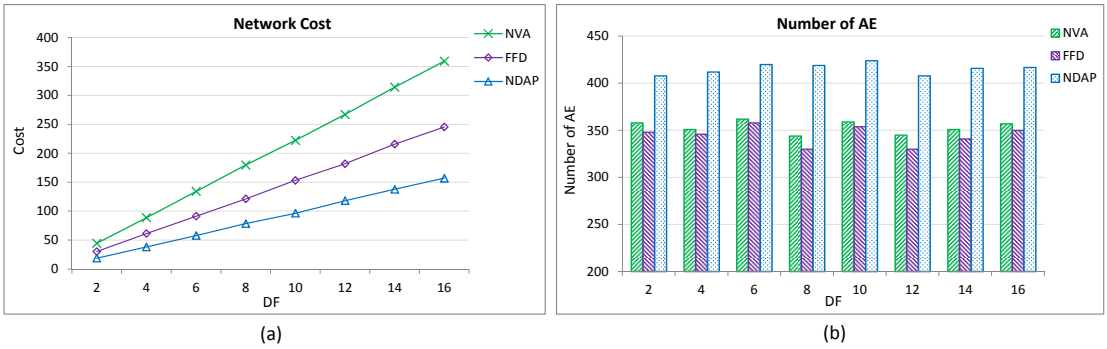

Figure 5.12: Performance with increasing distance factor (DF): (a) Network cost and (b) Number of AEs deployed in DC (best viewed in color).

fashion for the group scenario. Nevertheless, the gradients for the different placement algorithms are not the same and the rise in cost for NDAP is much lower than for other algorithms.

Figure 5.12(b) shows the average number of AEs deployed in the data center for each DF value for the individual scenario. Since network distance does not contribute to any of the resource capacities or demands (e.g., CPU or bandwidth), the number of AEs deployed remains mostly unchanged with the scaling of DF. Nonetheless, by efficient placement, NDAP outpaces other algorithms and successfully deploys 18% and 21% more AEs than NVA and FFD, respectively.

5.4.7 Network Utilization

This part of the experiment was conducted for the purpose of comparing the network utilization of the placement algorithms at the access, aggregation, and core switch levels of the data center network. This was done by stressing the network in two different scaling factors separately: the mean and the standard deviation of the VLs bandwidth demand $meanVLBW$ and $sdVLBW$, respectively. In order to ensure that the computing and storage resource demands (of VMs and DBs, respectively) did not stress the computing and storage resource capacities (of the CNs and SNs, respectively), the $meanCom$ and $meanStr$ parameters were kept at a fixed small value of 0.05, and the standard deviation $sdComStr$ for both computing and storage resource demands is were to 0.1. The other parameters, N and DF, were set to 1152 and 2, respectively.

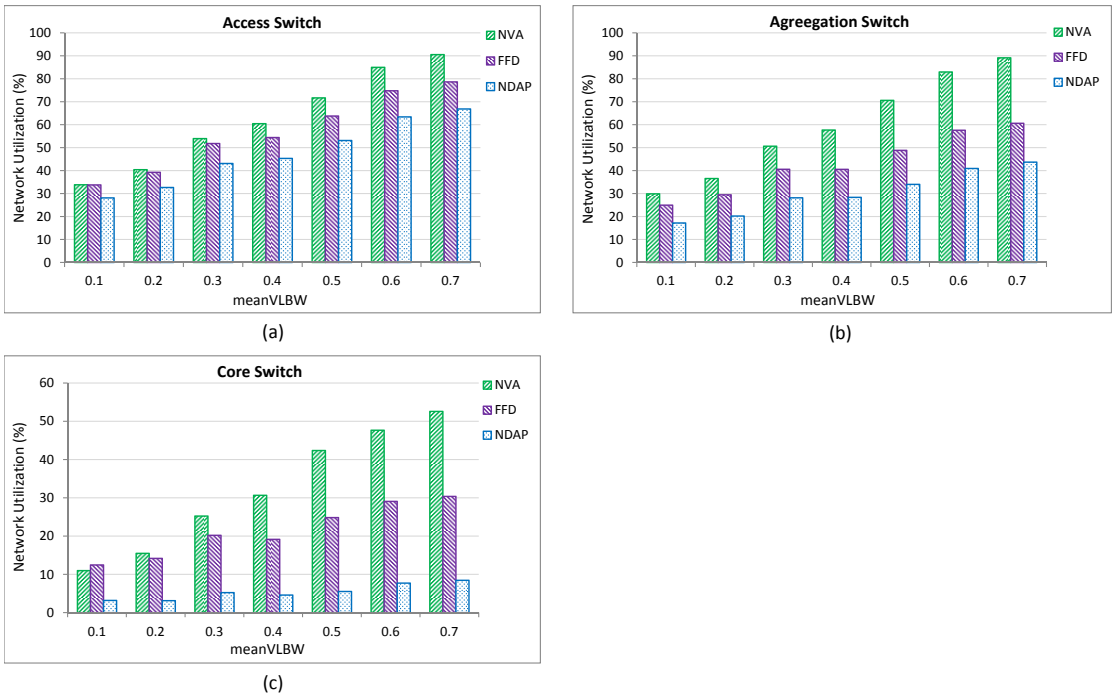

Figure 5.13: Average network utilization with increasing mean VL bandwidth demand: (a) Access switch, (b) Aggregation switch, and (c) Core switch (best viewed in color).

Scaling Mean Bandwidth Demand

This part of the experiment stressed the data center network for the group scenario where all the algorithms terminate if any of the algorithms fail to place an AE. Application of the group scenario for this experiment ensured that the total network loads imposed for each of the placement algorithms were the same when any of the algorithms failed. Initially, the mean VL bandwidth demand $meanVLBW$ was set to 0.1 and raised to 0.7, in steps of 0.1. The standard deviation of VL bandwidth demand $sdVLBW$ was kept fixed at 0.3.

Figure 5.13 shows the average network utilization of the access, aggregation, and core switches for different $meanVLBW$ values. It is evident from the charts that, for all the switch levels, NDAP incurs minimum average network utilization, and compared to NVA and FFD, NDAP placements on average result in 24% and 16% less network usage for access layer, 49% and 30% less network usage for aggregation layer, and 83% and 75% less network usage for core layer. This represents the fact that NDAP localizes network traffic more efficiently than other algorithms and achieves incrementally higher network efficiency at access, aggregation, and core switch levels. Furthermore, as the figure demonstrates, the results reflect a similar trend of performance to the results of

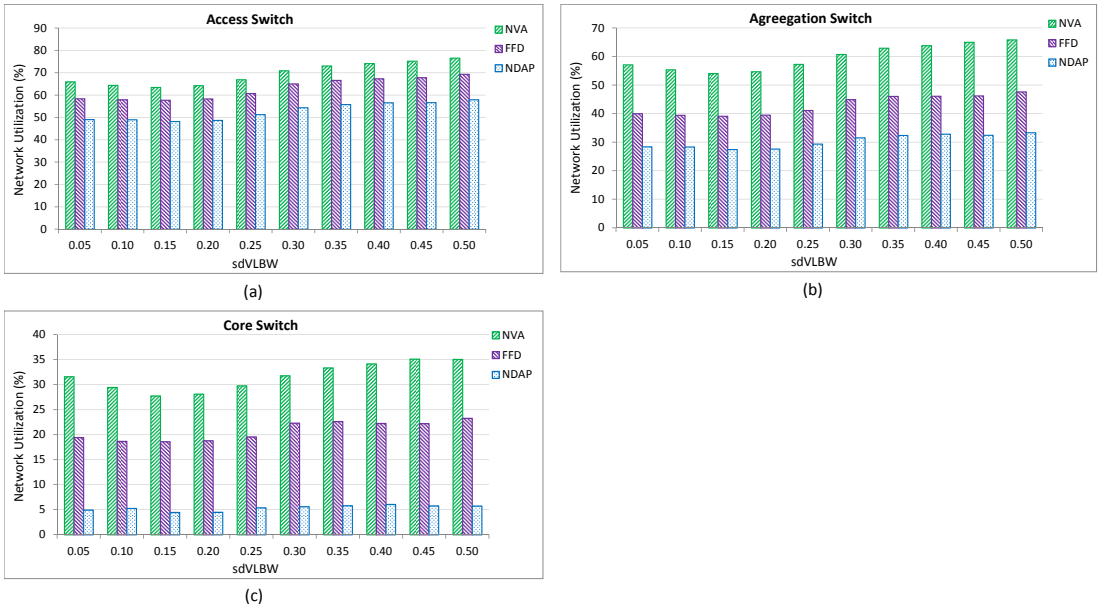

Figure 5.14: Average network utilization with increasing standard deviation of VL bandwidth demand: (a) Access switch, (b) Aggregation switch, and (c) Core switch (best viewed in color).

average network cost for placement algorithms presented in the previous subsections. This is reasonable, since network cost is proportional to the distance and bandwidth of the VLs and greater network distance indicates the use of higher layer switches during a VL placement operation. Therefore, these results validate the proposed network cost model (5.1 & 5.2) in the sense that indeed the cost model captures the network load perceived by the network switches. It can also be observed that the utilization for each switch increases with increasing $meanVLBW$. This is due to the fact that $meanVLBW$ contributes to the average amount of data transferred through the switches, since $meanVLBW$ is used as the mean to generate the VLs bandwidth demands.

Diversification of Bandwidth Demand

This experiment is similar to the above one, however, here the standard deviation of VLs bandwidth demands ($sdVLBW$) was scaled rather than the mean. Initially, $sdVLBW$ was set to 0.05 and gradually increased to 0.5, in steps of 0.05. The mean VLs bandwidth demand $meanVLBW$ was set to 0.4.

The results of this experiment are shown in Figure 5.14. The charts clearly demonstrate the superior performance of NDAP, which causes minimum network usage across all switch levels. In addition, compared to NVA and FFD, it has on average 26% and 16%

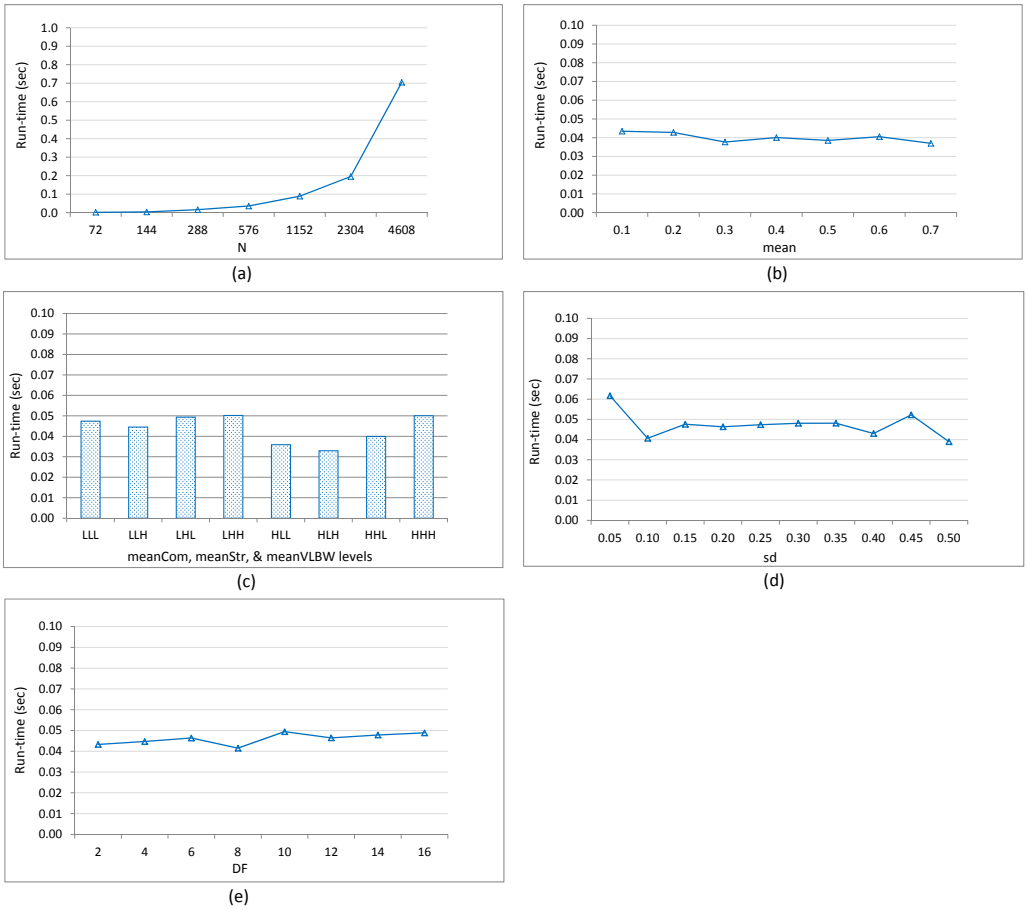

Figure 5.15: NDAP's placement decision time while scaling (a) Data center size (N), (b) Homogeneous mean ($mean$), (c) Heterogeneous mean ($meanCom$, $meanStr$, $meanVLBW$), (d) Diversification of workload (sd), and (e) Distance factor (DF).

less network usage for the access layer, 50% and 30% less network usage for the aggregation layer, and 84% and 75% less network usage for the core layer. Furthermore, the figure shows that the network utilization for each algorithm at each layer across different $sdVLBW$ values does not fluctuate much. This is due to the fact that, although the variation of VLs' bandwidth demand increases with increasing $sdVLBW$, the overall network load levels do not change much, and as a result, the average network loads perceived by the network switches at different layers differ within a small range.

5.4.8 NDAP Decision Time

In this part of the experiment, the time taken by NDAP for making AE placement decision was measured in order to assess the feasibility of using NDAP for real-time, online placement scenarios. Figure 5.15 shows the average time needed by NDAP for computing AE placements in the individual scenario by scaling all the above-mentioned

scaling factors. For each of the scaling factors, other parameters were set similar to the corresponding preceding subsections (Subsections 5.4.3-5.4.6).

From Figure 5.15(a), it can be seen that NDAP's run-time increases non-linearly with increasing data center size (other parameters were set as in Section 5.4.3). As explained in Section 5.4.1, the time complexity of the proposed NDAP algorithm is at best quadratic in terms of the number of CNs in the data center ($\mathcal{O}(N_c^2)$) and this graph effectively conforms to the complexity analysis. It is evident from the figure that for small to medium data centers, NDAP's decision-making time is in the range of a small fraction of a second, whereas for the largest data center simulated with $N = 4608$ (i.e., several thousand servers), NDAP needs only about 0.7 second. Furthermore, it can be observed from Figure 5.15(b)-(e) that NDAP's run time remains largely unaffected by other scaling factors and the run time is within the range of 0.03-0.06 second for $N = 1152$. From the above results and discussion, it can be concluded that NDAP is suitable for online AE placement scenarios, even for large data centers.

5.5 Summary and Conclusions

With the growing complexity of modern Internet applications and the increasing size of data, network resource demands in large data centers, such as Clouds, are becoming increasingly complex. Rising bandwidth requirements among application components are causing increasing pressure on the underlying communication infrastructure, making it a key area of performance bottleneck. This chapter has addressed the issue of network-focused, multi-component application placement in large data centers and formally defined it as an optimization problem. After presenting the constitutional components of the proposed network- and data-aware application placement approach, it has proposed NDAP, a fast greedy heuristic that performs simultaneous deployment of VMs and data components while respecting computing, network, and storage resource requirements and capacity constraints with the goal of minimizing the network cost incurred due to the placement decision. NDAP has been developed as a generic application placement heuristic, which is application type-agnostic, so that it can be used for a wide range of multi-tier or composite applications.

Moreover, a detailed analysis on the computational complexity for each of the compared algorithms in terms of run-time and memory overhead has been presented. Analysis has

revealed that the proposed NDAP algorithm has quadratic time complexity, which is slightly higher than those of the compared algorithms (linear and linearithmic), and the memory overheads are the same for each of the algorithms.

Furthermore, an extensive simulation-based performance evaluation has been presented, and the results demonstrate the superior performance of NDAP over competitor approaches across multiple performance metrics and scaling factors. NDAP reduces the network cost by up to 67% and successfully deploys up to 26% more applications compared to other approaches. The effectiveness of NDAP in minimizing the overall communications overhead has been validated for two different representative application environments: multi-tier and scientific (Montage) workflows. In addition, among all the placement algorithms, NDAP achieves best network utilization by reducing network usage of core, aggregation, and access switches by up to 84%, 50%, and 26%, respectively. Furthermore, the feasibility of applying NDAP in online or real-time scenarios for large-scale data centers has been demonstrated by performance evaluation, which has shown that NDAP requires only a fraction of a second to make effective multi-component application placement decisions.

While online VM placement strategies, such as the ones presented in this chapter and in the previous one, can optimize VM placement at the initial deployment phase, active VMs can exhibit variations in run-time resource usage during VM life cycle due to potential workload variations. Moreover, Clouds are dynamic environments where VMs are created and terminated in the data centers based on customer requests. As a result, physical server resources become fragmented over time, which eventually reduces overall resource utilization and hosting capacity of the data center. Such run-time underutilization of computing resources is one of the main reasons for very high resource wastage and power consumption. The next chapter addresses this run-time underutilization problem and proposes a multi-objective dynamic VM consolidation scheme utilizing VM live migration technique. Since VM migration has non-negligible impact on hosted applications and data center components, a migration overhead estimation model is also presented by taking into account realistic migration parameters.

Chapter 6

Multi-objective, Decentralized Dynamic Virtual Machine Consolidation

This chapter deals with the problem of offline, migration impact-aware, multi-objective dynamic Virtual Machine (VM) consolidation in the context of large-scale virtualized data center environments. The problem is formulated as an NP-hard discrete combinatorial optimization problem with simultaneous objectives of minimizing resource wastage, power consumption, and VM migration overhead. As solution approach, a VM live migration overhead estimation technique is proposed, which takes into account pragmatic migration parameters and overhead factors. In order to tackle scalability issues, a hierarchical, decentralized dynamic VM consolidation framework is presented that helps to localize migration related network traffic and reduce network cost. Moreover, a multi-objective, dynamic VM consolidation algorithm is proposed by utilizing the Ant Colony Optimization (ACO) metaheuristic, with integration of the proposed VM migration overhead estimation technique. Comprehensive performance evaluation makes it evident that the proposed dynamic VM consolidation approach outpaces the state-of-the-art offline, migration-aware dynamic VM consolidation algorithm across all performance metrics by reducing the overall power consumption by up to 47%, resource wastage up to 64%, and migration overhead up to 83%.

6.1 Introduction

The previous two chapters presented two online VM placement strategies focusing on optimization of resource utilization, energy efficiency, and network load in data centers. Complementary to those VM placement strategies, this chapter presents an offline, decentralized, dynamic VM consolidation framework and an associated VM consolidation algorithm, leveraging the VM live migration technique with the goal of optimizing the run-time resource utilization and energy consumption within the data center, while at the same time minimizing the associated VM live migration overhead required to achieve the consolidation. In order to increase scalability and reduce migration overhead, servers are grouped into clusters based on mutual network cost of data communication and VM consolidation is performed within each group separately. Moreover, in order to reduce the overall migration impact on the hosted applications and the data center network, VM migration overhead estimation models are presented using realistic parameters. Furthermore, a multi-objective, dynamic VM consolidation algorithm is proposed by adapting the *Ant Colony Optimization* (ACO) metaheuristic coupled with the proposed migration overhead estimation models.

While online VM placement and allocation techniques, such as the one presented in Chapter 4 and Chapter 5, have potentials to optimize placement decisions at the time of VM initiation, active VMs exhibit variations in actual resource usage during the VM life cycle due to workload variations. Furthermore, due to the features of on-demand resource provisioning and a pay-per-use business model, VMs are created and terminated dynamically and, as a consequence, data center resources become fragmented, which leads to degradation of server resource utilization and overall hosting capacity of the data center. Such underutilization of computing resources is one of the main reasons for high resource wastage in enterprise data centers. A recent measurement study [30] shows that VMs in Google data centers only utilize around 35% and 55% of the requested CPU and memory resources. Moreover, due to the narrow dynamic power range of physical servers, underutilized servers cause non-proportional power consumption in data centers. Another measurement study showed that even completely idle servers consume about 70% of their peak power usage [42]. Both the problems of run-time server resource wastage and power consumption can be addressed by improving server resource utilization through the application of dynamic VM consolidation.

Unlike the consolidated VM cluster placement problem presented in Chapter 4, the *Dynamic VM Consolidation* problem focuses on run-time environments where VMs are active and already hosted by servers in the data center. Consolidation of such VMs are achieved by the VM live migration operations [26, 93], where a running VM is relocated from its current host to another server while it is still running and providing service to its consumers [123]. After the consolidation of the VMs, servers that are released by migrating their hosted VMs to other servers are turned to lower power states, such as standby or turned off, in order to save energy. Moreover, such consolidation improves the overall resource utilization of the active servers and resource wastage is minimized. Besides these obvious advantages, dynamic VM consolidation has other benefits such as data center physical space minimization, maintenance automation, and reduced labor costs.

Nonetheless, this adds additional complexity to the consolidation operation since the current placement of the VMs needs to be considered while VM migration decisions are made. This is because VM migration operations incur migration impact or cost on both the hosted applications and data center components, such as the host server and communication network [122]. As a consequence, any efficient dynamic VM consolidation must consider both the gain achieved by consolidating the VMs into a reduced number of servers and the overhead of the necessary VM migrations needed to achieve the consolidation. Therefore, such dynamic VM consolidation techniques need to be *multi-objective*, where they opt for maximizing the gain of energy saving and resource utilization, as well as reducing the cost or overhead of necessary VM migrations.

Most of the existing works on multi-objective dynamic VM consolidation try to rearrange active VMs into the minimum number of servers in order to save energy, by turning the idle servers to lower power states while reducing the number of VM migrations needed to perform the VM consolidation operation [12, 43, 45, 82, 90, 94]. A profound limitation that exists in these approaches is that every VM migration is considered equivalent in terms of migration cost or overhead. Experimental studies [122, 124] on VM live migration shows that a migrating VM experiences performance degradation during its total migration time. For example, performance analysis on migrating a VM that hosts a web server reveals that the server throughput can drop by 20% during the migration period [26]. Moreover, a VM live migration operation results in a VM down period (formally, VM

downtime) during which the services hosted by the VM remain unavailable to the consumers [93]. Furthermore, VM migration causes extra energy consumption for the servers and network devices, as well as generation of additional network traffic for transferring VM memory pages.

In an environment where different categories of applications are hosted by the running VMs, such as Cloud data centers, VMs exhibit high variations in their resource usage patterns and performance footprints. As a consequence, different VM migrations have different performance overhead both on the hosted services and on the data center components. A recent study [128] has attempted to consider the VM migration cost while making consolidation decisions by incorporating a couple of VM properties into its consolidation scoring method. However, it overlooked data center properties, such as migration link bandwidth and network costs, as well as the dynamic nature of VM migration. Moreover, the evaluation is based solely on score values, rather than realistic properties. Therefore, dynamic VM consolidation approaches need to consider realistic measures of individual VM migration costs or overheads for making practical and efficient consolidation decisions.

Contrary to the existing methods mentioned above, the approach presented in this chapter considers various migration parameters relating to VMs and data center components in order to estimate extent of realistic migration overheads. Such migration parameters and overheads are adopted from the insights demonstrated by previous measurement studies [4, 78, 120] on VM live migration in order to ensure the estimation technique is as pragmatic as possible. Such a realistic measure has obvious benefits over simplistic measures, such as the number of migrations, in that it can reveal the VM migrations that are beneficial for containing the migration overhead to a limited extent while at the same time improving server resource utilization. This is important since, in a dynamic data center environment, there can be instances where two or more VM migrations may have lower migration overhead than a different individual VM migration, an occurrence which it cannot be determined with a simplistic measure that considers only the number of migrations.

Moreover, the migration overhead estimation method is further integrated with the proposed multi-objective, dynamic VM consolidation algorithm that generates migration plans for the VMs running in the data center. The primary benefit of this methodology is

that it adopts a practical approach to quantifying the cost or impact of each VM migration in the data center that can be readily integrated with any other dynamic VM consolidation technique. Furthermore, unlike many of the existing studies [12, 82, 90, 94] that suggest greedy heuristics, the proposed dynamic VM consolidation technique adapts the multi-agent-based *Ant Colony Optimization* (ACO) metaheuristic that works, in multiple iterations, based on solution refinement method. Utilization of such a refinement-based metaheuristic helps the algorithmic procedure to avoid early stagnation at local optima.

