Multi-objective mechanisms for dynamic resource adaptation in virtualized network environments

by

Mirna AWAD

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE IN PARTIAL FULFILLMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, AUGUST 08, 2023

ECOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mrs. Nadjia Kara, Thesis Supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Aris Leivadeas, Thesis Co-supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Zbigniew Duong, President of the Board of Examiners
Department of Electrical Engineering, École de technologie supérieure

Mr. Abdelouahed Gherbi, Member of the jury
Department of Electrical Engineering, École de technologie supérieure

Mr. Mohammad Hamdaqa, External Examiner
Department of Computer Engineering and Software Engineering, École Polytechnique de Montréal

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

ON AUGUST 04, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FOREWORD

The present thesis extensively explores various aspects of resource management in virtualized network environments, which has been and is still a challenging and broad subject. It addresses this problem from multiple perspectives and targets various sub-research problems, resulting in the production of three journal articles. Two of these articles have already been published in top journals, while the third is currently under review. Throughout the thesis, aside from the introductory and literature review chapters that provide an explanation of the research objectives and position them within the most recent state-of-the-art, the subsequent chapters present the journal articles without any modifications. Although each article discusses different contributions, they are all closely interconnected and complement one another.

ACKNOWLEDGMENTS

I am deeply grateful to the countless persons who have supported and guided me throughout my PHD journey. Without their unwavering assistance, this work would not have come to fruition.

First and foremost, I express my heartfelt appreciation to my family for their constant support and unwavering encouragement. Their presence and understanding have been instrumental in keeping me resilient, especially during the challenging times we faced, such as the COVID pandemic. Their patience and belief in me have been a constant source of strength, and I am forever grateful.

I extend my sincere thanks to my supervisors, for their financial support and invaluable contributions to this research endeavor. Their insightful feedback and guidance have consistently pushed me to elevate the quality of the work. Special thanks to Ericsson Canada for their collaboration in this research work.

I am immensely grateful for the professors at ETS who bestowed upon me the honor of teaching their courses and laboratories. I am truly appreciative of the valuable opportunities they presented me to enrich my curriculum vitae. Their trust in my teaching abilities, evident in their willingness to entrust me with classes of approximately 40 students, is something for which I am deeply thankful.

I would like to express my appreciation to my professors in the IT department at Lebanese University, and to Dr. Azzam Mourad, for recommending and encouraging me to pursue this study. Their support and belief in my potential to become a future professor have been invaluable.

Finally, I would like to thank all my friends who have shared joyful moments with me and engaged in activities that have enriched my life outside of studying and working. Their

companionship and support have brought balance and joy to this journey, and I am truly grateful for their presence.

Mécanismes multi-objectifs pour l'adaptation dynamique des ressources dans les environnements de réseaux virtualisés

Mirna AWAD

RÉSUMÉ

La croissance des technologies de virtualisation et des solutions cloud a fondamentalement transformé la gestion et l'utilisation des ressources informatiques. Ces avancées ont permis aux fournisseurs de cloud d'offrir des services évolutifs, flexibles et rentables à leurs clients. Cependant, avec la demande croissante de services cloud, la gestion efficace des ressources est devenue une préoccupation majeure. Pour relever ce défi, diverses stratégies, telles que la consolidation des ressources, la prédiction de l'utilisation des ressources et les techniques d'élasticité et de migration des ressources, ont été proposées afin d'optimiser la gestion des ressources dans les environnements virtualisés. Cette thèse présente un ensemble novateur de techniques adaptatives de gestion des ressources, conçues pour maximiser l'utilisation des ressources, réduire la consommation d'énergie et garantir la conformité aux exigences des contrats de service (SLA).

En tenant compte des défis inhérents à la variabilité des charges de trafic, à la diversité des applications et aux objectifs d'optimisation contradictoires, cette recherche englobe plusieurs contributions significatives, chacune se concentrant sur un aspect spécifique de la gestion des ressources. Tout d'abord, le problème d'adaptation dynamique des ressources dans les environnements de virtualisation des fonctions réseau (NFV) est exploré, en intégrant des stratégies d'élasticité et de migration des ressources pour les chaînes de fonctions virtualisées (SFC). Ce problème d'allocation des ressources est abordé sous un angle innovant, formulé comme un modèle de programmation linéaire (ILP) et développé pour générer des solutions optimales. Deuxièmement, des algorithmes décisionnels métaheuristiques innovants et multi-objectifs, basés sur NSGAII, CRO et PSO, sont proposés pour adapter les ressources en temps réel avec des solutions sous-optimales. Troisièmement, la réallocation proactive des ressources est étudiée grâce au développement d'un modèle de prédiction des charges de trafic multi-ressources et multi-étapes. En intégrant le filtre de Kalman et la régression par vecteur de support (SVR), ce modèle anticipe avec précision la consommation des ressources des hôtes, y compris le CPU, la mémoire et la bande passante. Quatrièmement, en s'appuyant sur cette capacité prédictive, une approche de consolidation optimisée est présentée, incorporant des stratégies d'estimation proactive de l'état des hôtes pour la détection de la surcharge et de la sous-charge. Pour valider l'efficacité des techniques proposées, des expérimentations approfondies sont menées en utilisant des ensembles de données divers tels que Planetlab, Materna et Bitbrains, couplés au simulateur Cloudsim. Les résultats démontrent le potentiel de ces techniques pour améliorer la gestion des ressources dans les environnements virtualisés.

Mots-clés : gestion et réallocation des ressources, élasticité et migration des ressources, virtualisation des fonctions réseau, infonuagique, méta-heuristique, algorithme génétique

NSGAII, optimisation des réactions chimiques CRO, optimisation des essaims de particules PSO, modèle ILP, chaînes de services, prédiction des charges de trafic, consolidation, régression vectorielle de support SVR, filtre de Kalman, Cloudsim.

Multi-objective mechanisms for dynamic resource adaptation in virtualized network environments

Mirna AWAD

ABSTRACT