Furthermore, in order to address the scalability issue of data center-wide dynamic VM consolidation, a hierarchical, decentralized dynamic VM consolidation framework is proposed where the servers of the data center are grouped into clusters and it is recommended that VM consolidation operations be performed individually within the clusters. A network cost-aware cluster formation approach is suggested in this chapter in order to localize the VM migration related network traffic within the lowest level of the network topology. This clustering approach has the advantage that migration-related network traffic does not travel through upper-layer network switches, and thereby avoiding data center network clogging. Moreover, such traffic does not need to travel long distances and therefore reduces the related network cost. Having said that, the proposed framework is not restricted to any cluster formation approach and any other dynamic clustering technique can be readily integrated with the framework, as well as with the dynamic VM consolidation algorithm.

The proposed techniques and strategies for multi-objective dynamic VM consolidation are built upon some assumptions regarding the data center environment. The migration overhead estimation technique assumes that the active memory size and page dirty rate of each VM running in the data center are known *a priori*. In a virtualized data center, this information can be made readily available by utilizing virtualization Application Programming Interfaces (APIs), such as Red Hat *libvirt*[1], or using virtualization tools, such Red Hat *virt tools*[2]. It is further assumed that the inter-server network distance and available network bandwidth for performing VM live migrations are also known prior to the migration overhead estimation. Such network information can be measured using network tools, for example, the *iPerf*[3] network testing tool can be used to measure the

[1] libvirt: The virtualization API, 2016. `http://libvirt.org/`
[2] virt tools: Open Source Virtualization Management Tools, 2016. `http://virt-tools.org/`
[3] iPerf, 2016. `https://iperf.fr/`

maximum achievable bandwidth on IP networks, and the *MTR*[4] network monitoring tool can measure the end-to-end network distance in terms of number of hops or in terms of delay in packet forwarding.

Moreover, it is assumed that the hypervisors (e.g., Xen [9], KVM [70], etc.) running in the data center servers are homogeneous. This is important to ensure the compatibility of the VM live migration operations among servers. In such an environment, it is presumed that several hypervisor properties relating to the VM live migration operation are already known, such as the remaining dirty memory threshold and the maximum number of rounds for the pre-copy migration. Furthermore, the dynamic VM consolidation algorithm takes into account the current resource demands of the active VMs in the data center, such as the CPU, main memory, and network I/O. In addition, the consolidation algorithm also needs to know the usable resource capacities of the server running in the data center. Last but not least, it is assumed that the data center network topology is already known for successful application of the proposed hierarchical, decentralized VM consolidation framework.

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

1. The *Multi-objective, Dynamic VM Consolidation Problem* (MDVCP) is formally defined as a discrete combinatorial optimization problem with the objective of minimizing data center resource wastage, power consumption, and overall migration overhead due to VM consolidation.

2. VM migration overhead estimation models are proposed with consideration of realistic migration parameters and overhead factors in the context of the pre-copy VM live migration technique. The estimation models are not restricted to any specific VM consolidation method and can be readily integrated to any online or offline consolidation strategies.

3. A hierarchical, decentralized VM consolidation framework is proposed to improve the scalability of dynamic VM consolidation in the context of medium to large-scale data centers.

4. A novel *ACO-based, Migration overhead-aware Dynamic VM Consolidation* (AMDVMC) algorithm is put forward as a solution to the proposed MDVCP problem. The

[4]MTR, 2016. `http://www.bitwizard.nl/mtr/`

AMDVMC algorithm is integrated with the recommended decentralized consolidation framework and utilizes the proposed migration overhead estimation models.

5. Extensive simulation-based experimentation and performance analysis is conducted across multiple scaling factors and several performance metrics. The results suggest that the proposed dynamic VM consolidation approach significantly optimizes the VM allocations by outperforming the compared migration-aware VM consolidation techniques across all performance metrics.

The rest of this chapter is organized as follows. The next section introduces the multi-objective, dynamic VM consolidation problem and presents the necessary mathematical frameworks to model it as a combinatorial optimization problem. Section 6.3 describes the proposed VM live migration estimation models, the hierarchical, decentralized dynamic VM consolidation framework, and the ACO-based multi-objective, dynamic VM consolidation algorithm. Section 6.4 presents the performance evaluation and analysis of the results where the proposed dynamic VM consolidation approach is compared with other state-of-the-art approaches. Finally, Section 6.5 concludes the chapter with a summary of the contributions and results.

6.2 Multi-objective, Dynamic VM Consolidation Problem

By the use of dynamic VM consolidation, active VMs are live migrated from one server to another to consolidate them into a minimal number of servers in order to improve overall resource utilization and reduce resource wastage. Servers released by this process can be turned to lower power states (such as suspended or turned off) with the goal of minimizing the overall power consumption. For example, in Figure 6.1, 10 VMs are running in 5 servers, each having an overall resource utilization of not more than 65%. The VMs can be reassigned to be consolidated into 3 servers resulting in higher utilization and, in this way, 2 servers can be released and turned to lower power state to save energy.

However, dynamic VM consolidation at run-time is not merely a vector packing problem, such as the consolidated VM placement problem as presented in Chapter 4, since it needs to consider the current VM-to-server placements and the impact of necessary VM migrations on the performance of the hosted applications and the data center network [26]. Current dynamic VM consolidation approaches [45,82,90] try to pack VMs into a minimal

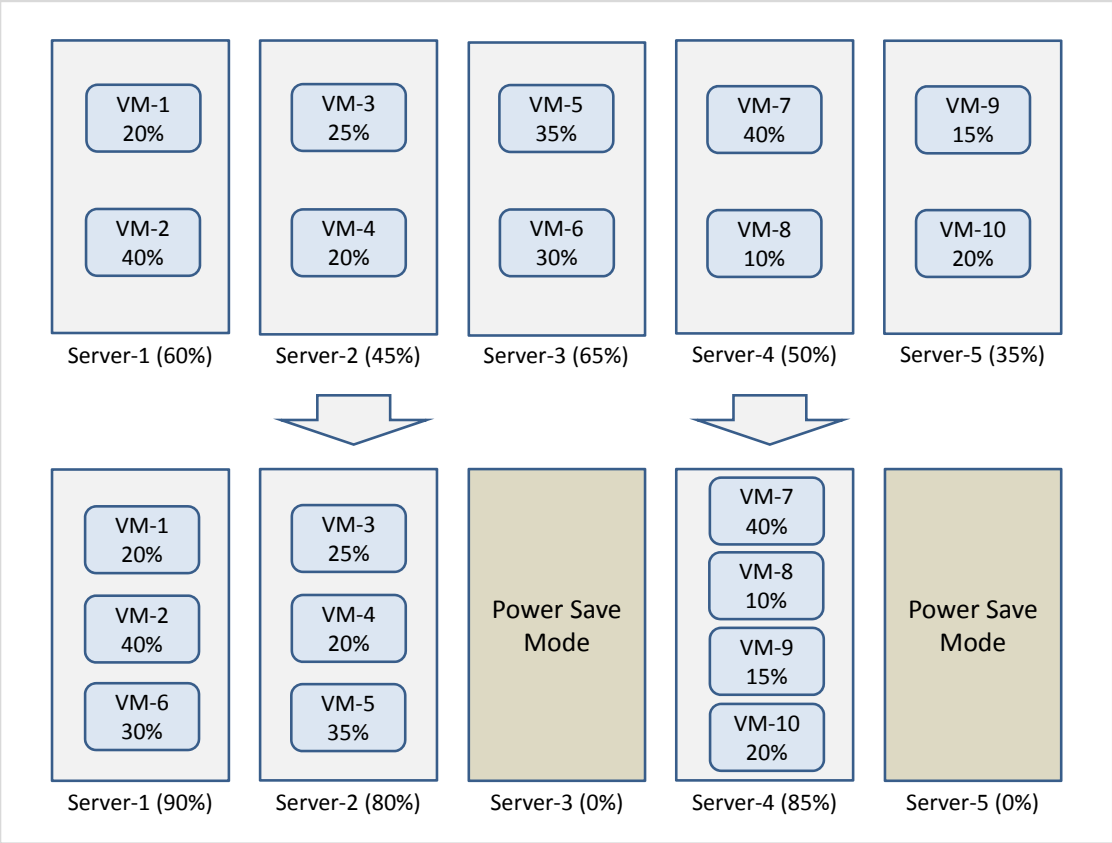

Figure 6.1: Improving resource utilization and energy consumption through dynamic VM consolidation.

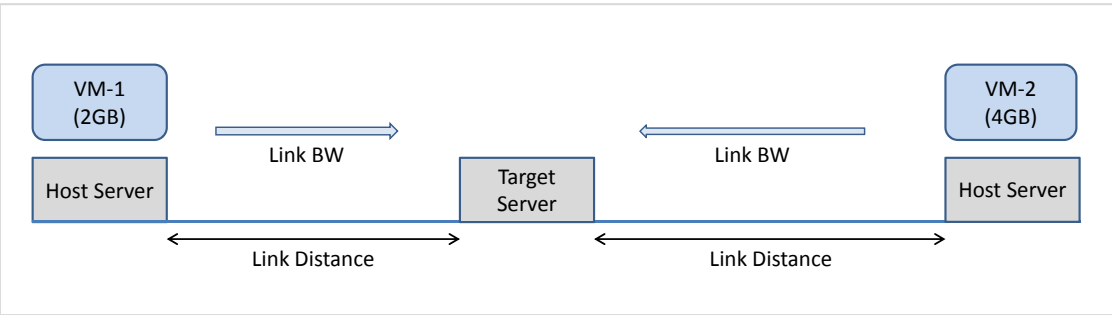

Figure 6.2: VM live migration decision based on VM characteristics.

number of servers while reducing the number of migrations. However, the overhead of a live VM migration varies, based on some specific characteristics of the migrating VM and the corresponding network link used for migrating VM memory pages from the source server to the destination server. For example, a VM with 2 GB memory will be migrated faster than a VM with 4 GB memory, given that the other conditions are exactly same and, thus it needs less migration time and also requires fewer memory pages to be transferred from the source server to the destination server (Figure 6.2). For these reasons, merely

considering the number of migrations needed for consolidation is an oversimplified metric to measure the impact of the necessary VM migrations. Therefore, in order to estimate the overhead or impact of the overall VM migrations required for achieving a particular VM consolidation state, it is important to estimate a realistic measure of the overhead of a single VM migration.

Given the above insights into the dynamic VM consolidation technique, the remaining part of this section formally defines the MDVCP problem with the necessary notations and models.

6.2.1 Modeling Multi-objective, Dynamic VM Consolidation as a Combinatorial Optimization Problem

This subsection presents the mathematical framework for representing the dynamic VM consolidation problem as a discrete combinatorial optimization problem.

Let *PMS* denote the set of active servers or *Physical Machines* (PMs) in a data center and *VMS* denote the set of active VMs running on those PMs. *RCS* represents the set of d types of resources available in each PM. Table 6.1 provides the various notations used in the problem definition and proposed solution.

Each PM p ($p \in PMS$) has a d-dimensional *Resource Capacity Vector* (RCV) $C_p = \{C_p^r\}$, where C_p^r denotes the total capacity of resource r ($r \in RCS$) of p. Similarly, each VM v ($v \in VMS$) is represented by its d-dimensional *Resource Demand Vector* (RDV) $D_v = \{D_v^r\}$, where D_v^r denotes the demand of resource r ($r \in RCS$) of v. Moreover, memory page dirty rate and current host PM for a VM v are denoted by v^{dr} and v^{hp}, respectively.

The set of VMs hosted by a PM p is denoted by HV_p. The *Resource Utilization Vector* (RUV) of p is denoted by $U_p = \{U_p^r\}$, where U_p^r denotes the utilization of resource r ($r \in RCS$) and is computed as the sum of the RDVs of its hosted VMs:

$$U_p^r = \sum_{v \in HV_p} D_v^r. \tag{6.1}$$

In this modeling, the data center is not restricted to any fixed network topology. Network distance and available network bandwidth used for VM live migration operations between any two PMs p_1 and p_2 are represented by $DS(p_1, p_2)$ and $BA(p_1, p_2)$. This network distance can be any practical measure, such as the number of hops or switches or

Table 6.1: Notations and their meanings

Notation	*Meaning*
v	Individual Virtual Machine
VMS	Set of active VMs in a data center
v^{cpu}	CPU demand of a VM v
v^{mem}	Memory demand of a VM v
v^{dr}	Page Dirty Rate of a VM
v^{hp}	Host PM of a VM
N_v	Total number of VMs in a data center
V	Set of active VMs in a PM cluster
N_{vc}	Number of VMs in a PM cluster
p	Individual Physical Machine
PMS	Set of active PMs in a data center
HV_p	Set of VMs hosted by PM p
N_p	Total number of PMs in a data center
P	Set of PMs in a PM cluster
N_{pc}	Number of PMs in a cluster
r	Single computing resource in PM (e.g., CPU, memory, network I/O)
RCS	Set of computing resources available in PMs
d	Number of resource types available in PM
$DS(p_1, p_2)$	Network distance between PMs p_1 and p_2
$BA(p_1, p_2)$	Available bandwidth between PMs p_1 and p_2
$OG(v, p)$	Overall gain of assigning VM v to PM p
$UG_p(v)$	Utilization gain of PM p after VM v is assigned in it
$MO(v, p)$	Migration overhead incurred due to transferring VM v to PM p
f_3	MDVCP Objective Function
MD	Amount of VM memory (data) transferred during a migration
MT	Total time needed for carrying out a VM migration operation
DT	Total duration during which VM is turned down during a migration
NC	Network cost that will be incurred for a migration operation
MEC	Energy consumption due to VM migration
MSV	SLA violation due to VM migration
MM	Migration map given by a VM consolidation decision

network link latency in the communication path between p_1 and p_2. Thus, the network distance DS and available bandwidth BA models are generic and different model formulations focusing on any particular network topology or architecture can be readily applied in the optimization framework and proposed solution. Although singular distance between two PMs is considered here, link redundancy and multiple communication paths in data centers can be incorporated in the proposed model and the consolidation algorithm by appropriate definition of the distance function (DS) and the available bandwidth function (BA).

Given the above models and concepts, the objective of the MDVCP problem is to search for a VM migration decision for all the VMs in the data center that maximizes the

number of released PMs (that can be turned to lower power states) at a minimal overall migration overhead, while respecting the PM resource capacity constraints. Therefore, the *Objective Function* (OF) f_3 of the MDVCP problem can be expressed as follows:

$$\textbf{maximize}\, f_3(MM) = \frac{nReleasedPM^{\phi}}{MO(MM)} \tag{6.2}$$

where MM is the *Migration Map* for all the VMs in the data center which is defined as follows:

$$MM_{v,p} = \begin{cases} 1, & \text{if } VM\ v \text{ is to be migrated to } PM\ p; \\ 0, & \text{otherwise.} \end{cases} \tag{6.3}$$

$MO(MM)$ represents the overall migration overhead of all the VM migrations denoted by migration map MM which are necessary for achieving the consolidation and is expressed by (6.13). Details on measuring an estimation of the migration overhead ($MO(MM)$) is presented in the next section. And, ϕ is a parameter that signifies the relative importance between the number of released PMs ($nReleasedPM$) and migration overhead (MO) for computing the OF f_3.

The above-mentioned OF is subject to the following PM resource capacity constraints:

$$\sum_{v \in VMS} D_v^r MM_{v,p} \leq C_p^r, \forall p \in PMS, \forall r \in RCS. \tag{6.4}$$

The above constraint ensures that the resource demands of all the VMs that are migrated to any PM do not exceed PM's resource capacity for any of the individual resource types. And, the following constraint guarantees that a VM is migrated to exactly one PM:

$$\sum_{p \in PMS} MM_{v,p} = 1, \forall v \in VMS. \tag{6.5}$$

For a fixed number of PMs in a data center, maximization of the number of released PM ($nReleasedPM$) otherwise means minimization of the number of active PMs ($nActivePM$) used for hosting the N_v VMs. Moreover, as argued in Chapter 4, Subsection 4.2.1, minimization of the number of active PMs otherwise indicates minimization of the power consumption and resource wastage of the active PMs in a data center, as well as maximization of packing efficiency (PE). Thus, the above OF f_3 models the addressed MDVCP problem as a multi-objective problem. Moreover, it is worth noting that f_3 represents an

expression of multiple objectives with potentially conflicting goals— it is highly likely that maximization of the number of released PMs would require VM migrations that have large migration overhead. Therefore, any solution, to be efficient in solving the MDVCP problem, would require it to maximize the number of released PMs with minimal migration overhead.

Within the above definition, the MDVCP is represented as a discrete combinatorial optimization problem since the objective is to find a migration map (i.e., the optimal solution) from the finite set of all possible migration maps (i.e., solution space) that gives maximum value for the OF f_3. Furthermore, it is worth noting that the search space of the problem increases exponentially with N_v and N_p. Effectively, the MDVCP problem falls in the category of $\mathcal{NP}$–$hard$ problems for which no exact solution can be obtained in realistic time.

6.3 Proposed Solution

A dynamic VM consolidation mechanism requires running VMs to be migrated and consolidated into fewer PMs so that empty PMs can be turned to lower power states in order to save energy. However, VM live migration impacts hosted applications, requires energy to transfer VM memory pages, and increases network traffic. Furthermore, these migration overheads vary from VM to VM, depending on several migration related parameters, such as VM memory size and the available bandwidth of the network link used for the migration. Therefore, dynamic VM consolidation schemes need to know a measure of the overhead for each VM migration in order to reduce the overall migration impact.

In light of the above discussion, this section first presents a VM live migration overhead estimation model considering the relevant migration parameters. Secondly, in order to improve scalability of the dynamic VM consolidation algorithm and reduce network traffic incurred due to required VM migrations, a PM clustering scheme is presented that groups PMs in the data center based on the inter-PM network distances and dynamic consolidation being performed within each cluster locally. Finally, the migration impact-aware dynamic VM consolidation algorithm (AMDVMC) is proposed by utilizing the various models presented in this chapter and in Chapter 4.

6.3.1 VM Live Migration Overhead Estimation

VM Live Migration [26] is a powerful feature of virtualization platforms that allows an active VM running live application services to be moved around within and across data centers. As presented in Section 2.4, compared to *Static* or *Cold VM Migration* [112], where the VM to be migrated is turned off and merely a configuration file is transferred to the destination PM to restart the VM, live migration ensures that the VM remains running during most of the migration duration and experiences a shorter downtime during which the hosted services become unavailable. However, live migration techniques may result in higher total migration time since VM memory pages are transferred from the source PM to the destination PM, either in multiple iterations [26] or on-demand based on page fault at the destination PM [61,131]. Nevertheless, because of the obvious benefit of uninterrupted service and shorter VM downtime, live migration is widely used for VM management and dynamic reconfiguration in production data centers.

Nonetheless, VM live migration has a negative impact on the performance of applications running in a VM during the migration duration, on the underlying communication network due to the traffic resulting from transferring VM memory pages, as well as energy consumption due to carrying out the migration operation [78]. These migration overheads can vary significantly for different application workloads due to the variety of VM configurations and workload patterns. For example, previous measurement studies on VM live migration demonstrated that VM downtime can vary significantly among workloads due to the differences in memory usage patterns, ranging from 60 milliseconds for a Quake 3 game server [26] to 3 seconds in the case of high-performance computing benchmarks [92]. Another experimental study showed that applications hosted by migrating VMs suffer from performance degradation during the whole migration duration [124]. As a consequence, it is important to identify the relevant parameters that affect the migration process and the migration overhead factors that result from the process. To this end, the remaining part of this subsection presents a brief overview of the VM live migration process and details on the proposed migration overhead estimation models.

Single VM Migration Overhead Estimation

Among the various VM live migration techniques (Subsection 2.4.1), *Pre-copy VM Migration* has been the most popular and widely used as the default VM migration subsystem in modern hypervisors, such as XenMotion [26] for Xen Server and VMotion [93] for

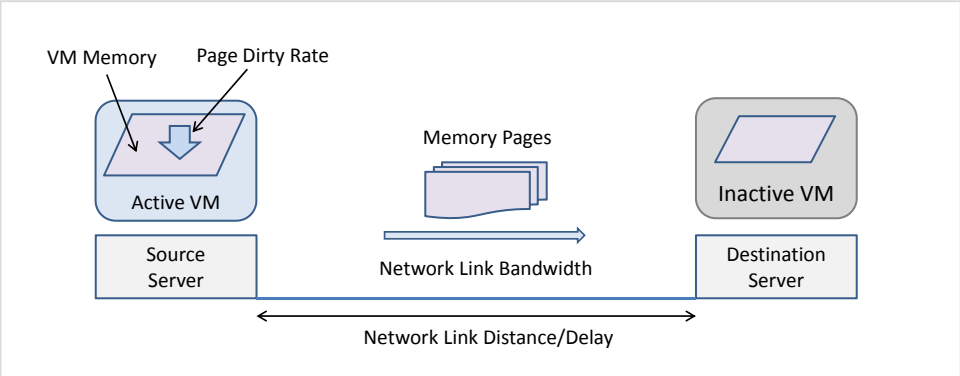

Figure 6.3: Pre-copy VM live migration techniques and factors that affect migration impact.

VMware ESXi Server. In this technique, the migrating VM continues to run in the source PM while the VM memory pages are iteratively transferred from the source to the destination (Figure 6.3). After a predefined number of iterations are completed, or a pre-specified amount of dirty memory remains to be transferred, or any other terminating condition is met, the VM is stopped at the source, the remaining memory pages are moved to the destination and, finally, the VM is restarted in the destination PM. The obvious benefits of this technique are the relatively short stop-and-copy phase and, therefore, shorter VM downtime compared to other live migration techniques, and higher reliability as it retains an up-to-date VM state in the source machine during the migration process. However, pre-copy migration can require longer total migration time since memory pages can be transmitted multiple times in several rounds depending on the page dirty rate and, for the same reason, it can generate much higher network traffic compared to other approaches. Therefore, given the migration parameters (e.g., VM memory size and page dirty rate) that affect VM live migration, it is necessary to estimate measures of migration overhead factors properly (e.g., total migration time and VM downtime) in order to decide which VMs to migrate for dynamic consolidation so as to reduce the overhead incurred due to the migration operation.

VM Migration Parameters There are several migration parameters that affect migration performance and, hence, the accuracy of migration overhead estimation [4]:

1. *VM Memory Size (v^{mem}):* In the first iteration, the pre-copy migration scheme transfers the whole VM memory from the source PM to the destination PM and,

thus, the duration of the first iteration is directly proportional to the memory size. As a result, the memory size impacts the total migration duration and on average, this duration varies linearly with the memory size. Moreover, a larger memory indicates that more network traffic will be generated for performing the migration operation.

2. *VM Page Dirty Rate (v^{dr}):* After the first iteration, only the memory pages that are modified (i.e., dirty) during an iteration are copied from the source to the destination PM in the next iteration. Thus, a higher page dirty rate causes more data transfer per iteration and results in a longer total migration duration. Furthermore, a higher page dirty rate indicates that more memory pages need to be transferred in the last transfer round and, as a consequence, this increases the VM downtime.

3. *Migration Link Bandwidth (BA):* For each of the pre-copy migration data transfer rounds, a higher link bandwidth will enable faster data transmission and shorten the round duration. As a result, both the total migration time and VM downtime will be reduced. Thus, the migration link bandwidth is inversely proportional to the total migration time and VM downtime.

4. *Migration Link Distance (DS):* Since VM migration causes a non-negligible amount of data transfer through the communication network, it incurs a traffic overhead on the network links of the data center. In this context, the migration link distance can refer to the physical distance that migration-related data needs to be transferred or the latency/delay in data communication from the source PM to the destination PM. Therefore, the migration link distance has a direct effect on the overall network overhead for the migration.

Apart from the above-mentioned general parameters, the VM migration impact can vary based on two migration configuration parameters of the specific hypervisor:

1. Threshold value for the remaining amount of dirty memory (DV_{th}), and

2. Maximum number of rounds for the pre-copy migration algorithm (max_round).

When the pre-copy migration process reaches either of the above two points, the VM is stopped, the remaining dirty memory is transferred, and the VM is restarted in the destination PM (termed *stop-and-copy* phase). However, for a data center with homogeneous

hypervisors, these two parameters can be considered predefined and fixed and, therefore, these are considered as constants in the proposed overhead estimation model.

Migration Overhead Factors Given the above migration parameters, the proposed model estimates the following four migration overhead factors that contribute to overall migration overhead:

1. *Migration Data Transferred (MD):* As the pre-copy migration process involves multiple rounds for sending dirtied memory in the previous rounds from the source to the destination, the total amount of data transferred due to the migration can be equal to or greater than the VM memory size. This factor has a direct impact on the energy consumption and network overhead incurred due to the migration.

2. *Migration Time (MT):* This implies the total duration for the migration from the initiation of the migration process to the point when the migrated VM has started running in the destination PM. This is an important element of the overall migration impact since the migrating VM suffers from performance degradation during the migration duration [12].

3. *VM Downtime (DT):* This represents the time duration for which the VM would be halted and the hosted service would be unavailable to the consumers. VM downtime is composed of the time duration needed for the stop-and-copy phase to transfer the remaining dirty memory in the last iteration and the time spent in resuming the VM at the destination PM (Section 2.4).

4. *Network Cost (NC):* The network cost or overhead of a VM migration is modeled as the product of the total data (i.e., memory pages) transferred (MD) from the source PM to the destination PM during the migration process and the network distance (DS) between PMs. Since data center networks are usually designed in a hierarchical fashion (e.g., tree topology [5]), VM migrations that involve transmission of migration data through higher layer network switches (e.g., core/aggregation switch) incur more network cost compared to migrations that involve data transmission among PMs under the same access switches.

The model for estimating single VM migration overhead follows the internal operational steps of the pre-copy migration technique and extends the VM live migration performance

modeling presented by Liu *et al.* [78]. The algorithmic steps of the process (VMMigOverhead) is presented in Algorithm 6.1. As input, it takes the above-mentioned migration parameters and the destination PM, and computes estimates for the above-mentioned migration overhead factors. The algorithm starts by initializing MD and MT to zero which store the estimates of the total data to be transmitted and time duration for the whole migration process.

After setting p_s to the current host PM of the VM, the VMMigOverhead algorithm checks whether the source and destination PMs are the same [lines 1–2]. If yes, then it sets DT and NC to zero and terminates, since there is no memory data transfer in this case [lines 3–6].

If the source and destination PMs differ, it sets DV_0 to the VM memory size which indicates that the whole memory will be transmitted during the first round as per the pre-copy migration technique [line 7].

In each migration round [lines 8–19], the model estimates the time duration of the round (T_i) by dividing the memory data to be transferred (DV_i) in this round (which is estimated in the previous round) by the available bandwidth (BA) of the migration network link [line 9]. It also estimates the size of the Writable Working Set (WWS) for the next round and deducts it from the memory size that is expected to be dirtied in this round in order to estimate the memory data that will be transmitted in the next round [lines 10–12]. The WWS is deducted since it represents the memory pages that are modified very frequently and are skipped during the pre-copy rounds. And, μ_1, μ_2, and μ_3 are model parameters that can be learned through benchmarking and learning techniques (e.g., linear regression). Table 6.2 shows the values of these parameters used for the purpose of performance evaluation. Details on this derivation and parameter values can be found in the original paper [78].

If the estimate of the memory data to be transferred in the next round goes below the predefined threshold (DV_{th}) or is greater than the memory data that is estimated to be transferred in this round [line 13], then it indicates that the termination condition is met and the next round would be the stop-and-copy phase of the migration process. For that round, it estimates the size of the memory data to be transferred, the duration of the stop-and-copy phase, and the VM downtime (DT) [lines 14–16]. Finally, the algorithm estimates the total memory data to be transferred (MD) and the migration duration (MT)

Algorithm 6.1 VMMigOverhead Algorithm

Input: v^{mem}, v^{dr}, BA, DS, and p.
Output: MD, MT, DT, and NC.
Initialization: $MD \leftarrow 0$; $MT \leftarrow 0$.

```
1: p_s ← v^hp
2: if p_s = p then {Check whether the source PM and destination PM are the same}
3:    DT ← 0
4:    NC ← 0
5:    return
6: end if
7: DV_0 ← v^mem {In the first iteration, the whole VM memory is transferred}
8: for i = 0 to max_round do
9:    T_i ← DV_i/BA(p_s, p) {Estimate the time duration for this pre-copy round}
10:   κ ← μ_1 × T_i + μ_2 × v^dr + μ_3
11:   W_{i+1} ← κ × T_i × v^dr {Estimate the size of WWS for the next round}
12:   DV_{i+1} ← T_i × v^dr − W_{i+1} {Estimate the migration data size for the next round}
13:   if DV_{i+1} ≤ DV_th ∨ DV_{i+1} > DV_i then {Check if termination condition is met}
14:      DV_{i+1} ← T_i × v^dr
15:      T_{i+1} ← DV_{i+1}/BA(p_s, p)
16:      DT ← T_{i+1} + T_res {Estimate the duration of VM downtime}
17:      break
18:   end if
19: end for
20: for i = 0 to max_round do
21:   MD ← MD + DV_i {Estimate the total memory data transfer}
22:   MT ← MT + T_i {Estimate the total migration time}
23: end for
24: NC ← MD × DS(p_s, p) {Estimate network cost for the migration}
```

by accumulating the memory data size and the time duration for each of the rounds, respectively [lines 20–23], as well as the network cost as a product of the total memory data and the network distance between the VM's current host PM (p_s) and the destination PM (p) [line 24].

Finally, the unified *Migration Overhead MO* for migrating a VM v from its current host (v^{hp}) to the destination PM p is modeled as a weighted summation of the estimates of the above-mentioned migration overhead factors computed by algorithm VMMigOverhead:

$$MO(v,p) = \alpha_1 \times MD(v,p) + \alpha_2 \times MT(v,p) + \alpha_3 \times DT(v,p) + \alpha_4 \times NC(v,p) \quad (6.6)$$

where α_1, α_2, α_3, and α_4 are input parameters that indicate the relative importance of the contributing migration overheads and $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in [0,1]$ such that $\sum_{i=0}^{4} \alpha_i = 1$. In order to keep the migration overhead within a fixed range of $[0,1]$ so that it is compatible

to be integrated into dynamic VM consolidation mechanisms, all the contributory factors *MD*, *MT*, *DT*, and *NC* are normalized against their maximum possible values before feeding them to compute the migration overhead *MO*.

Modeling Energy Consumption due to Migration

In a large data center with hundreds or thousands of running VMs, dynamic VM consolidation decisions can involve a large number of VM migrations. As a result, energy consumption due to the migration decisions should be taken into account and migration decisions that require a lower amount of energy consumption should be given preference over those that require higher energy. Since VM live migration is an I/O intensive task, the energy is primarily consumed due to the memory data transfer from the source to the destination. This data transfer involves the source PM, the destination PM, and the network switches. This work utilizes the energy consumption model presented by Liu *et al.* [78]. Here, the energy consumption by the switches is not taken into account due to the inherent complexity of the switching fabric. Moreover, since the data transmitted by the source and received by the destination are equal, it can be assumed that that the energy consumption by the two ends is the same. Therefore, the total energy consumption due to a single VM migration is shown to be linearly correlated with the amount of memory data transmitted from the source to the destination:

$$MEC(v,p) = \gamma_1 \times MD(v,p) + \gamma_2 \tag{6.7}$$

where γ_1 and γ_2 are model parameters. The specific values of these parameters used for the evaluation are shown in Table 6.2 and are taken as reported by Liu *et al.* [78]. When migration data (*MD*) is measured in Megabytes, the migration energy consumption (*MEC*) is estimated in Joules.

Modeling SLA Violation due to Migration

Since the applications hosted by a migrating VM experience performance degradation during the course of the migration operation, it is important to estimate the corresponding SLA violation and take this into consideration for any VM consolidation operation [12]. Obviously, VM consolidation decisions that result in fewer SLA violations compared to others are preferable in terms of migration overhead. An experimental study [124] on the impact of VM migration on an application demonstrates that performance degradation

depends on the application behavior, particularly on the number of memory pages modified by the application (i.e., page dirty rate). Furthermore, the study suggests that, for the class of web-applications, the average performance degradation can be estimated at around 10% of the CPU utilization during the migration operation. Therefore, the SLA violation due to a VM migration is modeled as follows:

$$MSV(v,p) = \sigma \times v^{cpu} \times MT(v,p). \tag{6.8}$$

where σ is an input parameter that indicates the percentage of performance degradation due to the migration.

Overall VM Migration Impact due to Dynamic VM Consolidation

For medium to large data centers, an offline, dynamic VM consolidation operation can require multiple VM migrations in order to achieve the desired consolidation. With the single migration overhead estimation models presented above, he estimates of the aggregated VM migration overhead factors are defined below:

1. Each VM migration data (MD) implies the amount of VM memory data that is needed to be transferred from the source PM to the destination PM and this data amount is directly proportional to the amount of energy needed for performing the migration operation. Therefore, the estimate of the aggregated migration data that will be transferred due to the VM migrations represented by a particular migration map MM is given by:

$$MD(MM) = \sum_{\langle v,p\rangle \in MM} MD(v,p). \tag{6.9}$$

2. Since the applications, which are hosted by a VM, experience performance degradation during the period of VM migration (MT) and therefore, the corresponding SLA violation is proportional to the migration time, the aggregated migration time for all the VM migrations represented by migration map MM is modeled as follows:

$$MT(MM) = \sum_{\langle v,p\rangle \in MM} MT(v,p). \tag{6.10}$$

3. For any VM performing live migration, the services provided by the hosted applications remain unavailable to the consumers during the total period of the VM's downtime. In order to reflect on the overall service outage of the migrating VMs, the aggregated VM downtime is measured as the accumulated downtime of all the migrating VMs given by migration map MM:

$$DT(MM) = \sum_{\langle v,p\rangle \in MM} DT(v,p). \tag{6.11}$$

4. The network cost (NC) that is incurred due to each VM migration implies the amount of additional network traffic and the corresponding energy consumption by the network switches due to the migration operation. Therefore, the network costs are considered additive and the aggregated network cost for a particular migration map MM is given by:

$$NC(MM) = \sum_{\langle v,p\rangle \in MM} NC(v,p). \tag{6.12}$$

Given the above estimation models for single VM migration overhead factors, the overall migration overhead for a particular dynamic VM consolidation plan (or MM) for a group or cluster of PMs is modeled as the accumulated overhead of all the necessary VM migrations within that group or cluster:

$$MO(MM) = \sum_{\langle v,p\rangle \in MM} MO(v,p). \tag{6.13}$$

Similarly, the estimate of the aggregated migration energy consumption for migration map MM is computed as the summation of the migration energy consumption of the individual VMs:

$$MEC(MM) = \sum_{\langle v,p\rangle \in MM} MEC(v,p). \tag{6.14}$$

And, the estimate of aggregated SLA violation for all the VM migrations given by a migration map MM is defined as follows:

$$MSV(MM) = \sum_{\langle v,p\rangle \in MM} MSV(v,p). \tag{6.15}$$

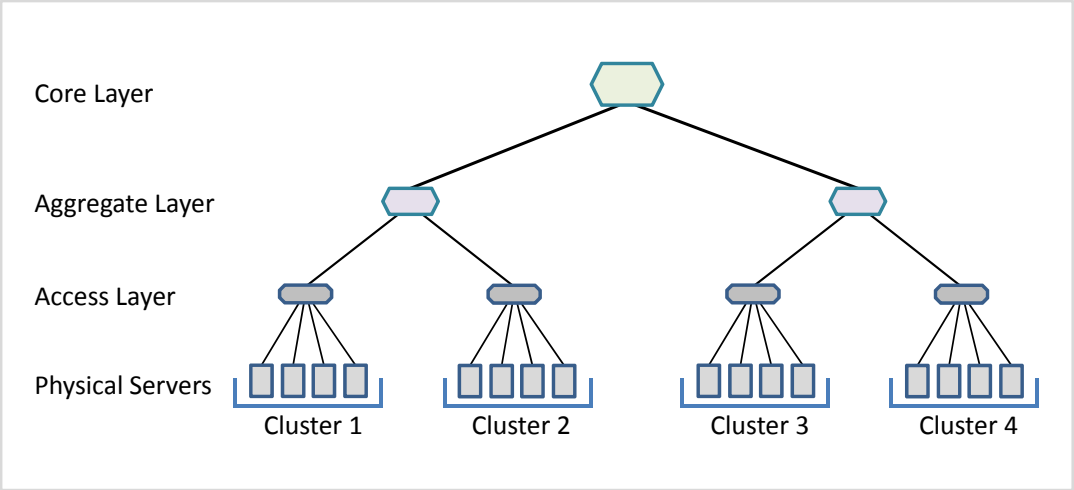

Figure 6.4: Clustering data center servers based on network proximity.

6.3.2 Hierarchical, Decentralized, Dynamic VM Consolidation Framework

This subsection presents a hierarchical, decentralized, dynamic VM consolidation framework. In order to achieve scalability, PMs in a data center are grouped into smaller clusters and the dynamic VM consolidation operation is performed separately within each cluster. With the goal of reducing the network overhead incurred due to the necessary VM migrations resulting from dynamic VM consolidation operation, PM clusters are formed based on the network cost of data communications among the PMs. The network cost can be derived through practical measures, such as the one presented in Section 5.2. In this proposed framework, the number of switches in the data communication path among the PMs is considered as a measure of the network distance and, based on this definition, PMs under the same access switch are grouped as an individual cluster (Figure 6.4). However, such hierarchical, decentralized framework and the VM consolidation algorithm are not restricted to this particular cluster formation approach and any other static or dynamic cluster formation techniques can be readily integrated with this framework. Cluster formation based on network proximity ensures that VMs are migrated to short distant target servers and this then limits the overall migration impact on applications and data center network. This is reflected in the network cost incurred due to the migration decision.

Figure 6.5 presents an overview of the hierarchical structure of the framework. Each PM in a cluster runs a Local Agent that collects VM related information, such as a list of hosted VMs (HV_p) and their resource usage (D_v). The Global Controller is the topmost

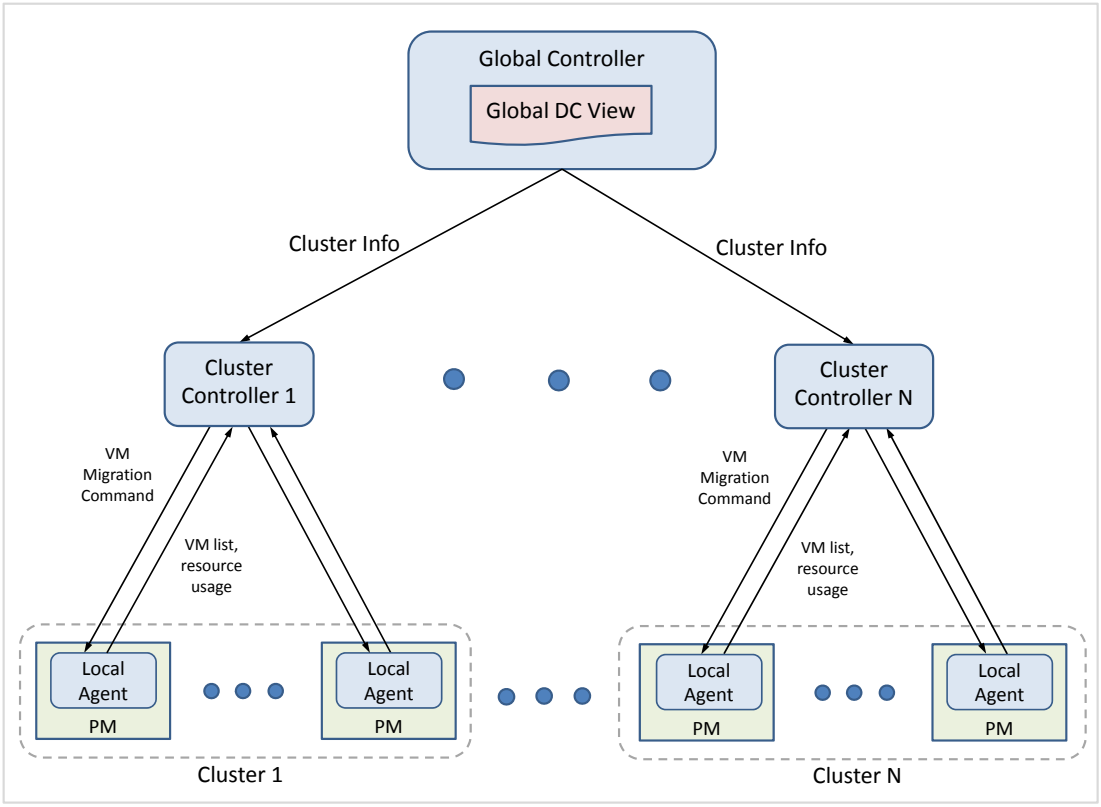

Figure 6.5: A Hierarchical, Decentralized Dynamic VM Consolidation Framework.

entity that has a global view of the data center, including the list of PM (PMS) and network information (DS and BA), and is responsible for cluster formation decisions. The Global Controller periodically sends cluster related information to each of the Cluster Controllers, such as a set/list of PMs in a cluster (P). Within each cluster, the Cluster Controller periodically receives information of the hosted VMs from each of the Local Agents and forms a cluster-wide list of VMs (V) hosted by the PMs. Within each cluster, the Cluster Controller periodically receives information on the hosted VMs from each of the Local Agents and forms a cluster-wide list of VMs (V) hosted by the PMs. When a triggering event occurs for the offline, dynamic VM consolidation operation (e.g., periodic or resource utilization threshold-based), each Cluster Controller runs the dynamic VM consolidation algorithm for its cluster and issues the necessary VM migration commands to the respective hypervisors. Global controller and cluster leader selection decision can be made either using static configuration or dynamic cluster leader selection algorithms [34, 79]. However, this aspect is beyond the scope of this chapter.

6.3.3 Migration Overhead-aware, Multi-objective Dynamic VM Consolidation Algorithm

This subsection presents the migration impact-aware, multi-objective dynamic VM consolidation algorithm (AMDVMC) based on the *Ant Colony System* (ACS) metaheuristic [36] that iteratively refines migration plans in order to maximize the OF f_3 (6.2). The ACO-based metaheuristic is chosen as a solution for the MDVCP problem because of its proven effectiveness in the field of combinatorial optimization and polynomial time complexity [40]. The main components of the proposed AMDVMC algorithm are shown in Figure 6.6. As input, it takes the PM cluster along with the hosted VMs, and the AMDVMC consolidation scheme makes use of the single and overall VM migration overhead models presented in Subsection 6.3.1, as well as the various resource and energy related models presented in Subsection 4.2.3 and Subsection 4.2.4. Within the scheme, the AMDVMC Controller creates multiple ant agents and delivers every ant an instance of the input PM cluster. The ants run in parallel, compute solutions (i.e., VM migration maps $MM = \{\langle v, p\rangle\}$), and pass the maps to the Controller. Each migration map consists of a list of VM-to-server migration commands ($\langle v, p\rangle$) for all the VMs in the cluster. For the migration commands where the source and the destination PMs are the same, all the migration factors and overhead for these VMs would be zero and would not contribute to the overall migration overhead. The AMDVMC Controller then detects the best migration map based on the OF f_3 (6.2), updates the shared pheromone information, and executes the ants once again for the next cycle. Finally, when the predefined stop condition is met, the controller outputs the so-far-found best migration map.

Adaptation of ACO Metaheuristic for AMDVMC

Following a similar adaption performed for the AVVMC algorithm proposed for solving the multi-objective consolidated VM cluster placement problem (MCVPP) presented in Subsection 4.4.2, each VM-to-PM migration within a cluster is considered as an individual solution component for adapting the ACS metaheuristic [36] in order to solve the multi-objective dynamic VM consolidation problem. However, from the perspective of the solution-building process of ACS metaheuristic, there are two fundamental differences between the MCVPP problem outlined in Chapter 4 and the MDVCP problem defined in this chapter:

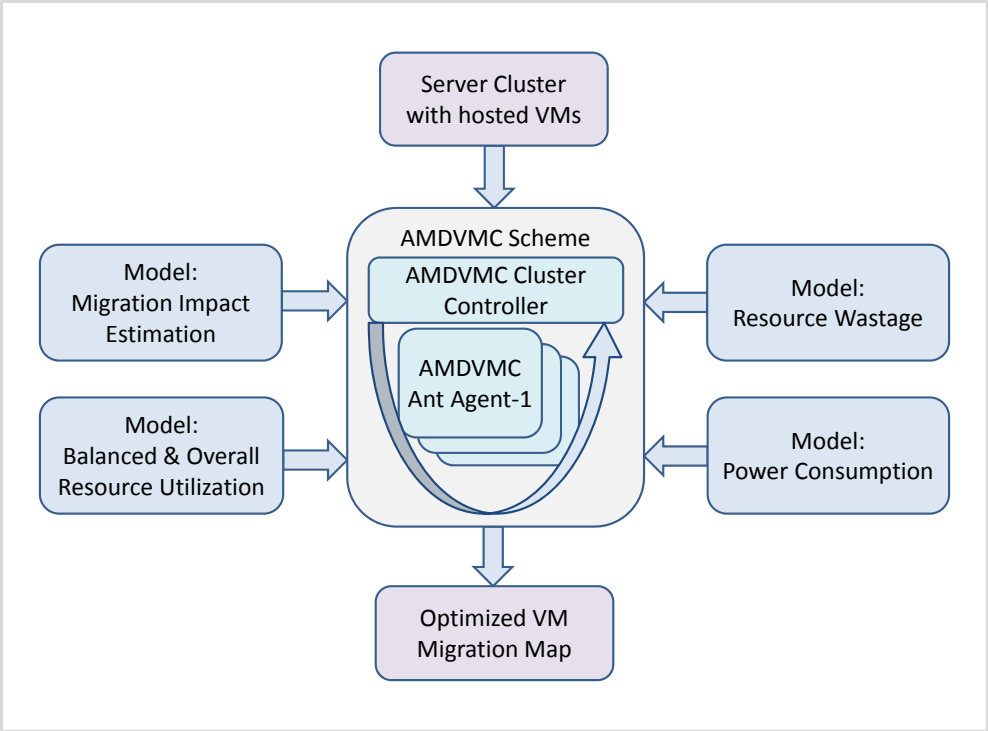

Figure 6.6: AMDVMC algorithm with associated models.

1. The initial data center states of MCVPP and MDVCP are not the same: in the case of MCVPP, VMs are initially unassigned and the PMs are considered empty whereas in the case of the MDVCP, VMs are already assigned to their host PMs and therefore the PMs are not empty.

2. In the case of MCVPP, each VM-to-PM assignment provides some benefit in terms of resource utilization but does not have any associated overhead or negative impact, whereas for the MDVCP, migrating a VM to a PM other than its current host provides some benefit in terms of resource utilization, but at the same time, incurs migration overhead.

Without any loss of generality in the ACS's solution-building process, the first difference is addressed by considering that the VMs are initially pulled out of the host PMs and kept in a virtual VM pool and, in this way, the PMs can be considered virtually empty. As for the second difference, both the OF f_3 (6.2) and the heuristic information (6.17) are updated in order to reflect the differences.

It is worth noting that by the use of such adaptation, as well as the resource capacity and migration-related constraints defined in Section 6.2.1, the proposed VM consolidation approach effectively prevents the occurrences of unfavorable situations, such as a VM migrates between two PMs continually or cascading VM migrations. The addressed VM consolidation problem is formulated using strong constraints so that the modeled scenario can be a reflection of VM consolidation operations in real world settings. VM migration constraints are represented by formulations 6.3 and 6.5 which express that any VM within the data center can be migrated to at most one destination server during a VM consolidation round. This constraint effectively prevents the occurrence of situations such as a particular VM is selected for multiple migrations. Furthermore, situations such as cascading VM migrations can never arise as well since the proposed AMDVMC consolidation algorithm ensures that any VM migration decision is taken only when the destination server has enough resource capacities to host the migrating VM. Cascading VM migrations can occur in cases when migration of a VM from its current host to a destination server requires other VM migration(s) from that destination server to third server(s) in order to make room for the initial VM migration. However, the proposed adaptation of the ACO metaheuristic for solving the dynamic VM consolidation problem effectively prevents any possibilities of such situations.

Pheromone values are associated to each VM-to-PM migration that denotes the desirability of migrating a VM to a target PM (6.16 & 6.22) and are implemented using an $N_v \times N_p$ pheromone matrix τ. During the solution-building process, heuristic values are computed dynamically for each VM-to-PM migration, which represents the preference of migrating a VM to a target PM in terms of both PM resource utilization and VM migration overhead (6.17). In an ACO cycle, each ant agent generates a solution (migration map) comprising of a list of VM-to-PM migrations. At the end of each cycle, the best solution is identified based on the OF f_3 (6.2) value and the pheromone levels of the solution components are updated so as to navigate the search space more effectively and shun early stagnation to a sub-optimal.

The AMDVMC Algorithm

The pseudocode of the proposed AMDVMC algorithm is shown in Algorithm 6.2. It starts with a list of PMs (P) in a cluster along with the set of hosted VMs (V) and the relevant parameters as input, and generates a migration map (MM) as output. At the

beginning of each cycle, each ant starts with an empty migration map, a set of empty PMs having total resource capacities similar to the PMs in P (generated by subroutine *EmptyPMSet*), and a set of VMs having total resource demands similar to the VMs in V (generated by subroutine *CopyVMSet*), and shuffles the VMs in *vmList* [lines 2–7]. Ants work with empty PMs and the VMs are considered to be not-yet-placed in order to facilitate consolidation of VMs with the goal of maximizing resource utilization and eventually, minimizing energy consumption by increasing the number of released PMs. Moreover, when assigning VMs to PMs, ants take into consideration where the VMs are currently hosted and the corresponding migration overhead is taken into account for making the migration decisions. And, shuffling the VMs of *vmList* adds randomization in the subsequent search process.

Within lines 11–21, all the ants generate their migration maps (solutions) using a modified ACS rule (6.19). In each while loop iteration, an ant is chosen randomly [line 12]. If the ant has at least one VM in its *vmList*, it chooses a VM-to-PM migration from all the feasible VM migration options for the VM in *vmList*, adds the $\langle VM,PM \rangle$ pair in its migration map (*MM*), and removes the VM from its *vmList* [lines 13–16]. Otherwise, the ant has finished making migration decisions for all the VMs and it computes the OF (f_3) value for its solution according to (6.2) and the ant is removed from *antList* [lines 17–20].

When all the ants have completed building their solutions, the while loop ends and the new *Global-best Migration Map* (*GBMM*) is identified by comparing the existing *GBMM* with the newly computed migration maps [lines 23–29]. Thereafter, the pheromone reinforcement amount is computed based on the quality of the *GBMM* [line 31] accordingly to (6.23) and the pheromone matrix is updated by simulating pheromone evaporation and deposition for each $\langle VM,PM \rangle$ pair accordingly to (6.22) [lines 32–36]. The algorithm reinforces the pheromone values only on the $\langle VM,PM \rangle$ pairs that belong to the *GBMM*.

Finally, the algorithm checks whether there has not been any improvement in the quality of the solution for the last $nCycleTerm$ cycles or a total of $nResetMax$ cycle resets have occurred [line 37]. If it finds improvement, the search process repeats; otherwise, the algorithm terminates with the current *GBMM* as output. The $nResetMax$ parameter is used to set an upper bound on the number of cycle resets so that AMDVMC does not

Algorithm 6.2 AMDVMC Algorithm

Input: Set of PMs P and set of VMs V in the cluster, set of ants $antSet$. Set of parameters $\{nAnts, nCycleTerm, nResetMax, \omega, \lambda, \beta, \delta, q_0, a, b\}$.
Output: Global-best migration map $GBMM$.
Initialization: Set parameter values, set pheromone value for each $\langle VM, PM\rangle$ pair ($\tau_{v,p}$) to τ_0 [(6.16)], $GBMM \leftarrow \emptyset, nCycle \leftarrow 0, nCycleReset \leftarrow 0$.

repeat
  for each $ant \in antSet$ **do** {Initialize data structures for each ant}
    $ant.mm \leftarrow \emptyset$
    $ant.pmList \leftarrow EmptyPMSet(P)$
    $ant.vmList \leftarrow CopyVMSet(V)$
    Shuffle $ant.vmList$ {Shuffle VMs to randomize search}
  end for

  $nCycle \leftarrow nCycle + 1$
  $antList \leftarrow antSet$
  while $antList \neq \emptyset$ **do**
    Pick an ant randomly from $antList$
    if $ant.vmList \neq \emptyset$ **then**
      Choose a $\langle v, p\rangle$ from set $\{\langle v, p\rangle | v \in ant.vmList, p \in ant.pmList\}$ according to (6.19)
      $ant.mm \leftarrow ant.mm \cup \langle v, p\rangle$ {Add the selected $\langle v, p\rangle$ to the ant's migration map}
      $ant.vmList.remove(v)$
    else{When all VMs are placed, then ant completes a solution and stops for this cycle}
      Compute the objective function (OF) value for $ant.mm.f_3$ according to (6.2)
      $antList.remove(ant)$
    end if
  end while

  for each $ant \in antSet$ **do** {Find global-best migration map for this cycle}
    if $ant.mm.f_3 > GBMM.f_3$ **then**
      $GBMM \leftarrow ant.mm$
      $nCycle \leftarrow 0$
      $nCycleReset \leftarrow nCycleReset + 1$
    end if
  end for

  Compute $\Delta\tau$ based on (6.23) {Compute pheromone reinforcement amount for this cycle}
  for each $p \in P$ **do** {Simulate pheromone evaporation and deposition for this cycle}
    for each $v \in V$ **do**
      $\tau_{v,p} \leftarrow (1 - \delta) \times \tau_{v,p} + \delta \times \Delta\tau_{v,p}$
    end for
  end for
until $nCycle = nCycleTerm$ **or** $nCycleReset = nResetMax$ {AMDVMC ends either if it sees no progress for consecutive $nCycleTerm$ cycles, or a total of $nResetMax$ cycle resets have taken place}

run indefinitely. The remainder of this section formally defines the various parts of the AMDVMC algorithm.

Definition of Pheromone and Initial Pheromone Amount: ACO algorithms [35] start with a fixed amount of pheromone value for each of the solution components. For each solution component (here each $\langle v, p\rangle$ migration pair), its pheromone level provides a

measure of desirability for choosing it during the solution-building process. In the context of AMDVMC, a fixed and uniform pheromone level for each of the solution components means that, at the beginning, each VM-to-PM migration has equal desirability. Following the approach used in the original ACS metaheuristic [36], the initial pheromone amount for AMDVMC is set to the quality of the migration map generated by the referenced L1 norm-based First Fit Decreasing (FFDL1) baseline algorithm:

$$\tau_0 \leftarrow f_{FFDL1}. \tag{6.16}$$

Definition of Heuristic Information: Heuristic value provides a measure of preference for selecting a solution component among all the feasible solution components during the solution-building process. For the AMDVMC algorithm, heuristic value $\eta_{v,p}$ indicates the apparent benefit of migrating a VM v to a PM p in terms of the improvement in the PM's resource utilization and the overhead incurred for migrating v to p. However, an increase in a PM's resource utilization provides a positive incentive for improving the quality of the overall migration decision, whereas the migration overhead works as a negative impact since it reduces the quality of the migration decision according to the OF f_3 (6.2). Therefore, the heuristic value $\eta_{v,p}$ for selecting $\langle v, p\rangle$ migration is measured as follows:

$$\eta_{v,p} = \lambda \times UG_p(v) + (1 - \lambda) \times (1 - MO(v, p)) \tag{6.17}$$

where $UG_p(v)$ is the *utilization gain* of PM p after placing VM v in it and is computed as follows:

$$UG_p(v) = \omega \times (-\log_{10}\|RIV_p(v)\|) + (1 - \omega) \times Utilization_p(v) \tag{6.18}$$

where $\|RIV_p(v)\|$ is the magnitude of the *Resource Imbalance Vector* (RIV) of the PM p after assigning the VM v to it (4.11), $Utilization_p(v)$ is the overall resource utilization of the PM p if the VM v is assigned to it (4.19), and $\omega \in [0, 1]$ is a parameter that trades off the relative importance of balanced versus overall resource utilization; and $MO(v, p)$ is the migration overhead incurred due to transferring the VM v to the PM p as expressed in 6.6. Finally, $\lambda \in [0, 1]$ is a parameter that sets the relative weight between the achieved utilization gain and migration overhead incurred as per the definition. In order to ensure metric compatibility for the heuristic formulation (6.17), both the utilization gain UG and migration overhead MO are normalized against their maximum values.

Pseudo-random Proportional Rule: During the migration map generation process (Algorithm 6.2, line 14), an ant k uses the following probabilistic decision rule [36] to select a VM v to be migrated to PM p:

$$s = \begin{cases} \arg\max_{v \in FM_k(vmList, pmList)} \{\tau_{v,p} \times [\eta_{v,p}]^{\beta}\}, & \text{if } q \leq q_0; \\ S, & \text{otherwise} \end{cases} \tag{6.19}$$

where $q \in [0,1]$ is a uniform random number, $q_0 \in [0,1]$ is an input parameter, $\eta_{v,p}$ is the heuristic value for $\langle v,p \rangle$ migration (6.17), $\tau_{v,p}$ is the current pheromone value of $\langle v,p \rangle$ pair (6.22), β is a non-negative parameter that trades off between the significance of the pheromone amount and the heuristic value in the decision rule, and S is a random variable selected according to the probability distribution given below by (6.21). $FM_k(vmList, pmList)$ defines the set of feasible migrations ($\langle v,p \rangle$) for ant k based on the VMs in *vmList* and PMs in *pmList* (i.e., VM migrations that do not violate the resource capacity constraint of target PM p given by 6.4):

$$FM_k(vmList, pmList) = \left\{ \langle v,p \rangle \middle| \forall l \in RCS, \forall v \in vmList, \forall p \in pmList : U_p^l + D_v^l \leq C_p^l \right\}. \tag{6.20}$$

The above-mentioned decision rule works as follows: when $q \leq q_0$, then the $\langle v,p \rangle$ pair that results in the largest $\tau_{v,p} \times [\eta_{v,p}]^{\beta}$ value is selected and added to the migration map (exploitation), otherwise a $\langle v,p \rangle$ pair is chosen with probability $P_k(v,p)$ using the following *random-proportional rule* (exploration):

$$P_k(v,p) = \begin{cases} \frac{\tau_{v,p} \times [\eta_{v,p}]^{\beta}}{\sum_{\langle u,p \rangle \in FM_k(vmList, pmList)} \tau_{u,p} \times [\eta_{u,p}]^{\beta}}, & \text{if } \langle u,p \rangle \in FM_k(vmList, pmList); \\ 0, & \text{otherwise.} \end{cases} \tag{6.21}$$

The above random-proportional rule uses the pheromone values ($\tau_{v,p}$) of each $\langle v,p \rangle$ pair multiplied by the corresponding heuristic value ($\eta_{v,p}$) so as to prefer $\langle v,p \rangle$ pairs that improve PM resource utilization (both balanced and overall) and incur lower migration overhead, as well as having larger pheromone values.

Global Pheromone Update: With the aim of favoring the VM-to-PM migrations that constitute the GBMM so that the ants can be better guided in the following iterations,

the pheromone level of each $\langle v, p \rangle$ pair is updated using the following rule:

$$\tau_{v,p} \leftarrow (1 - \delta) \times \tau_{v,p} + \delta \times \Delta\tau_{v,p} \tag{6.22}$$

where δ is the global pheromone decay parameter ($0 < \delta < 1$) and $\Delta\tau_{v,p}$ is the pheromone reinforcement applied to each $\langle v, p \rangle$ pair that make up the GBMM. The value of the reinforcement is measured based on the quality of the solution in terms of the OF (f_3) value:

$$\Delta\tau_{v,p} = \begin{cases} f_3(GBMM), & \text{if } \langle v, p \rangle \in GBMM; \\ 0, & \text{otherwise.} \end{cases} \tag{6.23}$$

6.4 Performance Evaluation

This section presents the performance evaluation of the proposed AMDVMC algorithm through simulation-based experimentation where the results are compared to both migration impact-unaware and migration impact-aware dynamic VM consolidation algorithms. Subsection 6.4.1 provides description of the algorithms compared, Subsection 6.4.2 explains the simulation setup, and finally, the results and their analyses are presented in the subsequent subsections.

6.4.1 Algorithms Compared

The following VM consolidation algorithms are implemented and compared in this performance evaluation:

First Fit Decreasing based on L1-norm (FFFL1)

The FFFL1 algorithm is used as the baseline algorithm for the performance evaluation. This algorithm does not take into account the current VM-to-PM placements and it is, therefore, a migration impact-unaware algorithm. Scalability is ensured by running FFDL1 separately for each PM cluster as presented in the previous subsection. For each cluster, VMs are considered to be pooled out of the PMs and sorted in decreasing order of their resource demands. The L1-norm mean estimator is utilized to represent the three different resources (CPU, memory, and network I/O) into a scalar form. Thereafter, FFDL1 places each VM from the sorted list in the first feasible PM in the cluster following the *First Fit* (FF) approach. The VM placements are subject to the resource capacity constraints

presented in (6.4) and (6.5). When the dynamic consolidation is performed for all the PM clusters, data center-wide performance metrics (resource, power, and migration overhead) are accumulated using the pre-defined formulations (4.13), (4.15), and (6.9–6.15).

Time Complexity: For the above-mentioned implementation, the worst-case time complexity for the FFDL1 algorithm is given by:

$$T_{FFDL1} = \mathcal{O}(N_{vc}\lg N_{vc}) + \mathcal{O}(N_{vc}N_{pc}). \tag{6.24}$$

For the cases of average Cloud data centers, N_{pc} is expected to be greater than $\lg N_{vc}$. With this assumption, the above equation can be reduced to the following:

$$T_{FFDL1} = \mathcal{O}(N_{vc}N_{pc}). \tag{6.25}$$

Memory Overhead: Given that merge sort [28] is used in FFDL1 implementation, then the memory overhead for sorting the VMs in a cluster would be $\mathcal{O}(N_{vc})$. Apart from sorting, the placement decision part of FFDL1 works in-place without using any additional data structure. Therefore, the overall memory overhead of the FFDL1 algorithm is given by:

$$M_{FFDL1} = \mathcal{O}(N_{vc}). \tag{6.26}$$

Max-Min ant system-based Dynamic VM Consolidation (MMDVMC)

The MMDVMC algorithm [45] is an offline, dynamic VM consolidation algorithm that is executed in a random neighborhood of PMs based on an unstructured Peer-to-Peer network [125]. It aims to increase the number of released PMs and the variance of the scalar valued PM used capacity vectors, and reduce the number of necessary VM migrations within each neighborhood of the data center. It utilizes the Max-Min Ant System (MMAS) [111] to solve the dynamic consolidation problem where multiple ant agents iteratively refine migration plans. This algorithm runs for $nCycles$ cycles and in each cycle a total of $nAnts$ ant agents compute solutions. In each cycle, every ant selects the VM migrations that eventually maximize the defined objective function value. At the end of each iteration, the cycle-best migration plan is determined and compared against the existing global-best plan in order to identify the new global-best migration plan. Finally, pheromone values are updated for each VM-PM pair using a pheromone update rule that bounds that pheromone values with a pre-defined range of $[\tau_{max}, \tau_{min}]$. The algorithm runs for a pre-defined number of iterations and returns that final global-best

migration plan. The relevant parameter values for the algorithm are taken as reported in the original paper [45]. The MMDVMC algorithm takes into account the number of VM migrations as a measure of migration overhead or impact, which is analyzed as an oversimplified measure in Section 6.3.1. Similar to FFDL1, data center-wide performance metrics (resource, power, and migration overhead) are accumulated over all the clusters using the pre-defined formulations (4.13), (4.15), and (6.9–6.15).

Time Complexity: From the algorithmic pseudocode presented in the original paper [45], the worst-case time complexity of MMDVMC algorithm can be given by:

$$\begin{aligned} T_{MMDVMC} &= \mathcal{O}(nCycles.nAnts.N_{vc}.N_{pc}.N_{vc}) \\ &= \mathcal{O}(nCycles.nAnts.N_{vc}^2.N_{pc}). \end{aligned} \tag{6.27}$$

Furthermore, considering the ACO parameters as constants, the worst-case time complexity can be simplified to the following:

$$T_{MMDVMC} = \mathcal{O}(N_{vc}^2.N_{pc}). \tag{6.28}$$

Memory Overhead: Since MMDVMC has used the MMAS metaheuristic, it has a memory overhead of $\mathcal{O}(N_{vc}N_{pc})$ for maintaining pheromone information. Moreover, it has another $\mathcal{O}(nAnts)$ memory overhead for managing $nAnts$ ant agents. Furthermore, in every iteration, each ant agent computes its own migration plan, using its local list of PMs for a cluster with their associated hosted VMs, and modifies the VM-to-PM assignments. As a consequence, each ant agent has another $\mathcal{O}(N_{vc}N_{pc})$ memory overhead due to the local information of a cluster. Therefore, the overall memory overhead of MMDVMC is given by:

$$\begin{aligned} M_{MMDVMC} &= \mathcal{O}(N_{vc}.N_{pc}) + \mathcal{O}(nAnts.N_{vc}.N_{pc}) \\ &= \mathcal{O}(nAnts.N_{vc}.N_{pc}). \end{aligned} \tag{6.29}$$

Considering the number of ants is fixed, the memory overhead is simplified as follows:

$$M_{MMDVMC} = \mathcal{O}(N_{vc}.N_{pc}). \tag{6.30}$$

ACO-based Migration impact-aware Dynamic VM Consolidation (AMDVMC)

The proposed AMDVMC algorithm is implemented based on the description presented in Section 6.3.3 and follows the execution flow presented in Algorithm 6.2. The PMs in the data center are grouped into PM clusters, as presented in Section 6.3.2, and dynamic VM consolidation is performed by executing the AMDVMC algorithm in each cluster

separately. Finally, data center-wide performance metrics (resource, power, and migration overhead) are accumulated over all the clusters using the pre-defined formulations (4.13), (4.15), and (6.9–6.15).

Time Complexity: The time complexity for initializing the ant-related data structures (lines 2–7) is $T_{2\text{–}7} = \mathcal{O}(nAnts.N_{vc})$. The complexity of the migration map generation process for each of the ants is $\mathcal{O}(N_{vc}.N_{pc})$. Therefore, the time complexity of the migration maps generation for $nAnt$ (lines 11–21) would be $T_{11\text{–}21} = \mathcal{O}(nAnts.N_{vc}.N_{pc})$. Thereafter, the new GBMM identification part [lines 23–29] requires $T_{23\text{–}29} = \mathcal{O}(nAnts)$ time. Finally, the pheromone update part [lines 31–36] has $T_{31\text{–}36} = \mathcal{O}(N_{vc}.N_{pc})$ time complexity. Therefore, the overall time complexity for a single ACO iteration can be given by:

$$\begin{aligned} T_{AMDVMC_1} &= T_{2\text{–}7} + T_{11\text{–}21} + T_{23\text{–}29} + T_{31\text{–}36} \\ &= \mathcal{O}(nAnts.N_{vc}) + \mathcal{O}(nAnts.N_{vc}.N_{pc}) + \mathcal{O}(nAnts) + \mathcal{O}(N_{vc}.N_{pc}) \\ &= \mathcal{O}(nAnts.N_{vc}.N_{pc}). \end{aligned} \tag{6.31}$$

And, finally the repeat-until loop can run for a maximum of $\mathcal{O}(nCycleTerm.nResetMax)$ times. Therefore, the worst-case time complexity of AMDVMC algorithm can be given by:

$$T_{AMDVMC} = \mathcal{O}(nCycleTerm.nResetMax.nAnts.N_{vc}.N_{pc}). \tag{6.32}$$

Furthermore, considering the ACO parameters as constants, the worst-case time complexity can be simplified to the following:

$$T_{AMDVMC} = \mathcal{O}(N_{vc}.N_{pc}). \tag{6.33}$$

Memory Overhead: Similar to MMDVMC, AMDVMC has an $\mathcal{O}(N_{vc}N_{pc})$ memory overhead for maintaining the pheromone information that represents all possible VM-to-PM migration preferences in the cluster and another $\mathcal{O}(nAnts)$ memory overhead for managing $nAnts$ ant agents. In addition, in every iteration, each ant agent generates its migration map using its local list of PMs in the cluster with their associated hosted VMs and updates the VM-to-PM placements. As a result, each ant agent has an additional $\mathcal{O}(N_{vc}N_{pc})$ memory overhead for managing local information of the cluster during each iteration. Therefore, the overall memory overhead of AMDVMC algorithm is the following:

$$M_{AMDVMC} = \mathcal{O}(nAnts.N_{vc}.N_{pc}). \tag{6.34}$$

Considering the number of ants is fixed, the memory overhead is simplified as follows:

$$M_{AMDVMC} = \mathcal{O}(N_{vc}.N_{pc}). \tag{6.35}$$

6.4.2 Simulation Setup

Data Center Setup

The simulated data center consists of N_p homogeneous PMs with three-dimensional resource capacities: CPU, memory, and network I/O. The total resource capacities of the PMs are set as 5.0 GHz, 10 GB, and 1 Gbps. Absolute values of PM resource capacities are simulated so that the migration overhead factors can be measured using the proposed migration overhead estimation model. The power consumption for an active PM is calculated according to the power consumption model presented in (4.14) and the values for E_{idle} and E_{full} in the model are set to 162 watts and 215 watts, respectively, as used by Gao *et al.* [50].

Given the fact that three-tier tree network topology with core-aggregation-access switch levels are predominantly used in production data centers [71], PMs in the simulated data center are interconnected with each other using three-tier tree network topology, as shown in Figure 6.4, where each of the network switches have 8 ports. The maximum bandwidth capacity of the inter-PM communication links used for the VM migrations is set to 1 Gbps, and the available bandwidths of such links at run-time are synthetically generated using random numbers from the normal distribution with a mean ($MeanBW$) of 0.05 and a standard deviation ($SDBW$) of 0.2. The network distance between any two PMs is measured as $DS = h \times DF$, where h is the number of physical hops (specifically, network switches) between any two PMs in the simulated data center architecture as defined above, and DF is the *Distance Factor* that implies the physical inter-hop distance. The value of h is computed using the analytical expression for tree topology as presented in [85], and DF is fed as a parameter to the simulation which is set to 2 for the performed experiments. The network distance of a PM with itself is 0 which implies that data communication is done using memory copy without going through the network. A higher value of DF indicates greater relative communication distance between any two data center nodes.

Before the dynamic consolidation operation, N_v number of VMs are considered to be running and are distributed randomly among the N_p PMs in a load balanced mode.

Such an initial data center state is simulated in order to provide an incentive for the offline consolidation operation so that there is scope for the algorithms to improve resource utilization and reduce energy consumption. The VM resource demands for CPU, memory, and network I/O are also synthetically generated using random numbers from the normal distribution with variable mean $MeanRsc$ and standard deviation $SDRsc$. The corresponding VM page dirty rates (v^{dr}) are generated using uniform random numbers from the range $[0, PR * v^{mem}]$, where PR is the ratio of maximum possible page dirty rate and VM memory size, and it is set to 0.25 for the whole simulation. Thus, the VM page dirty rate (v^{dr}) is effectively parameterized in the simulation by the VM memory demand (v^{mem}) and this eliminates the need for another parameter.

Table 6.2 summarizes the values of the parameters used in various formulations for the VM migration overhead estimation presented in Section 6.3.1. Specifically, the values of the remaining dirty memory threshold (DV_{th}), the maximum number of pre-copy migration rounds (max_round), the coefficients for WWS computation (μ_1,μ_2, and μ_3), and the VM resume time (T_{res}) are taken as reported from the original paper [78] and are used in Algorithm 6.1. The coefficients of VM migration energy consumption (γ_1 and γ_2) are used in (6.7) and their values are taken as reported in by Liu *et al.* [78]. The coefficients for computing the overall VM migration overhead MO (α_1, α_2, α_3, and α_4) are used in (6.6) and each of them is set to 0.25 in order to give each of the overhead factors equal weight. The overhead factors are normalized against their maximum possible values before using in formulation (6.6), where the maximum values are identified by conducting preliminary experiments. The percentage of SLA violation (σ) during the migration duration is used in (6.8) and set to 0.1 as reported in a previous experimental study [124].

Finally, Table 6.3 shows the optimal values for the input parameters used for the proposed ACO-based, Migration overhead-aware Dynamic VM Consolidation (AMDVMC) algorithm, including those for the ACO metaheuristic. These values are determined by rigorous parameter sensitivity analysis conducted during the preliminary phase of the experiment, similar to those presented in Subsection 4.5.2. Input parameters for other consolidation algorithms are taken as reported in the respective papers. All the experiments presented in this paper have been repeated 1000 times and the average results are reported.

Table 6.2: VM migration-related parameters used in the simulation

Constants/Values	*Meaning*
$DV_{th} = 200(MB)$	Remaining dirty memory threshold
$max\_round = 20$	Maximum number of rounds of pre-copy migration
$\mu_1 = -0.0463$ $\mu_2 = -0.0001$ $\mu_3 = 0.3586$	Coefficients for computing Writable Working Set
$T_{res} = 20(ms)$	Time needed to resume a VM in the destination PM
$\alpha_1 = 0.25$ $\alpha_2 = 0.25$ $\alpha_3 = 0.25$ $\alpha_4 = 0.25$	Coefficients for computing overall VM migration overhead
$\gamma_1 = 0.512$ $\gamma_2 = 20.165$	Coefficients of VM migration energy computation
$\sigma = 0.1$	Percentage of SLA violation during migration

Table 6.3: ACS parameter values used for the AMDVMC algorithm in evaluation

$nAnts$	$nCycleTerm$	$nCycleMax$	β	δ	q_0	ω	λ	ϕ
5	5	100	1	0.3	0.8	0.5	0.05	1

Performance Evaluation Metrics

The quality of the consolidation decisions produced by the algorithms are compared across several performance metrics. Each dynamic VM consolidation has two types of performance factors: gain factors and cost factors.

The gain factors indicate the benefit or profit that can be achieved by a particular consolidation. The first gain factor reported in the results is the number of released PMs in the data center ($nReleasedPM$). The consolidation decision that releases the maximum number of PMs effectively consolidates the running VMs in the minimum number of active PMs. The PMs released in this process can be turned to lower power states to reduce power consumption in the data center, or can be utilized to accommodate further VM requests, which effectively improves the capacity of the data center, and eventually, maximizes profit. Another closely related performance metric shown in the results is the packing efficiency (PE) given by Equation (4.17). The PE indicates the average number of VMs packed or consolidated in each of the active PMs. Therefore, as the term and the formulation suggest, it effectively captures each algorithm's efficiency in packing or consolidating the running VMs in the active PMs.

The overall power consumption (measured in KW) of the active PMs according to formulations (4.14) and (4.15) is reported as the third gain factor in this evaluation. This

is one of the most important performance indicators for any consolidation scheme, which is directly proportional to the operating cost of hosting the running VMs in the data center, since reduction of the power consumption is equivalent to saving on the electricity costs of data center operation. The last factor reported in this category is the overall resource wastage (normalized against the total resource capacity of PM) of the active PMs after the VM consolidation. This factor is measured as the accumulated resource wastage of the PMs that are active after the consolidation where the individual PM's resource wastage (normalized) is measured according to formulation (4.13). Reduction in resource wastage indicates efficiency in resource utilization in the data center, and thus the consolidation that causes the least amount of resource wastage is to be preferred over others.

The cost factors reported in this evaluation are the migration overhead factors and associated metrics for achieving a particular dynamic VM consolidation in the data center. As described in earlier sections, dynamic VM consolidation achieved by VM live migrations has adverse effects on the hosted applications and on the data center itself. The measures of the cost factors incurred due to a particular consolidation decision, represented by migration map MM, are captured primarily by the four aggregated migration overhead factors presented in Section 6.3.1: the estimate of aggregated migration data transmitted (memory) across the data center due to the VM migrations $MD(MM)$ (6.9) in terabytes (TB), the aggregated migration time $MT(MM)$ (6.10) and the aggregated VM downtime $DT(MM)$ (6.11), both in the number of hours, and the aggregated network cost $NC(MM)$ (6.12). Obviously, for all of these cost factors, the VM consolidation decision that results in the lowest overhead factors will be preferable over others. Moreover, the overall migration overhead $MO(MM)$ (6.13) is also reported as a unified metric that captures the overall migration impact of a consolidation. Furthermore, an estimate of the aggregated migration energy consumption $MEC(MM)$ (in Kilo Joules) by the data center components (6.14) and an estimate of the aggregated SLA violation $MSV(MM)$ (6.15) of hosted applications due to VM migrations are reported.

All the above performance metrics are measured against the following scaling factors: (1) DC size (N_p), (2) mean resource demands of VMs ($MeanRsc$), and (3) diversification of workloads ($SDRsc$). The following subsections present the experimental results and analysis for each of the experiments conducted.

Simulation Environment

The algorithms are implemented in Java (JDK and JRE version 1.7.0) and the simulation is conducted on a Dell Workstation (Intel Core i5-2400 3.10 GHz CPU (4 cores), 4 GB of RAM, and 240 GB storage) hosting Windows 7 Professional Edition.

6.4.3 Scaling Data Center Size

This part of the experiment demonstrates the quality of the VM consolidation decisions produced by the algorithms with increasing problem size— the number of PMs (N_p) in the data center is set to 64 and increased to 4096 in stages, in each step by doubling the previous value. The number of VMs running in the data center is derived from the simulated number of PMs: $N_v = 2 * N_p$. As for the other parameters, $MeanRsc$ and $SDRsc$ are set to 0.05 and 0.2, respectively.

Figure 6.7 shows the four gain factors mentioned above that resulted from the VM consolidation decisions produced by the algorithms for different data center sizes. The average number of PMs released by the algorithms for each N_p value is plotted in Figure 6.7(a). As the figure demonstrates, on average, FFDL1, MMDVMC, and AMDVMC algorithms release 42%, 23%, and 36% of the PMs, respectively, for different data center sizes. FFDL1, being migration-unaware, consolidates the VMs without any regard to the current VM-to-PM placements and, therefore, releases the maximum number of PMs. MMDVMC, on the other hand, releases the least number of PMs, given that it tries to keep the number of VM migrations minimal at the same time. Finally, the performance of the proposed AMDVMC algorithm lies between the other algorithms by releasing 15% fewer PMs compared to FFDL1 and 63% more PMs than MMDVMC. This is also reflected in Figure 6.7(b) where it can be observed that AMDVMC achieves an average PE of 3.1, whereas FFDL1 and MMDVMC achieve PEs of 3.5 and 2.6, respectively.

Similar performance patterns are demonstrated in Figure 6.7(c) and Figure 6.7(d) which show the average power consumption (4.15) and the normalized resource wastage (4.13) of the active PMs after consolidation, respectively. It can be seen from the figures that both the power consumption and the resource wastage increase at the same rates as the number of PMs (N_p) are increased in the data center. Furthermore, compared to MMDVMC, the consolidation decisions produced by AMDVMC result in, on average, 13% less power consumption and 42% less resource wastage, respectively, whereas compared to

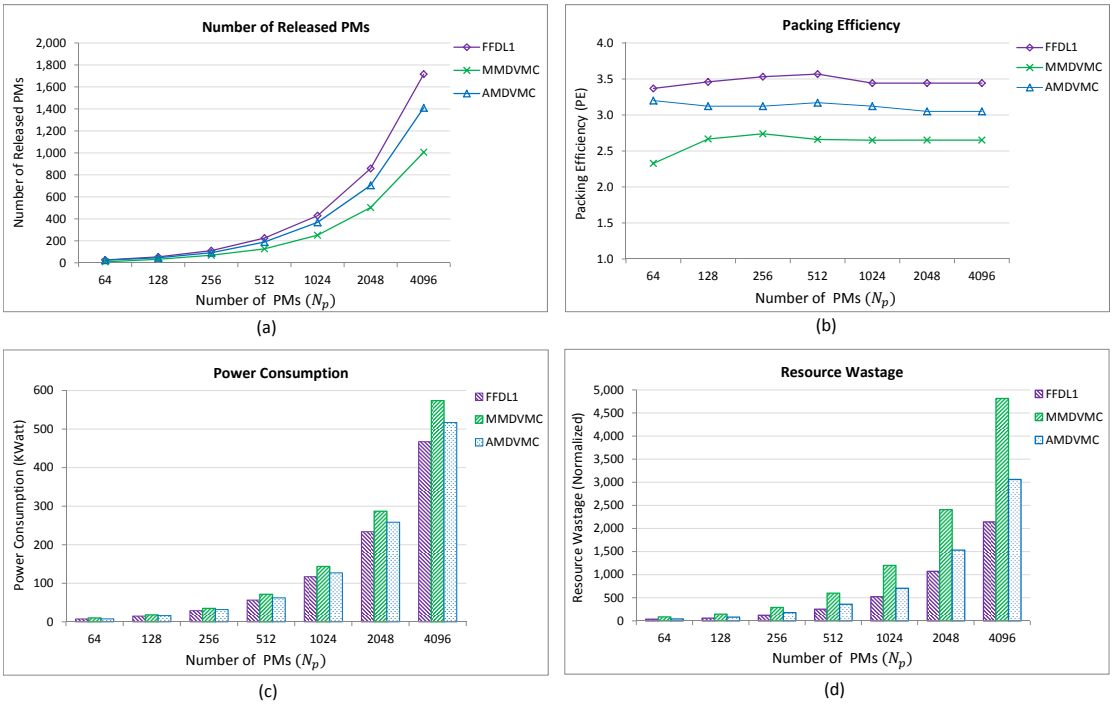

Figure 6.7: Performance of the algorithms with increasing N_p: (a) Number of Released PMs, (b) Packing Efficiency, (c) Power Consumption, and (d) Resource Wastage (best viewed in color).

FFDL1, AMDVMC incurs 9% more average power consumption and 38% more resource wastage. Therefore, it is evident from these results that AMDVMC performs better in terms of power consumption and resource wastage compared to the other migration-aware approach, whereas the migration-unaware approach beats AMDVMC in these metrics.

Figure 6.8 shows the four primary cost factors of dynamic consolidation decisions produced by the algorithms for various data center sizes. The estimate of the aggregated amount of VM memory data to be transmitted across the data center due to VM migrations is plotted in Figure 6.8(a). As the figure depicts, the data transmission rises sharply for FFDL1 with the increasing number of PMs. This is due to the fact that FFDL1 is migration-unaware and therefore, causes many VM migrations that result in a large amount of VM memory data transmission. MMDVMC, being multi-objective, tries to reduce the number of migrations and therefore, causes a lower amount of migration related data transmission. Lastly, AMDVMC is also a multi-objective consolidation approach which takes the estimate of memory data transfer into account during the solution-building process and as a consequence, it incurs the least amount of data transmission relating to VM migrations. Figure 6.8(b) shows the percentage of improvement of AMDVMC compared to FFDL1 and MMDVMC for this performance metric. In summary, on average,

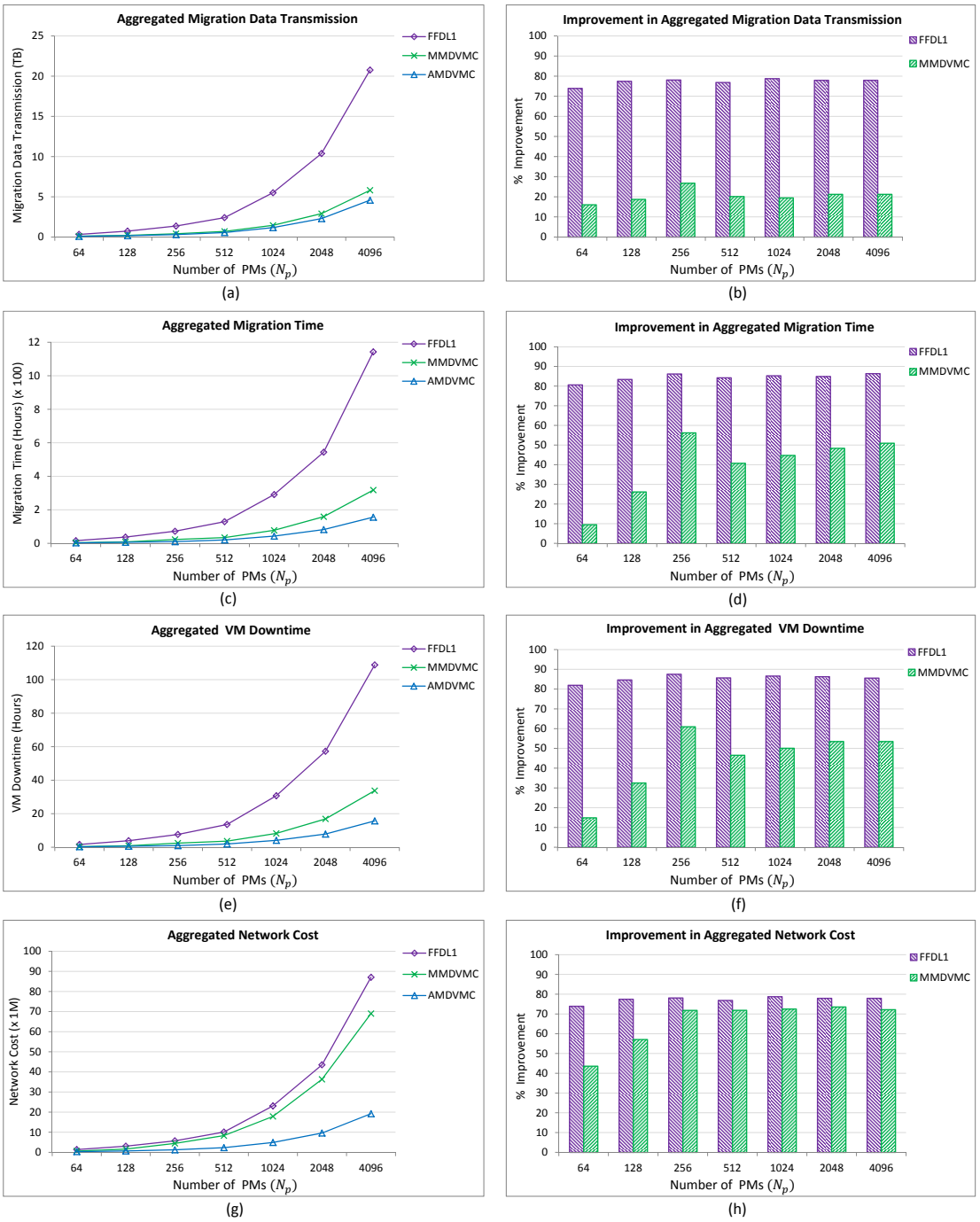

Figure 6.8: Performance of the algorithms with increasing N_p: (a) Aggregated Migration Data Transmission, (b) Improvement of AMDVMC over other algorithms in Aggregated Migration Data Transmission, (c) Aggregated Migration Time, (d) Improvement of AMDVMC over other algorithms in Aggregated Migration Time,(e) Aggregated VM Downtime, (f) Improvement of AMDVMC over other algorithms in Aggregated VM Downtime,(g) Aggregated Network Cost, (h) Improvement of AMDVMC over other algorithms in Aggregated Network Cost (best viewed in color).

AMDVMC resulted in 77% and 20% less migration data transmission compared to FFDL1 and MMDVMC, respectively.

For aggregated migration time (6.10) and total VM downtime (6.11), a similar performance pattern can be found from Figure 6.8(c) and Figure 6.8(e), respectively, where both the values increase at a proportional rate with the increase of N_p. This is reasonable since the number of VMs (N_v) increases in proportion to the number of PMs (N_p), which in turn contributes to the proportional rise of aggregated migration time and VM downtime. Figure 6.8(d) and Figure 6.8(f) show the relative improvement of AMDVMC over FFDL1 and MMDVMC for these performance metrics: on average, AMDVMC caused 84% and 85% less aggregated migration time, and 85% and 43% less aggregated VM downtime across all data center sizes, respectively.

Figure 6.8(g) shows the estimate of aggregated network cost (6.12) due the VM migrations for the consolidation decisions. The figure shows that for both FFDL1 and MMDVMC, the network cost increases sharply with the number of PMs in the data centers, whereas it increases slowly for AMDVMC. This is due to the fact that FFDL1 is migration overhead-unaware and MMDVMC, although it is in a way migration-aware, forms neighborhoods of PMs randomly for performing consolidation operations and therefore, does not take any network cost into account while making migration decisions. The relative improvement of AMDVMC over FFDL1 and MMDVMC is shown in Figure 6.8(h) and on average, the improvements are 77% and 65%, respectively.

Figure 6.9(a) presents a summary of the overall migration overheads incurred by the algorithms as per formulation (6.13) where on average, AMDVMC incurs 81% and 38% less migration overhead compared to FFDL1 and MMDVMC, respectively. Furthermore, the estimate of aggregated migration energy consumption (6.14) and SLA violation (6.15) are shown in Figure 6.9(b) and Figure 6.9(d), respectively. Since such energy consumption and SLA violation depend on the migration-related data transmission and migration time, respectively, these figures have similar performance patterns as those of Figure 6.7(a) and Figure 6.7(c), respectively. Finally, Figure 6.9(c) and Figure 6.9(e) present the relative improvement of AMDVMC over other algorithms where in summary, compared to FFDL1 and MMDVMC, on average AMDVMC reduces the migration energy consumption by 77% and 20%, and SLA violation by 85% and 52%, respectively.

From the results and discussions presented above, it can be concluded that, for all three compared algorithms, both the gain factors and the cost factors increase at a proportional

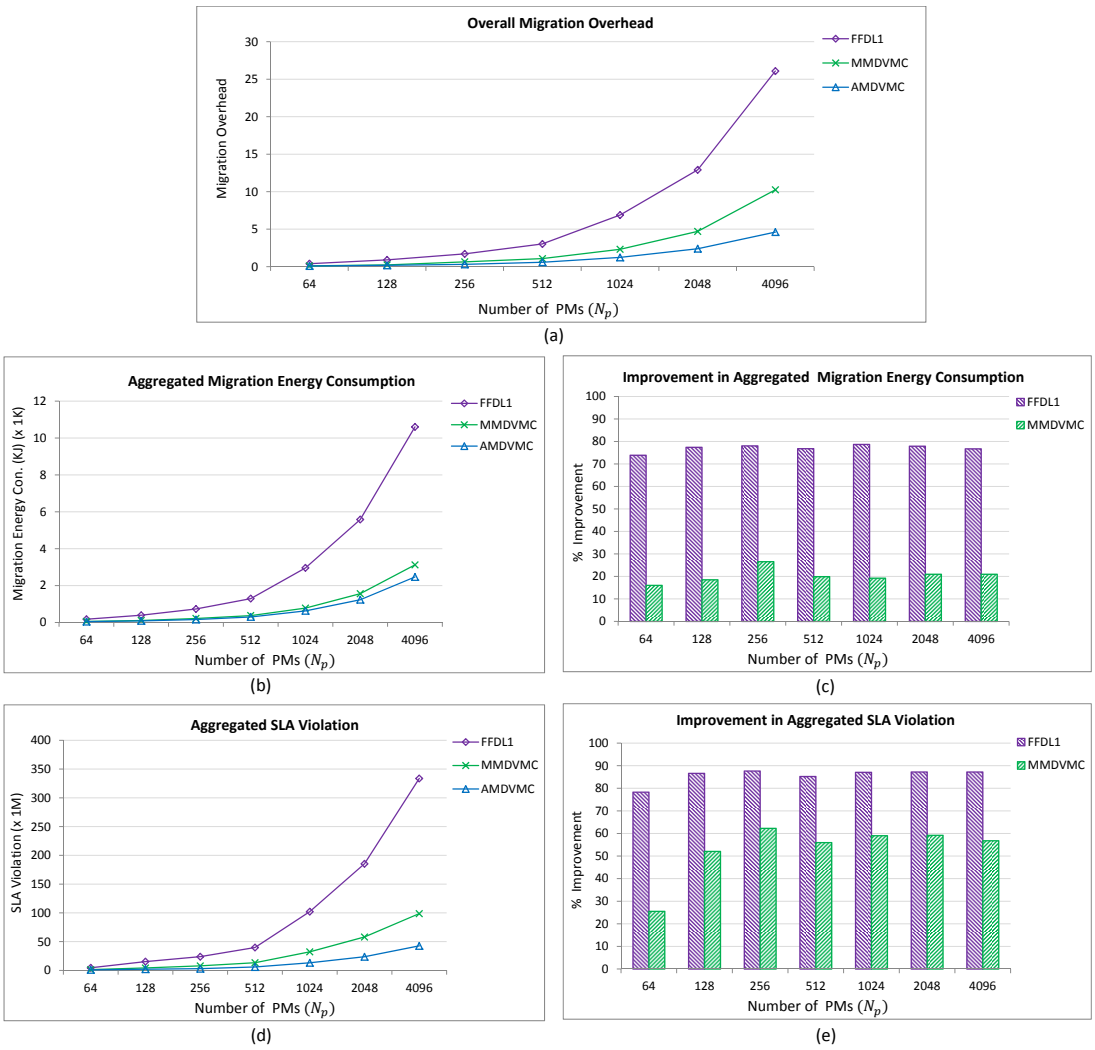

Figure 6.9: Performance of the algorithms with increasing N_p: (a) Overall Migration Overhead, (b) Aggregated Migration Energy Consumption, (c) Improvement of AMDVMC over other algorithms in Aggregated Migration Energy Consumption, (d) Aggregated SLA Violation, (e) Improvement of AMDVMC over other algorithms in Aggregated SLA Violation (best viewed in color).

rate with the size of the data center (N_p). In comparison to the migration-aware MMDMVC approach, the proposed AMDVMC scheme outperforms MMDVMC on both gain factors and cost factors by generating more efficient VM consolidation plans that result in reduced power consumption, resource wastage, and migration overhead. On the other hand, FFDL1, being migration-unaware, generates VM consolidation plans that result in lower power consumption and resource wastage compared to AMDVMC; however, this is achieved at the cost of much higher migration overhead factors.

6.4.4 Scaling Mean Resource Demand

In order to compare the quality of the solutions produced by the algorithms for various sizes of the active VMs, this part of the experiment starts with a mean VM resource

Table 6.4: Number of VMs (N_v) for corresponding *MeanRsc* values

MeanRsc	N_p	N_v
0.05	1024	2048
0.10	1024	1843
0.15	1024	1638
0.20	1024	1434
0.25	1024	1229
0.30	1024	1024

demand (*MeanRsc*) of 0.05 and increases it up to 0.3, raising it each time by 0.05. The maximum value for *MeanRsc* is kept at 0.3 in order to ensure that the VMs are not too large compared to the PM so that there will be little scope for performing consolidation operations. Moreover, multi-dimensionality of resource types reduces the scope of VM consolidation. Otherwise, if on average, one VM can be assigned per PM, there is no way of consolidating VMs and releasing PMs to improve power and resource efficiency. The number of PMs (N_p) in the simulated data center is set at 1024 and the number of simulated active VMs (N_v) in the data center is derived from the number of PMs using the following formulation:

$$N_v = N_p * (0.55 - MeanRsc)/0.25. \tag{6.36}$$

Table 6.4 shows the different values for N_v produced by the above equation for each *MeanRsc* value. This approach ensures that for the initial states, on average, each PM hosts two VMs when $MeanRsc = 0.05$ and with a gradual increase of *MeanRsc*, the average number of VMs hosted by each PM is reduced up to a point where, when $MeanRsc = 0.30$, each PM hosts one VM. Such an approach creates initial states such that there is scope for VM consolidation so that the efficiency of VM consolidation algorithms can be compared. The standard deviation of VM resource demand *SDRsc* is set to 0.2.

The four gain factors for each of the algorithms for the various means of VM resource demand are plotted in Figure 6.10. It can be observed from Figure 6.10(a) that the number of PMs released by the algorithms gradually increases as the *MeanRsc* increases. This is due to the fact that the number of VMs in the data center decreases with the increase of *MeanRsc* and as a result, more PMs are released by the algorithms even though the VM size increases. On average, FFDL1, MMDVMC, and AMDVMC have released 45%, 26%, and 38% of PMs in the data center, respectively, across different values of

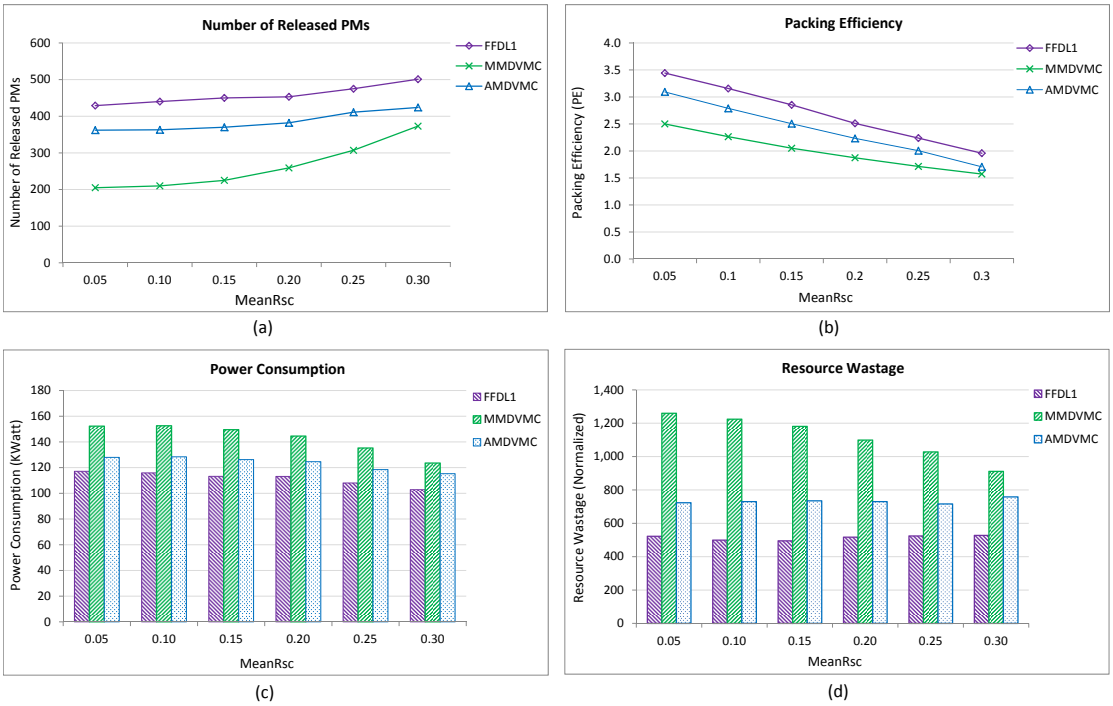

Figure 6.10: Performance of the algorithms with increasing *MeanRsc*: (a) Number of Released PMs, (b) Packing Efficiency, (c) Power Consumption, and (d) Resource Wastage (best viewed in color).

MeanRsc. In contrast to Figure 6.10(a), the packing efficiency PE for all the algorithms decreases consistently with the increase of *MeanRsc* (Figure 6.10(b)). This makes sense since PM's packing efficiency is reduced when packing larger VMs. On average, FFDL1, MMDVMC, and AMDVMC achieve PEs of 2.7, 2.0, and 2.4, respectively. Furthermore, Figure 6.10(c) shows a bar chart representation of the power consumption of data center PMs after the VM consolidation decisions. It can be observed from the chart that, for all the algorithms, power consumption reduces with the increase of *MeanRsc*. Since, with the increase of mean VM resource demands, the algorithms release more PMs, which means that the VMs are packed into a reduced number of active PMs, this causes reduction in power consumption. On average, compared to MMDVMC, AMDVMC reduces the power consumption by 13%, whereas compared to FFDL1, it incurs 11% more power consumption.

And, a bar chart representation of the resource wastage of active PMs in the data center is shown in Figure 6.10(d). It can be seen from the chart that with the increase of *MeanRsc*, resource wastage is reduced gradually for MMDVMC. This indicates that MMDVMC can utilize multi-dimensional resources better for larger VMs compared to

smaller VMs. However, in the case of FFDL1 and AMDVMC, the resource wastage gradually reduces for smaller VM sizes. On average, compared to MMDVMC, AMDVMC reduces resource wastage by 34%, whereas compared to FFDL1, it incurs 42% more resource wastage. Therefore, it can be concluded from the results that, similar to the results for scaling N_p, for the gain factors, the AMDVMC algorithm outperforms the migration-aware MMDVMC algorithm, while AMDVMC performs poorly compared to the migration-unaware FFDL1 algorithm.

Figure 6.11 presents the four primary cost factors of dynamic VM consolidation decisions generated by the algorithms for different means of VM resource demands. Figure 6.11(a) shows how the estimate of aggregated migration data transmission (6.9) values with respect to *MeanRsc*. FFDL1, being migration-unaware, requires an increasing amount of migration-related data transmission as *MeanRsc* increases. This is due to the fact that, with the increase of *MeanRsc*, memory sizes of the VMs also increase, which in turn contributes to the rise of migration data transmission (Algorithm 6.1). MMDVMC, on the other hand, though considers minimizing the number of migrations, it does not consider the VM memory sizes while making migration decisions and assumes every VM migration to have the same migration overhead, and as consequence, its aggregated migration data transmission also increases with the increase of VM sizes. Lastly, in the case of AMDVMC, the estimate of aggregated migration data transmission is reduced with the increase of *MeanRsc*. This is because AMDVMC considers the estimate of migration data transmission (6.9) as a contributory factor for the migration overhead estimation and takes this migration overhead factor into account while making VM consolidation decisions. As a result, with the increase of *MeanRsc* (consequently, VM memory sizes) and decrease of N_v (as per Table 6.4), AMDVMC makes efficient selections of VMs for migration that in turn reduces the aggregated migration data transmission. The relative improvement of AMDVMC over FFDL1 and MMDVMC is depicted in Figure 6.11(b) which can be summarized as follows: on average, AMDVMC incurs 82% and 43% less aggregated migration data transmission compared to FFDL1 and MMDVMC, respectively.

Similar performance traits can be observed from Figure 6.11(c) and Figure 6.11(e). These figure show the estimates of aggregated migration time (6.10) and VM downtime (6.11) that results from the VM consolidation decisions made by the algorithms. With the increase of *MeanRsc* and VM memory sizes, both aggregated migration time and

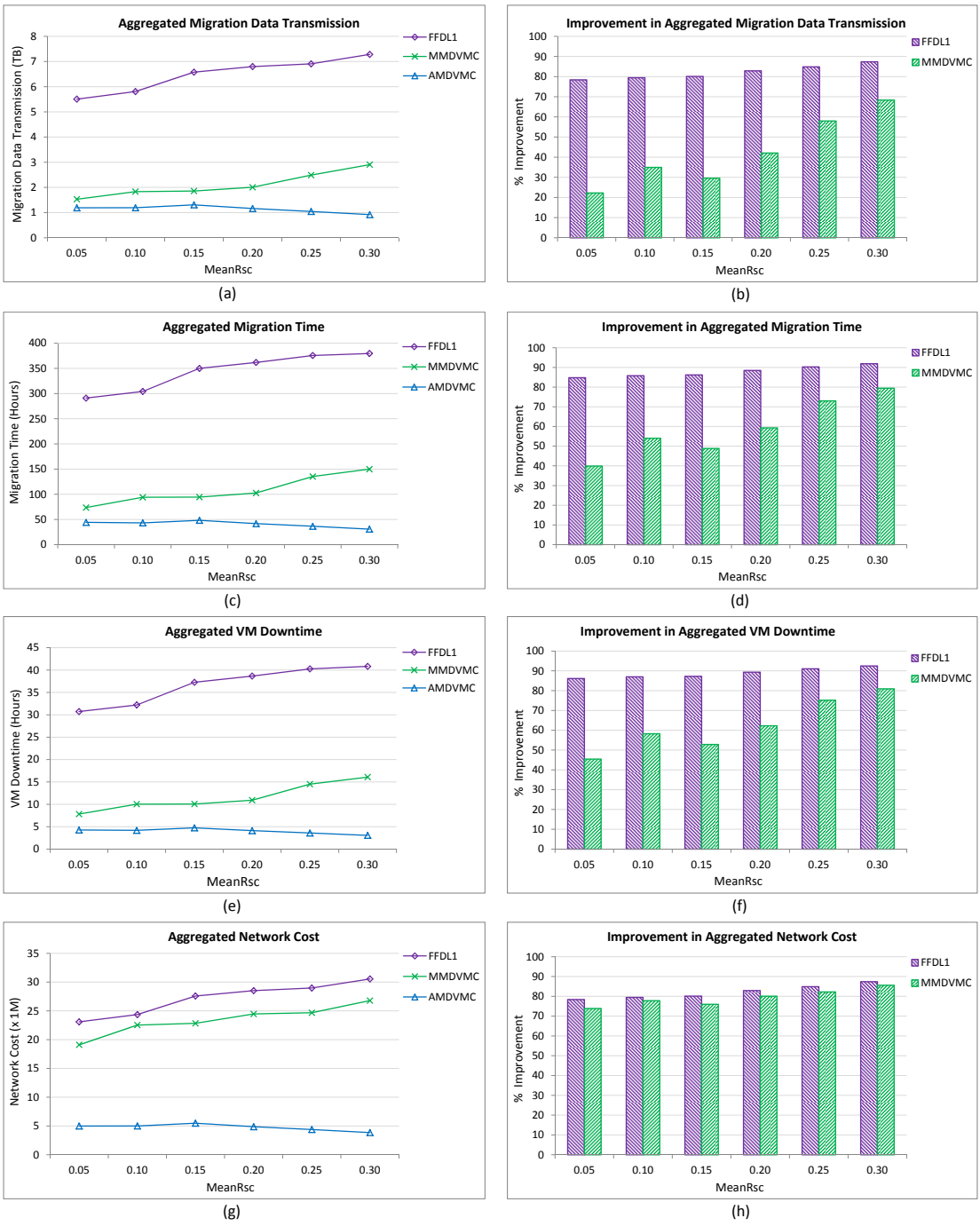

Figure 6.11: Performance of the algorithms with increasing (*MeanRsc*): (a) Aggregated Migration Data Transmission, (b) Improvement of AMDVMC over other algorithms in Aggregated Migration Data Transmission, (c) Aggregated Migration Time, (d) Improvement of AMDVMC over other algorithms in Aggregated Migration Time,(e) Aggregated VM Downtime, (f) Improvement of AMDVMC over other algorithms in Aggregated VM Downtime,(g) Aggregated Network Cost, (h) Improvement of AMDVMC over other algorithms in Aggregated Network Cost (best viewed in color).

VM downtime increase for FFDL1 and MMDVMC, whereas these values decrease for AMDVMC. This is due to the same reason as explained for migration data transmission

metric. Furthermore, Figure 6.11(d) and Figure 6.11(f) depict the relative improvement of the proposed AMDVMC algorithm over its competitors which can be summarized as follows. On average, AMDVMC reduces the aggregated migration time by 88% and 59% compared to FFDL1 and MMDVMC, respectively and it reduces the aggregated VM downtime by 89% and 63% compared to FFDL1 and MMDVMC, respectively, across all VM sizes.

The estimate of aggregated network cost (6.12) for each of the algorithms for different values of *MeanRsc* is presented in Figure 6.11(g). As the chart shows, the network cost for FFDL1 and MMDVMC increase gradually with the increase of *MeanRsc*. This is due to the fact that with the increase of *MeanRsc*, VM memory sizes also increase and the network cost is proportional to the amount of migration data transmission. It can be further observed that the network cost for AMDVMC decreases with respect to *MeanRsc*. This is again credited to the network cost awareness property of AMDVMC algorithm. Figure 6.11(h) shows the relative improvement of AMDVMC over FFDL1 and MMDVMC in terms of network cost for various VM sizes and on average, the improvements are 82% and 79%, respectively.

A summary of the overall migration overhead according to formulation (6.13) for various *MeanRsc* values is presented in Figure 6.12(a). It can be seen from the figure that AMDVMC incurs 85% and 61% less migration overhead compared to FFDL1 and MMDVMC, respectively. Figure 6.12(b) and Figure 6.12(d) present the estimate of aggregated migration energy consumption (6.14) and SLA violation (6.15) due to the VM consolidation decisions for each algorithm for various VM sizes. The figures show that, for FFDL1 and MMDVMC, both migration energy consumption and SLA violation increase with the increase of *MeanRsc*. This is due the fact that both FFDL1 and MMDVMC do not take into account the migration overhead factors while making consolidation decisions and therefore, the values of these metrics increase with the increase of VM memory sizes. Relative improvement of AMDVMC over FFDL1 and MMDVMC are presented in Figure 6.12(c) and Figure 6.12(e). In summary, compared to FFDL1 and MMDVMC, AMDVMC reduces the aggregated migration energy consumption by 82% and 42%, respectively and SLA violation by 89% and 64%, respectively.

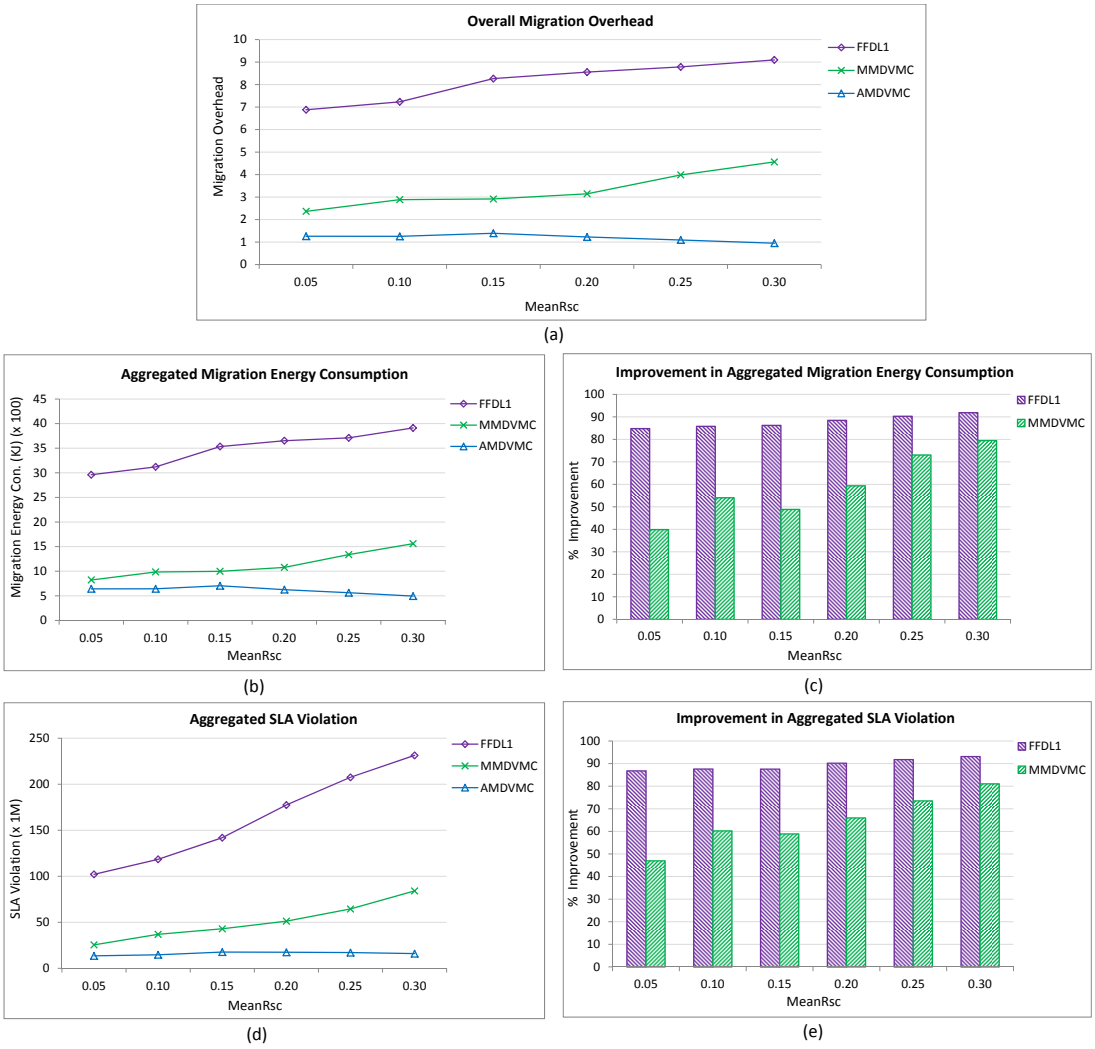

Figure 6.12: Performance of the algorithms with increasing *MeanRsc*: (a) Overall Migration Overhead, (b) Aggregated Migration Energy Consumption, (c) Improvement of AMDVMC over other algorithms in Aggregated Migration Energy Consumption, (d) Aggregated SLA Violation, (e) Improvement of AMDVMC over other algorithms in Aggregated SLA Violation (best viewed in color).

In light of the above results and discussion it can be summarized that, with the gradual increase of mean VM resource demand (*MeanRsc*) and corresponding decrease of the number of VMs (N_v), the power consumption and resource wastage of the data center slowly reduces for both FFDL1 and MMDVMC, whereas for AMDVMC the power consumption slowly reduces, but the resource wastage slightly increases. However, with the increase of *MeanRsc*, the cost factors consistently increase for both FFDL1 and MMDMVC, whereas they remain almost steady for AMDVMC. When compared with the migration-aware MMDMVC approach, the proposed AMDVMC algorithm outpaces MMDVMC on both the gain and cost factors, thereby indicating the superior quality of the VM consolidation

Table 6.5: Number of VMs (N_v) for corresponding *SDRsc* values

SDRsc	N_p	N_v
0.05	1024	2048
0.10	1024	1843
0.15	1024	1638
0.20	1024	1434
0.25	1024	1229
0.30	1024	1024

plans produced by AMDVMC. In contrast, the FFDL1 algorithm produces VM consolidation plans that require less power consumption and resource wastage compared to AMDVMC; however, this migration-unaware approach results in much higher migration overhead.

6.4.5 Diversification of Workload

This part of the experiment was conducted to assess the VM consolidation decisions generated by the algorithms by diversifying the workloads of the VMs in the data center. This was done by varying the standard deviation of the VM resource demands (*SDRsc*), where the initial value is set to 0.05 and gradually increased up to 0.3, with an increment of 0.05 each time. The maximum value for *SDRsc* was kept at 0.3 so that the VM's resource demand for any resource dimension (e.g., CPU, memory, or network I/O) was not too large compared to the PM's resource capacity for the corresponding resource dimension and by this way it helps to keep scope of consolidation. Similar to the approach presented in the Subsection 6.4.4, the number of PMs (N_p) in the data center was kept at 1024 and the number of VMs (N_v) was derived from the number of PMs using the following formulation:

$$N_v = N_p * (0.55 - SDRsc)/0.25. \tag{6.37}$$

Table 6.5 shows the different values for N_v produced by the above equation for each *SDRsc* value. The mean VM resource demand *MeanRsc* was set to 0.05.

Figure 6.13 presents the four gain factors for the algorithms while scaling the standard deviation *SDRsc* of VM resource demand. It can be observed from Figure 6.13(a) that, with the increase of workload diversification, the number of released PMs gradually decreases for FFDL1 and AMDVMC, whereas an opposite trend is found for MMDVMC. This can be explained as follows. Since FFDL1 works with the greedy strategy of First Fit, when the variation in the amount of resource demands for different resource types

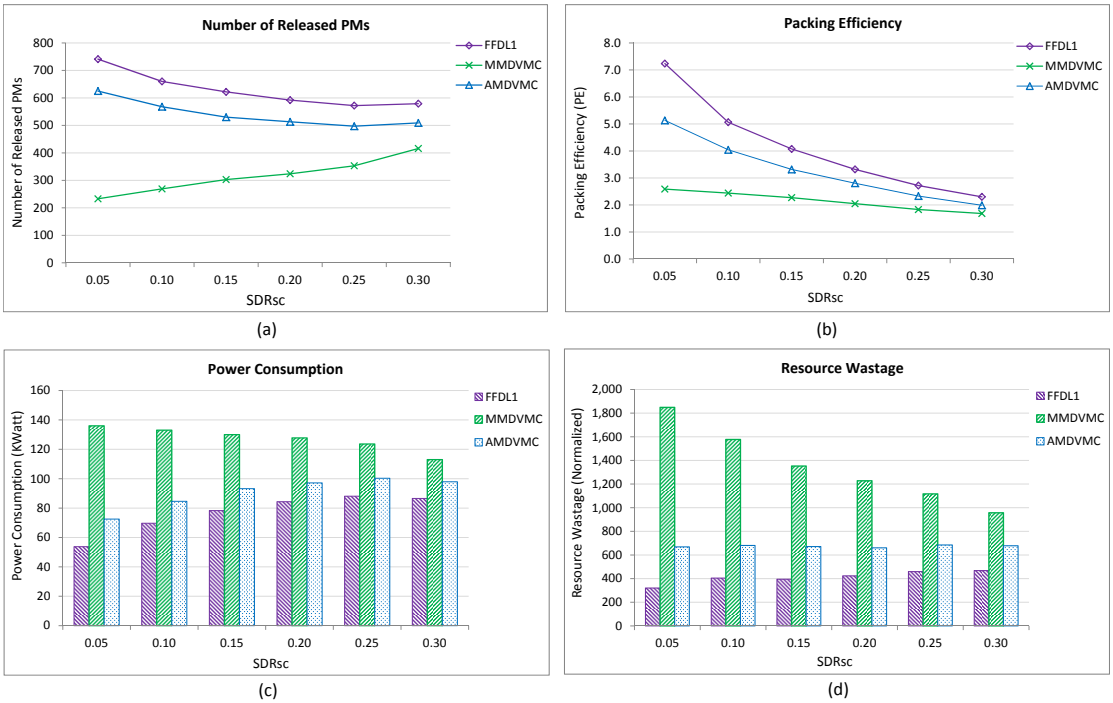

Figure 6.13: Performance of the algorithms with increasing *SDRsc*: (a) Number of Released PMs, (b) Packing Efficiency, (c) Power Consumption, and (d) Resource Wastage (best viewed in color).

increases, placement feasibility for the VMs decreases, and as a consequence, FFDL1 requires relatively more active PMs for higher *SDRsc* values. However, MMDVMC utilizes the MMAS metaheuristic [111] which is a iterative solution refinement method and therefore, can be effective even though resource demand variation is high. And in the case of the proposed AMDVMC, it utilizes the ACO metaheuristic [36] and at the same time being multi-objective, it also aims at reducing the migration overhead, and as a result, its performance, in terms of gain factors, reduces with the increase of *SDRsc* (which effectively increases the VM memory size for some VMs in the data center). Nevertheless, when the algorithms are compared, on average the proposed AMDVMC outperforms the MMDVMC algorithm by releasing 79% more PMs, whereas it release 14% fewer PMs compared to the migration-unaware FFDL1 algorithm.

With the increase of $SDRsc$, the packing efficiency of the algorithms gradually decreases as reflected in Figure 6.13(b). This is due to the fact that, with the increase of $SDRsc$, there is a higher probability of generating VMs with higher resource demands across the resource dimensions, which reduces the packing efficiency of PMs in the data center. Similar to the number of released PMs, the performance of the proposed AMDVMC lies between those of its competitor algorithms. On average, FFDL1, MMDVMC,

and AMDVMC achieve PEs of 4.1, 2.1, and 3.3, respectively. Figure 6.13(c) and Figure 6.13(d) depict power consumption and resource wastage (normalized) of the active PMs in the data center after the VM consolidation. Both figures demonstrate similar performance patterns for the algorithms across the $SDRsc$ values as in Figure 6.13(a). For FFDL1 and AMDVMC, since the number of active PMs increases with the increase of $SDRsc$, both power consumption and resource wastage gradually increase with respect to $SDRsc$. However, this is not the case with MMDVMC which reduces both power consumption and resource wastage with respect to $SDRsc$. Finally, on average, compared to MMDVMC, AMDVMC reduces the power consumption and resource wastage by 28% and 48%, respectively whereas, compared to FFDL1, it incurs 20% more power consumption and 66% more resource wastage.

Therefore, it can be concluded from the above results for gain factors that, similar to results for scaling N_p and $MeanRsc$, AMDVMC outperforms migration-aware MMDVMC algorithm, while the migration-unaware FFDL1 outdoes AMDVMC.

The four primary cost factors of dynamic VM consolidation with increasing diversity of workloads are shown in Figure 6.14. The estimate of aggregated migration data transmission (6.9) is depicted in Figure 6.14(a). Both for the FFDL1 and MMDVMC algorithms, migration data transmission increases as the VM workload increases. With the increase of $SDRsc$, more VMs tend to have larger memory sizes and as a consequence, migration data transmission for the FFDL1 algorithm increases steadily. In the case of MMDVMC, it is worth noting that it improves the gain factors steadily with the increase of $SDRsc$, and this is achieved at the cost of steady increase of migration data transmission and other cost factors. And, for the proposed AMDVMC algorithm, the migration data transmission slightly increases up to $SDRsc = 0.2$, and thereafter it decreases. The increase for the cases when $SDRsc \leq 0.2$ is explained by the fact that, as $SDRsc$ increases, the VM resource demands (including VM memory size) increases probabilistically, which in turn raises the migration data transmission. However, with the increase of $SDRsc$, the number of VMs (N_v) decreases according to (6.37), and AMDVMC, being migration overhead-aware, can reduce the migration data transmission for the relatively smaller number of VMs when $SDRsc > 0.2$. Figure 6.14(b) shows the performance improvement by AMDVMC over FFDL1 and MMDVMC. In summary, compared to FFDL1 and MMDVMC, on average AMDVMC requires 68% and 40% less migration data transmission, respectively.

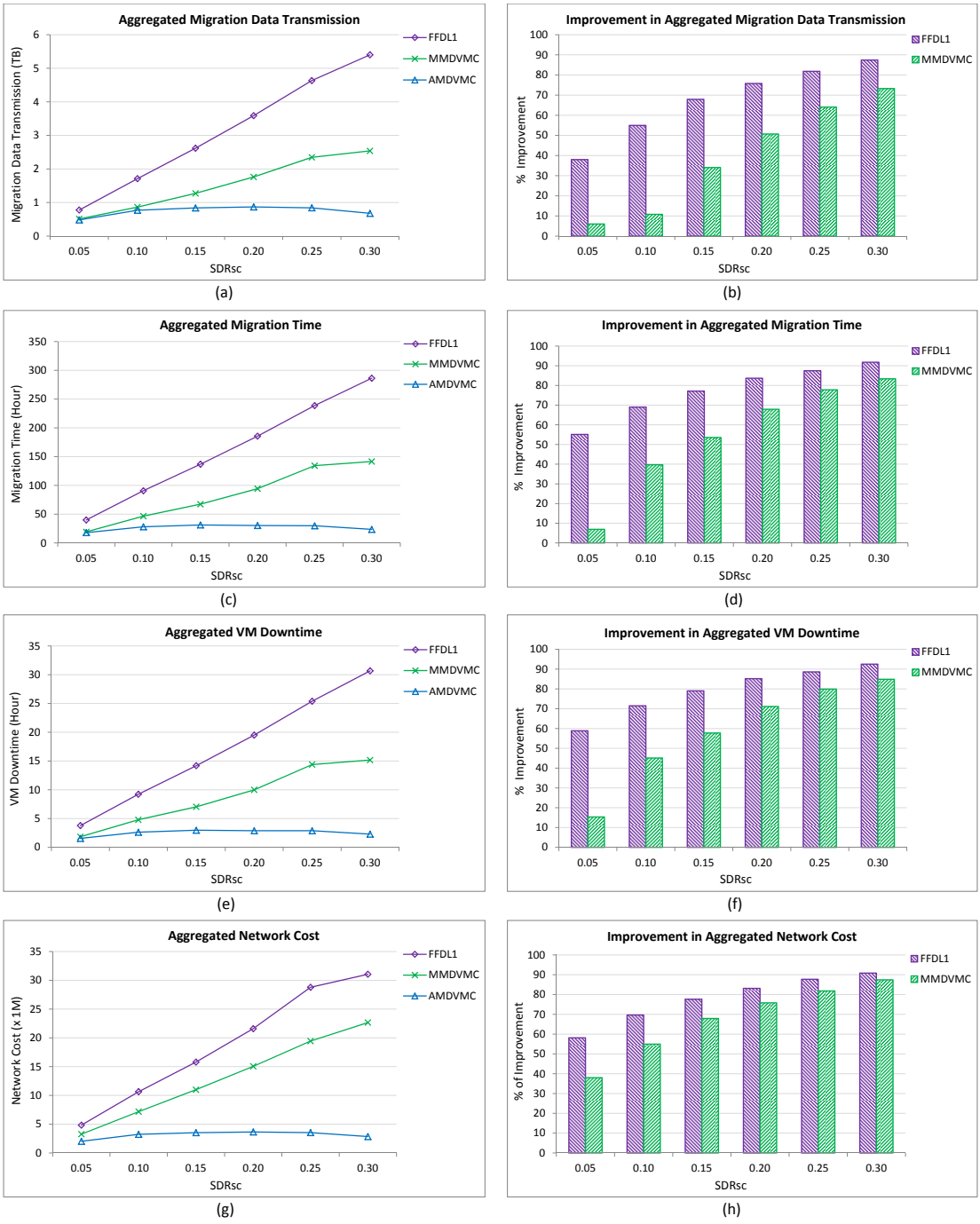

Figure 6.14: Performance of the algorithms with increasing *SDRsc*: (a) Aggregated Migration Data Transmission, (b) Improvement of AMDVMC over other algorithms in Aggregated Migration Data Transmission, (c) Aggregated Migration Time, (d) Improvement of AMDVMC over other algorithms in Aggregated Migration Time,(e) Aggregated VM Downtime, (f) Improvement of AMDVMC over other algorithms in Aggregated VM Downtime,(g) Aggregated Network Cost, (h) Improvement of AMDVMC over other algorithms in Aggregated Network Cost (best viewed in color).

Figure 6.14(c) and Figure 6.14(e) present the aggregated migration time (6.10) and VM downtime (6.11) for the algorithms across various values of $SDRsc$. It is evident from the figures that the performance patterns are similar to those for migration data

transmission (Figure 6.14(a)). Since both aggregated migration time and VM downtime are proportional to VM memory size, the above-mentioned explanation for migration data transmission metric also applies to these performance metrics. Figure 6.14(d) and Figure 6.14(f) depict bar chart representations for the relative performance improvement of AMDVMC over FFDL1 and MMDVMC that can be summarized as follows: on average, AMDVMC requires 77% and 55% less aggregated migration time and 79% and 59% less aggregated VM downtime compared to FFDL1 and MMDVMC, respectively, across all VM workload ranges.

Figure 6.14(g) shows the estimate of network cost (6.12) for different workload types. The network costs for both FFDL1 and MMDVMC algorithms increase sharply with the increase of $SDRsc$ since network cost is proportional to the migration data transmission, and both FFDL1 and MMDVMC are network cost oblivious. AMDVMC shows a similar performance pattern to that of Figure 6.14(a) and the same explanation applies for this performance metric as well. The relative performance improvement of AMDVMC over the other algorithms is presented in Figure 6.14(h). AMDVMC, being network cost-aware, incurs 78% and 68% less network cost than do FFDL1 and MMDVMC, respectively, on average across all $SDRsc$ values.

The uniform migration overhead (6.13) for all the algorithms is presented in Figure 6.15(a). In summary, the uniform migration overhead of AMDVMC is 73% and 57% less than FFDL1 and MMDVMC, respectively. Figure 6.15(b) and Figure 6.15(d) depict the estimate of aggregated migration energy consumption (6.14) and SLA violation (6.15) due to consolidation decisions across various $SDRsc$ values. Since migration-related energy consumption and SLA violation are proportional to migration data transmission and VM migration time, respectively, these two performance metrics display similar performance patterns to those of Figure 6.14(a) and Figure 6.14(c), respectively. Figure 6.15(c) and Figure 6.15(e) show bar chart representations of the relative improvement achieved by AMDVMC over other algorithms for aggregated migration energy consumption and SLA violation, respectively, which can be summarized as follows. When compared to FFDL1 and MMDVMC, AMDVMC incurs 68% and 40% less migration energy consumption, and 79% and 58% less SLA violation, respectively.

In view of the above results and analysis, it can be concluded that with the gradual increase of the diversification of workload ($SDRsc$) and a corresponding decrease in

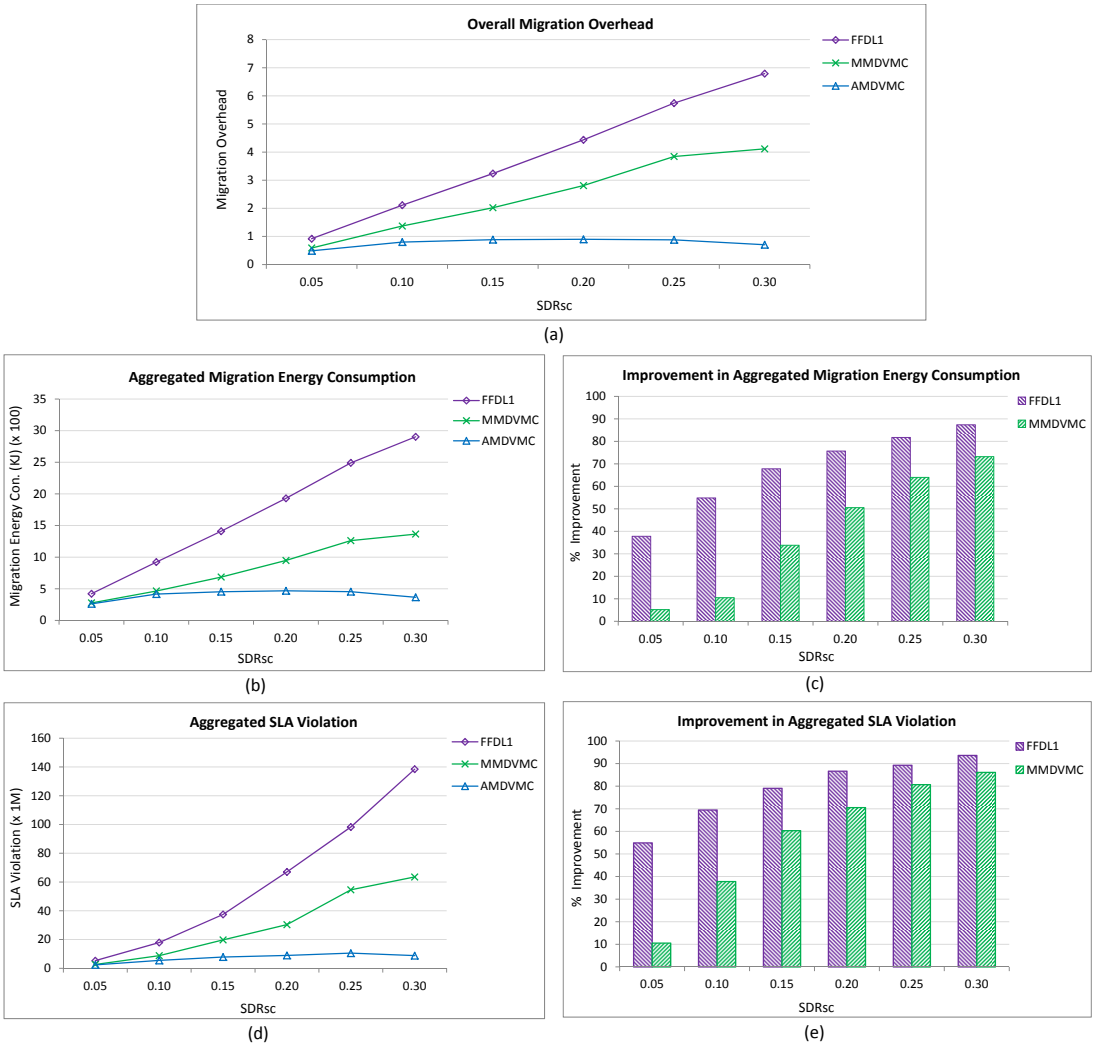

Figure 6.15: Performance of the algorithms with increasing *SDRsc*: (a) Overall Migration Overhead, (b) Aggregated Migration Energy Consumption, (c) Improvement of AMDVMC over other algorithms in Aggregated Migration Energy Consumption, (d) Aggregated SLA Violation, (d) Improvement of AMDVMC over other algorithms in Aggregated SLA Violation (best viewed in color).

the number of VMs (N_v), the power consumption and resource wastage of the data center slowly increase for both FFDL1 and AMDVMC, whereas these metrics decrease for MMDVMC. However, all the cost factors increase rapidly for both FFDL1 and MMDVMC with the increase of workload diversification, while these factors remain largely unchanged for AMDVMC across workload variations. When compared to the migration-aware MMDVMC, the proposed AMDVMC algorithm outperforms MMDVMC for both gain and cost factors. On the other hand, the migration-unaware FFDL1 algorithm achieves higher efficiency on power consumption and resource wastage than AMDVMC, however this is gained at the cost of very high migration overhead factors.

6.4.6 AMDVMC Decision Time

This part of the experiment was conducted in order to assess the feasibility of the proposed AMDVMC algorithm for performing offline, dynamic VM consolidation for data center environments discussed in the problem statement (Section 6.2). As presented in Subsection 6.3.2, scalability of the proposed dynamic VM consolidation is ensured by running the consolidation operation under the proposed hierarchical, decentralized framework where each cluster controller is responsible for generating VM consolidation decisions for its respective PM cluster. Therefore, when implemented using the decentralized framework where the proposed AMDVMC dynamic VM consolidation algorithm is executed by the cluster controllers separately and simultaneously for their respective PM clusters, it is the cluster size that has a potential effect on the solution computation time rather than the total number of PMs in the data center. Figure 6.16(a) shows AMDVMC's decision time for cluster sizes between 8 and 48. It can be observed that decision time increases smoothly and non-linearly with the cluster size, each time doubling the time for an additional 8 PMs in the cluster, even though the search space grows exponentially with N_{pc}. For a cluster of size 48, the decision time is around 15.4 seconds which is quite a reasonable run-time for an offline algorithm.

Figure 6.16(b) and Figure 6.16(c) show the solution computation time while scaling the mean ($MeanRsc$) and standard deviation ($SDRsc$) of VM resource demand for cluster size $N_{pc} = 8$. It can be observed from the figures that, in both instances, the decision time reduces with the increase of $MeanRsc$ and $SDRsc$. This is due to the fact that, in these instances, the number of VMs in the data center (N_v) declines with the increase of $MeanRsc$ and $SDRsc$ (Table 6.4 and Table 6.5), which reduces the solution computation time. In summary of these two cases, AMDVMC requires at most 0.05 second for computing consolidation plans. Therefore, when implemented using the proposed hierarchical, decentralized framework, it can be concluded that the proposed AMDVMC algorithm is a fast and feasible technique for offline, dynamic VM consolidation in large-scale data centers.

In order to assess the time complexity of AMDVMC for scenarios where the decentralized computation is not available, the solution computation time for a centralized system was also measured and analyzed. For this purpose, VM consolidation decisions for each

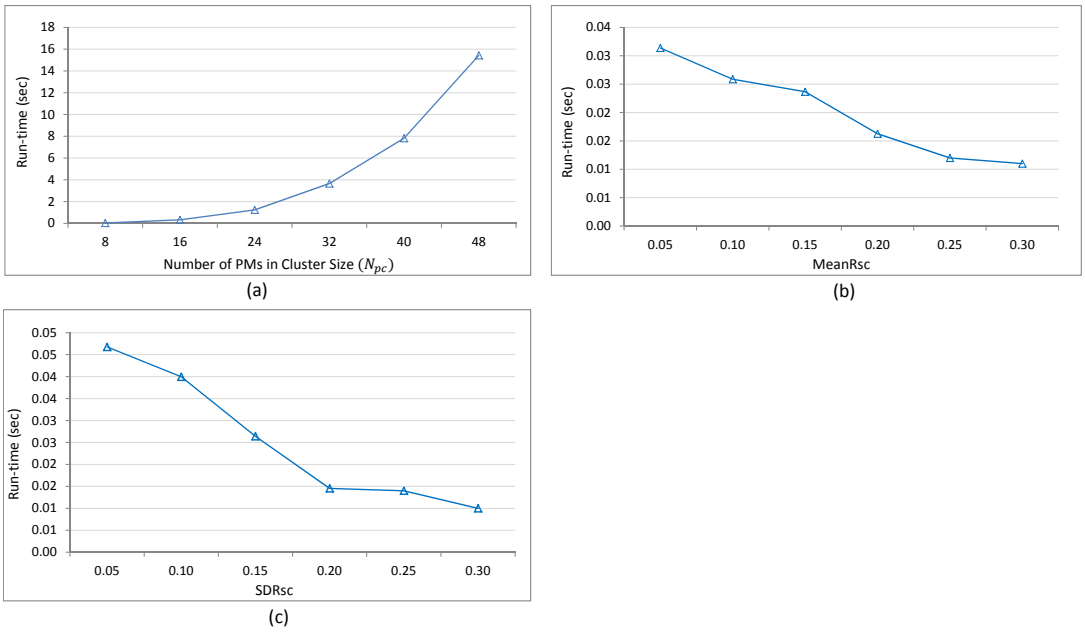

Figure 6.16: AMDVMC's VM consolidation decision time for decentralized implementation while scaling (a) PM cluster size (N_{pc}), (b) Mean of VM resource demand ($MeanRsc$), and (c) Diversification of VM workload ($SDRsc$).

of the PM clusters were computed in a centralized and single-threaded execution environment and the solution computation time for individual clusters were accumulated and reported in this evaluation. Figure 6.17 shows the average time needed by such centralized implementation of the AMDVMC algorithm for producing dynamic VM consolidation plans for the various scaling factors.

It can be observed from Figure 6.17(a) that the AMDVMC solution computation time increases smoothly and non-linearly with the number of PMs in the data center (N_p). It is evident from the figure that for a medium sized data center comprising 1024 PMs, AMDVMC requires around 4.3 seconds for computing the VM consolidation plan whereas for the largest data center simulated in this experiment with 4096 PMs (i.e., several thousand physical servers), AMDVMC needs around 30 seconds. Moreover, since AMDVMC utilizes the ACO metaheuristic which is effectively a multi-agent computation method, there is the potential for parallel implementation [98] of AMDVMC algorithm where individual ant agents can be executed in parallel in multiple Cloud nodes that can reduce the VM consolidation decision time significantly.

Furthermore, Figure 6.17(b) and Figure 6.17(c) show that the solution computation time of AMDVMC reduces with increasing $MeanRsc$ and $SDRsc$, respectively. This is also due to the above-mentioned fact that the number of VMs reduces with increasing

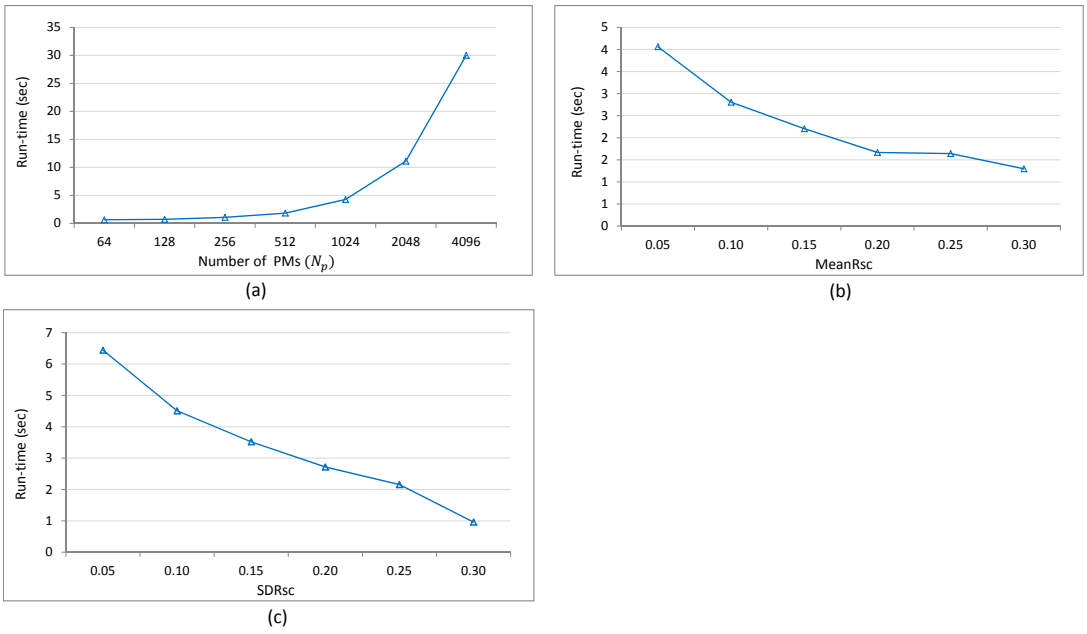

Figure 6.17: AMDVMC's VM consolidation decision time for centralized implementation while scaling (a) Data center size (N_p), (b) Mean of VM resource demand ($MeanRsc$), and (c) Diversification of VM workload ($SDRsc$).

mean and standard deviation of VM resource demands accordingly to (6.36) and (6.37), respectively. In summary of these two cases, AMDVMC requires at most 6.4 seconds for computing consolidation plans. Therefore, it can be concluded that, for centralized execution, the proposed AMDVMC algorithm is perfectly applicable for computing offline, dynamic VM consolidation plans for large-scale data centers.

6.5 Summary and Conclusions

Resource optimization has always been a challenging task for large-scale data center management. With the advent of Cloud Computing, and its rapid and wide adoption, this challenge has taken on a new dimension. In order to meet the increasing demand of computing resources, Cloud providers are deploying large data centers, consisting of thousands of servers. In these data centers, run-time underutilization of computing resources is emerging as one of the key challenges for successful establishment of Cloud infrastructure services. Moreover, this underutilization of physical servers is one of the main reasons for power inefficiencies in such data centers. Wide adoption of server virtualization technologies has opened opportunities for data center resource optimization. Dynamic VM consolidation is one of such techniques that helps in rearranging the active VMs among the physical servers in data centers by utilizing the VM live migration mechanism in order

to consolidate VMs into a minimal number of active servers so that idle servers can be turned to lower power states (e.g., standby mode) to save energy. Moreover, this approach helps in reducing the overall resource wastage of the running servers.

This chapter has addressed a multi-objective dynamic VM consolidation problem in the context of large-scale data centers. The problem was formally defined as a discrete combinatorial optimization problem with necessary mathematical models with the goals of minimizing server resource wastage, power consumption, and overall VM migration overhead. Since VM migrations have non-negligible impacts on hosted applications and data center components, an appropriate VM migration overhead estimation mechanism is also suggested that incorporates realistic migration parameters and overhead factors. Moreover, in order to address the scalability issues of dynamic VM consolidation operations for medium to large-scale data centers, a hierarchical, decentralized VM consolidation framework was proposed to localize VM migration operations and reduce their impacts on the data center network. Furthermore, based on ACO metaheuristic, a migration overhead-aware, multi-objective, dynamic VM consolidation algorithm (AMDVMC) was presented as a concrete solution for the defined run-time VM consolidation problem, integrating it with the proposed migration overhead estimation technique and decentralized VM consolidation framework.

In addition, comprehensive simulation-based performance evaluation and analysis have also been presented that demonstrate the superior performance of the proposed AMDVMC algorithm over the compared migration-aware consolidation approaches across multiple scaling factors and several performance metrics, where the results show that AMDVMC reduces the overall server power consumption by up to 47%, resource wastage by up to 64%, and migration overhead by up to 83%. Lastly, the feasibility of applying the proposed AMDVMC algorithm for offline dynamic VM consolidation in terms of decision time has been demonstrated by the performance evaluation, where it is shown that the algorithm requires less than 10 seconds for large server clusters when integrated with the proposed decentralized framework and a maximum of 30 seconds for large-scale data centers when executed in centralized mode.

Chapter 7

Conclusions and Future Directions

This chapter summarizes the research works on multi-objective Virtual Machine (VM) management presented in this thesis and outlines the notable research findings. Furthermore, it discusses open research problems and highlights a number of future research directions.

7.1 Conclusions and Discussion

Cloud Computing, a very recent paradigm shift in information technology industry, is growing rapidly with the goal of providing virtually infinite amount of computing, storage, and communication resources, where customers are provisioned these resources according to their demands using a pay-per-use business model. The rapidly growing customer demands for Cloud resources are responded by the Cloud providers with the deployment of large-scale data centers across the globe. These data centers incur very high investment and operating costs for the compute, storage, and network devices, as well as for the associated energy consumption. Moreover, because of the huge energy usage, such data centers leave large carbon footprints and thereby, adversely affects the environment. As a consequence, efficient management of resource utilization and energy consumption are crucial for ensuring the ultimate success and sustainability of Cloud Computing.

This thesis has investigated the problems of underutilization of data center resources and high energy consumption in the context of large-scale virtualized data centers. It has presented several aspects of utilizing server virtualization technologies to address the issues of resource utilization and energy consumption. Several Virtual Machine (VM) placement

and consolidation strategies, both online and offline, are proposed that can be utilized by the data center resource management system.

In the context of large-scale enterprise data centers, resource optimization through efficient VM placement and consolidation is a complex problem due to the multi-dimensionality aspect of server resources, complex structure of modern applications, and dynamic nature of data center environments. In fact, as discussed in the contributory chapters, both the problems of VM placement and consolidation, with the goal of reducing resource wastage, energy consumption, and VM management overhead, are instances of $\mathcal{NP}-hard$ combinatorial optimization problems. In order to tackle the identified research issues associated with efficient and scalable VM placement and consolidation, this thesis has achieved all the research objectives delineated in Chapter 1. A summary of the research findings and insights obtained from the contributions are discussed below.

Chapter 3 has presented an extensive analysis, taxonomy, and survey of the state-of-the-art VM management techniques, strategies, and policies, with focus on diversified optimization goals in relation to dynamic, virtualized data centers. This review on the existing research works and technologies has facilitated to identify the limitations, open issues, and future challenges in the field of VM management, and paved the way for determining the research direction undertaken in this thesis.

In Chapter 4, the multi-objective VM cluster placement problem has been modeled using an optimization framework that helps in representing the multiple objectives in a unified method. It is concluded from the mathematical representation that both the objectives of minimizing the overall server resource wastage and power consumption can be captured only by reducing the number of servers (homogeneous) used for the VM cluster placement. This indicates that consolidated VM cluster placement eventually helps solving the multi-objective placement problem. Moreover, it is also identified that sophisticated techniques for capturing server resource utilization, such as the proposed vector algebra-based method, helps in obtaining a uniform mean of the multi-dimensional server resource utilization. Such unified mean estimation approach has the benefit that it can be integrated with specialized VM management techniques with little or no adaptation. Another conclusion derived from this chapter is that the VMs offered by the Cloud providers demonstrate resource patterns that online VM placement techniques can exploit to optimize various goals. Moreover, it is identified in this chapter that, the consolidated VM

placement problem being $\mathcal{NP}-hard$, application of metaheuristics, such as Ant Colony Optimization (ACO), can effectively generate good quality solutions within polynomial time bound. This further implies that, with appropriate adaption, metaheuristic-based solution approaches have potential to be applicable and efficient for other optimization problems in the context of large-scale infrastructures.

Chapter 5 has presented another online VM placement approach focusing on network efficiency, while taking into account data communication among the VMs. The chapter has presented an observation that a major portion of modern Cloud applications have complex structures with multiple components, including VMs for computation and data blocks for storage, where such components usually have mutual communication dependencies among themselves. Considering this property of composite Cloud applications, the research presented in this chapter has concluded that simultaneous placement decisions of the VMs and the data blocks in data center components (such as servers and storage devices), taking into account of their mutual communication structure, have potential to improve network scalability. Being informed on the prospective communication requirements among application components, placement decisions have the capacity to work as preemptive measures for run-time network congestion and communication disruption. The addressed placement problem, being very closely related to the Quadratic Assignment Problem (QAP), is computationally hard and the proposed heuristic has been shown to be fast enough to compute efficient placement decisions, even for large data centers. The solution approach has emphasized on the localization of network traffic among the application components within the communication network, which have eventually helped in reducing the traffic load in the upper-layer network switches. This observation validates the conclusion that network-aware VM placement decisions have potential to provide benefit, which is complementary to the traditional developments on network architectures and routing protocols.

In addition to optimized VM placement during initial deployment, Chapter 6 has shown that dynamic VM management can further improve data center resource utilization and energy efficiency at run-time. Leveraging VM live migration technique, the proposed dynamic VM consolidation approach has been shown to improve resource utilization and power consumption, while keeping the overall migration overhead or impact minimal. One insight observed from the discussion presented in this chapter is that the impact of

a VM live migration operation depends on several migration parameters and therefore, live migration-based VM management policies should consider such parameters in order to make efficient decisions. Another important observation from this chapter is that centralized dynamic VM consolidation can suffer from scalability issues for center-wide consolidation. Since dynamic VM consolidation involves relocating VMs from current hosts to other servers that cause non-negligible amount of data transmission across the data center, decentralization is a necessary approach for designing scalable and efficient VM consolidation techniques. Furthermore, being a combinatorial optimization problem, the search space of the multi-objective dynamic VM consolidation grows exponentially with the number of active VMs and the proposed ACO metaheuristic-based solution approach has demonstrated to be efficient in balancing between potentially conflicting optimization goals and applicable as an offline approach in terms of time complexity in the context of Cloud data centers.

7.2 Future Research Directions

Cloud Computing, being a very dynamic environment, is rapidly evolving and opening new directions for further research and development. Moreover, VM management is a broad area of research, in particular in the context Cloud Computing. Based on the insights gained from the research presented in this thesis, the following open research challenges are identified.

- Cloud environments allow their consumers to deploy any kind of applications in an on-demand fashion, ranging from compute intensive applications, such as High Performance Computing (HPC) and scientific applications, to network and disk I/O intensive applications, such as video streaming and file sharing. Co-locating similar kinds of applications in the same physical server can lead to resource contentions for some types of resources, while leaving other types of resources underutilized. Moreover, such resource contention will have adverse effect on application performance, thus leading to Service Level Agreement (SLA) violations and profit minimization. Therefore, it is important to understand the behavior and resource usage patterns of the hosted applications in order to efficiently place VMs and allocate resources to the applications. Utilization of historical workload data and application of appropriate load prediction mechanisms need to be integrated with dynamic VM management

techniques in order to minimize resource contentions among applications, and increase resource utilization and energy efficiency of the data centers.

- Incentive-based VM migration and consolidation is yet another direction for future research in the area of Cloud Computing. Appropriate incentive policy can be formulated for Cloud consumers to trade off between incentive and SLA violation, which in turn will motivate Cloud providers to optimize infrastructure resources by specialized VM placement, migration, and consolidation strategies with the goal of saving energy and improving resource usage.

- VM migration coupled with VM profiling can be an efficient method for dynamic VM consolidation with simultaneous objectives of reducing energy consumption and resource wastage, as well as maintaining application performance. VM profile data can help consolidation policies to select appropriate VMs in order to reduce SLA violations. Since a diverse range of applications are deployed in Cloud data centers, different applications can have different performance requirements and levels of tolerance to SLA violations. Therefore, VM management policies can utilize such profile data to make efficient VM migration decisions in order to optimize resource utilization, while ensuring a minimal performance degradation.

- Deployment and management of composite applications comprising multiple computing (e.g., VMs) and data components (e.g., data blocks), where inter-component communication requirements are not sufficiently specified or are not known a priori at all, is an interesting problem that needs much attention. The primary challenges associated with such application management are appropriate modeling of the applications at hand and accurate prediction of the communication requirements, patterns, and structures. The problem is further complicated due to frequent variations in workloads experienced by modern applications and dynamic scaling of application components. Lastly, overprovisioning of network resources can potentially lead to network congestion, performance degradation, and SLA violations. Therefore, it is highly desirable that the run-time management system would be resilient enough to cope with the application dynamics.

- Incorporation of efficient network resource utilization in Cloud infrastructures, while making VM management decisions, is expected to be highly efficient. In particular,

placement of Cloud applications, with defined communication patterns focusing on both server resource optimization and network cost reduction, is an unexplored area that has high potential for further research and development. In this way, an overall VM management framework can be designed and implemented that will be aware of both server and network resource utilization, as well as power consumption.

- Widespread use of virtualization technologies, high speed communication, increased size of data and data centers, and above all, the broad spectrum of modern applications are opening new research challenges in network resource optimization. Appropriate combination and coordination of the online and offline VM placement and migration techniques with the goal of efficient network bandwidth management is one of the key areas for future research.

- VM migrations towards resource optimization as a continuous process, while taking into account migration impact, is another direction for future research. Such migrations can be effectively applied to move around VMs to release underloaded servers, so that they can be turned to lower power states to save energy. Also, VM migrations can be applied to offload overloaded servers in order to reduce performance degradation and maintain SLA. In such continuous optimization process, appropriate VMs can be selected by utilizing the VM migration overhead estimation approach proposed in Chapter 6 in order to achieve certain objectives, such as load balancing and SLA management.

- Periodic and threshold-based reconfiguration of application's virtual components using VM migration and reallocation, with focus on network traffic localization, can be an effective strategy for data center resource optimization. Such offline optimization technique must employ appropriate resource demand forecasting method, both for computing (e.g., CPU utilization) and network resources (e.g., bandwidth usage). In addition, VM migration and reconfiguration overhead can have significant impact on the performance of the hosted applications, as well as on the underlying communication substrate. Incorporation of reconfiguration overhead estimation with offline optimization techniques, focusing on multiple objectives, will produce pragmatic VM migration schemes, trading off between resource usage optimization and reconfiguration overhead.

Bibliography

[1] Bill Adra, Annika Blank, Mariusz Gieparda, Joachim Haust, Oliver Stadler, and Doug Szerdi. *Advanced POWER virtualization on IBM eserver P5 servers: Introduction and basic configuration.* IBM Corp., 2004.

[2] Shubham Agrawal, Sumit Kumar Bose, and Srikanth Sundarrajan. Grouping genetic algorithm for solving the serverconsolidation problem with conflicts. In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 1–8. ACM, 2009.

[3] Yasuhiro Ajiro and Atsuhiro Tanaka. Improving packing algorithms for server consolidation. In *Proceedings of the 2007 International Conference for the Computer Measurement Group (CMG)*, pages 399–406, 2007.

[4] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W Moore, and Andy Hopper. Predicting the performance of virtual machine migration. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 37–46. IEEE, 2010.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[6] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[7] Gordon C Armour and Elwood S Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 9(2):294–309, 1963.

[8] Marcos D Assunção, Rodrigo N Calheiros, Silvia Bianchi, Marco AS Netto, and Rajkumar Buyya. Big Data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79:3–15, 2015.

[9] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[10] A. Beloglazov and R. Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2010.

[11] Anton Beloglazov. *Energy-efficient management of Virtual Machines in data centers for Cloud computing*. PhD thesis, Department of Computing and Information Systems, The University of Melbourne, February 2013.

[12] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of Virtual Machines in Cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[13] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *Proceedings of the Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. IEEE, 2008.

[14] Laxmi N. Bhuyan and Dharma P. Agrawal. Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers*, 100(4):323–333, 1984.

[15] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Mitigating multi-tenancy risks in IaaS cloud through constraints-driven virtual resource scheduling. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 63–74. ACM, 2015.

[16] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware VM placement for Cloud systems. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pages 498–506. IEEE Computer Society, 2012.

[17] B. Brugger, K. Doerner, R. Hartl, and M. Reimann. Antpacking-an ant colony optimization approach for the one-dimensional bin packing problem. *Evolutionary Computation in Combinatorial Optimization*, pages 41–50, 2004.

[18] Rainer E Burkard, Eranda Cela, Panos M Pardalos, and Leonidas S Pitsoulis. The quadratic assignment problem. In *Handbook of Combinatorial Optimization*, pages 1713–1809. Springer, 1998.

[19] Rainer E Burkard and Franz Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17(2):169–174, 1984.

[20] R. Buyya, J. Broberg, and Andrzej Go sci nski. *Cloud Computing: Principles and Paradigms*. Wiley Online Library, 2011.

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

[22] Alberto Caprara and Paolo Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231–262, 2001.

[23] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the 10th Symposium on Discrete Algorithms*, pages 185–194, 1999.

[24] Ming Chen, Hui Zhang, Ya-Yunn Su, Xiaorui Wang, Guofei Jiang, and Kenji Yoshihira. Effective VM sizing in virtualized data centers. In *Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management*, pages 594–601. IEEE, 2011.

[25] Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2014-2019. [online] `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf`, October 2015. accessed on Aug. 30, 2016.

[26] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[27] Jack Clark. It now 10 percent of world's electricity consumption, report finds. [Online] http://www.theregister.co.uk/2013/08/16/it_electricity_use_worse_than_you_thought/, August 2013. Accessed: 29th September 2016.

[28] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[29] S. Crosby and D. Brown. The virtualization reality. *Queue*, 4(10):34–41, 2006.

[30] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Efficient datacenter resource utilization through cloud resource overcommitment. In *Proceedings of the 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, number 50, pages 330–335, 2015.

[31] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. *ACM SIGPLAN Notices*, 49(4):127–144, 2014.

[32] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3:159–168, 1990.

[33] Daniel S Dias and Luis Henrique MK Costa. Online traffic-aware Virtual Machine placement in data center networks. In *Proceedings of the 2012 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–8. IEEE, 2012.

[34] Qi Dong and Donggang Liu. Resilient cluster leader election for wireless sensor networks. In *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1–9. IEEE, 2009.

[35] M. Dorigo, M. Birattari, and T. Stutzle. Ant Colony Optimization. *IEEE Computational Intelligence Magazine*, 1(4):28 –39, 2006.

[36] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, Apr 1997.

[37] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

[38] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. Bradford Book. MIT Press, 2004.

[39] Marco Dorigo, Eric Bonabeau, and Guy Theraulaz. Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[40] Marco Dorigo and Thomas Stützle. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In *Handbook of Metaheuristics*, pages 250–285. Springer, 2003.

[41] Deniz Ersoz, Mazin S Yousif, and Chita R Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, pages 59–59. IEEE, 2007.

[42] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2):13–23, 2007.

[43] Fahimeh Farahnakian, Adnan Ashraf, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Ivan Porres, and Hannu Tenhunen. Using ant colony system to consolidate vms for green cloud computing. *IEEE Transactions on Services Computing*, 8(2):187–198, 2015.

[44] Sándor P Fekete and Jörg Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91(1):11–31, 2001.

[45] Eugen Feller, Christine Morin, and Armel Esnault. A case for fully decentralized dynamic vm consolidation in clouds. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012)*, pages 26–33. IEEE, 2012.

[46] Eugen Feller, Louis Rilling, and Christine Morin. Energy-aware Ant Colony based workload placement in Clouds. In *Proceedings of the 12th IEEE/ACM International Conference on Grid Computing (GRID 2011)*, pages 26–33. IEEE Computer Society, 2011.

[47] Md Hasanul Ferdaus and Manzur Murshed. *Cloud Computing: Challenges, Limitations and R&D Solutions*, chapter Energy-Aware Virtual Machine Consolidation in IaaS Cloud Computing, pages 179–208. Computer Communications and Networks. Springer International Publishing, 2014.

[48] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N Calheiros, and Rajkumar Buyya. Virtual Machine consolidation in Cloud data centers using ACO metaheuristic. In *Proceedings of the 20th International European Conference on Parallel and Distributed Computing (Euro-Par 2014)*, pages 306–317. Springer International Publishing, 2014.

[49] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. *Emerging Research in Cloud Distributed Computing Systems*, chapter Network-aware Virtual Machine Placement and Migration in Cloud Data Centers, pages 42–91. IGI Global, 2015.

[50] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective Ant Colony System algorithm for Virtual Machine placement in Cloud computing. *Journal of Computer and System Sciences*, 79(8):1230–1242, 2013.

[51] Stefanos Georgiou, Konstantinos Tsakalozos, and Alex Delis. Exploiting network-topology awareness for VM placement in IaaS Clouds. In *Proceedings of the 3rd International Conference on Cloud and Green Computing (CGC 2013)*, pages 151–158. IEEE, 2013.

[52] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. 39(4):51–62, 2009.

[53] Marco Guazzone, Cosimo Anglano, and Massimo Canonico. Energy-efficient resource management for cloud computing infrastructures. In *Proceedings of the Third IEEE*

International Conference on Cloud Computing Technology and Science (CloudCom 2011), pages 424–431. IEEE, 2011.

[54] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[55] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. 38(4):75–86, 2008.

[56] Rohit Gupta, Sumit Kumar Bose, Srikanth Sundarrajan, Manogna Chebiyam, and Anirban Chakrabarti. A two stage heuristic algorithm for solving the server consolidation problem with item-item and bin-item incompatibility constraints. In *Proceedings of the 2008 IEEE International Conference on Services Computing*, volume 2, pages 39–46. IEEE, 2008.

[57] Rui Han, Moustafa M Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Computer Systems*, 32:82–98, 2014.

[58] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[59] Sijin He, Li Guo, and Yike Guo. Real time elastic cloud management for limited resources. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 622 –629, july 2011.

[60] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 09)*, pages 41–50, New York, NY, USA, 2009. ACM.

[61] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[62] Daochao Huang, Yangyang Gao, Fei Song, Dong Yang, and Hongke Zhang. Multi-objective Virtual Machine migration in virtualized data center environments. In *Proceedings of the 2013 IEEE International Conference on Communications (ICC)*, pages 3699–3704. IEEE, 2013.

[63] Daochao Huang, Dong Yang, Hongke Zhang, and Lei Wu. Energy-aware Virtual Machine placement in data centers. In *Proceedings of the 2012 IEEE Conference on Global Communications (GLOBECOM)*, pages 3243–3249. IEEE, 2012.

[64] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. *Middleware 2009*, pages 163–183, 2009.

[65] Juniper. Cloud-ready data center reference architecture. [online] `http://www.juniper.net/us/en/local/pdf/brochures/1600040-en.pdf`, June 2012. Accessed on Aug. 30, 2016.

[66] N. Jussien, G. Rochart, and X. Lorca. The choco constraint programming solver. In *Proceedings of the 2008 Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP)*, 2008.

[67] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013.

[68] Dharmesh Kakadia, Nandish Kopri, and Vasudeva Varma. Network-aware Virtual Machine consolidation for large data centers. In *Proceedings of the 2013 Third International Workshop on Network-Aware Data Management (NDM 2013)*, pages 6:1–6:8. ACM, 2013.

[69] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 202–208. ACM, 2009.

[70] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the 2007 Linux Symposium*, volume 1, pages 225–230, 2007.

[71] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. DENS: data center energy-efficient network-aware scheduling. *Cluster Computing*, 16(1):65–75, 2013.

[72] Madhukar Korupolu, Aameek Singh, and Bhuvan Bamba. Coupled placement in modern data centers. In *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2009.

[73] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[74] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 100(10):892–901, 1985.

[75] George Leopold. AWS Rates Highest on Cloud Reliability. [Online] http://www.enterprisetech.com/2015/01/06/aws-rates-highest-cloud-reliability/, January 2015. Accessed: 29th September, 2016.

[76] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, 2004.

[77] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud: an energy-saving application live placement approach for cloud computing environments. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, pages 17–24. IEEE, 2009.

[78] Haikun Liu, Hai Jin, Cheng-Zhong Xu, and Xiaofei Liao. Performance and energy modeling for live migration of Virtual Machines. *Cluster Computing*, 16(2):249–264, 2013.

[79] Virginia Lo, Dayi Zhou, Yuhong Liu, Chris GauthierDickey, and Jun Li. Scalable supernode selection in peer-to-peer overlay networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 18–25. IEEE, 2005.

[80] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[81] Vijay Mann, Akanksha Gupta, Partha Dutta, Anilkumar Vishnoi, Parantapa Bhattacharya, Rishabh Poddar, and Aakash Iyer. Remedy: Network-aware steady state vm management for data centers. In *Proceedings of the 2012 International Conference on Research in Networking*, pages 190–204. Springer, 2012.

[82] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. Server consolidation in clouds through gossiping. In *Proceedings of the 2011 IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2011.

[83] David G McVitie and Leslie B Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

[84] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *Proceedings of the 7th International Conference on Autonomic Computing (ICAC 2010)*, pages 11–20. ACM, 2010.

[85] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware Virtual Machine placement. In *Proceedings of the 29th IEEE Conference on Computer Communication (INFOCOM 2010)*, pages 1–9. IEEE, 2010.

[86] Rich Miller. Inside Amazon's Cloud Computing Infrastructure. [Online] http://datacenterfrontier.com/inside-amazon-cloud-computing-infrastructure/, September 2015. Accessed: 29th September, 2016.

[87] Mark P Mills. The Cloud begins with Coal: Big Data, Big Networks, Big Infrastructure, and Big Power - An Overview of the Electricity Used by the Global Digital Ecosystem. [Online] http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf, August 2013. Accessed: 29th September 2016.

[88] M. Mishra and A. Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 275–282, July 2011.

[89] Rafael Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente. Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, 17(4):18–25, 2013.

[90] Aziz Murtazaev and Sangyoon Oh. Sercon: Server consolidation algorithm using live migration of virtual machines for green computing. *IETE Technical Review*, 28(3):212–231, 2011.

[91] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, volume 9, pages 39–50, 2009.

[92] A.B. Nagarajan, F. Mueller, C. Engelmann, and S.L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, pages 23–32. ACM, 2007.

[93] M. Nelson, B.H. Lim, G. Hutchins, et al. Fast transparent migration for virtual machines. In *Proceedings of the 2005 USENIX Annual Technical Conference*, pages 25–25, 2005.

[94] Trung Hieu Nguyen, Mario Di Francesco, and Antti Yla-Jaaski. A multi-resource selection scheme for Virtual Machine consolidation in Cloud data centers. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*, pages 234–239. IEEE, 2014.

[95] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

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

[97] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *Technical Report, Microsoft Research*, 2011.

[98] Martín Pedemonte, Sergio Nesmachnow, and Héctor Cancela. A survey on parallel ant colony optimization. *Applied Soft Computing*, 11(8):5181–5197, 2011.

[99] Industry Perspectives. Using a total cost of ownership (tco) model for your data center. http://www.datacenterknowledge.com/archives/2013/10/01/using-a-total-cost-of-ownership-tco-model-for-your-data-center/, October 2013. last accessed: 02-01-2014.

[100] Jing Tai Piao and Jun Yan. A network-aware Virtual Machine placement and migration approach in Cloud computing. In *Proceedings of the 9th International Conference on Grid and Cooperative Computing (GCC 2010)*, pages 87–92. IEEE, 2010.

[101] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[102] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[103] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[104] Constantine P Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. *ACM SIGOPS Operating Systems Review*, 36(SI):377–390, 2002.

[105] Huzur Saran and Vijay V Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.

[106] Vivek Shrivastava, Petros Zerfos, Kang-won Lee, Hani Jamjoom, Yew-Huey Liu, and Suman Banerjee. Application-aware Virtual Machine migration in data centers. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM 2011)*, pages 66–70. IEEE, 2011.

[107] Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, page 53. IEEE Press, 2008.

[108] J. Smith and R. Nair. *Virtual Machines: versatile platforms for systems and processes*. Morgan Kaufmann, 2005.

[109] Fei Song, Daochao Huang, Huachun Zhou, and Ilsun You. Application-aware Virtual Machine placement in data centers. In *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, pages 191–196. IEEE, 2012.

[110] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

[111] T. Stützle and H.H. Hoos. Max–min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[112] C. Takemura and L.S. Crawford. *The Book of Xen: A Practical Guide for the System Administrator*. No Starch Press, 2009.

[113] Ibrahim Takouna, Wesam Dawoud, and Christoph Meinel. Analysis and simulation of hpc applications in virtualized data centers. In *Proceedings of the 2012 IEEE International Conference on Green Computing and Communications (GreenCom)*, pages 498–507. IEEE, 2012.

[114] Ibrahim Takouna, Roberto Rojas-Cessa, Kai Sachs, and Christoph Meinel. Communication-aware and energy-efficient scheduling for parallel applications in virtualized data centers. In *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, pages 251–255. IEEE, 2013.

[115] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, volume 33, pages 291–302. ACM, 2005.

[116] Hien Nguyen Van, F.D. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proceedings of the 9th IEEE International Conference on Computer and Information Technology (CIT 2009)*, volume 1, pages 357 –362, 2009.

[117] Hien Nguyen Van, F.D. Tran, and J.-M. Menaud. Performance and power management for cloud infrastructures. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing (CLOUD 2010)*, pages 329 –336, july 2010.

[118] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[119] Adam Vaughan. How viral cat videos are warming the planet. [Online] http://www.theguardian.com/environment/2015/sep/25/server-data-centre-emissions-air-travel-web-google-facebook-greenhouse-gas, September 2015. Accessed: 29th September, 2016.

[120] A. Verma, G. Kumar, and R. Koller. The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial Track*, pages 11–16. ACM, 2010.

[121] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Proceedings of the 2008 ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 243–264. Springer, 2008.

[122] Akshat Verma, Gautam Kumar, Ricardo Koller, and Aritra Sen. Cosmig: Modeling the impact of reconfiguration in a cloud. In *Proceedings of the 19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 3–11. IEEE, 2011.

[123] W. Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, 2008.

[124] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. *Cloud Computing*, pages 254–265, 2009.

[125] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[126] Shao-Heng Wang, Patrick P-W Huang, Charles H-P Wen, and Li-Chun Wang. EQVMP: Energy-efficient and qos-aware Virtual Machine placement for software defined datacenter networks. In *Proceedings of the 2014 International Conference on Information Networking (ICOIN)*, pages 220–225. IEEE, 2014.

[127] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923 – 2938, 2009.

[128] Quanwang Wu and Fuyuki Ishikawa. Heterogeneous virtual machine consolidation using an improved grouping genetic algorithm. In *Proceedings of the 17th IEEE International Conference on High Performance and Communications (HPCC 2015)*, pages 397–404. IEEE, 2015.

[129] Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom) and International Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188. IEEE, 2010.

[130] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[131] Edward Zayas. Attacking the process migration bottleneck. 21(5):13–24, 1987.

[132] Bolei Zhang, Zhuzhong Qian, Wei Huang, Xin Li, and Sanglu Lu. Minimizing communication traffic in data centers with power-aware VM placement. In *Proceedings*

of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012), pages 280–285. IEEE, 2012.

[133] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.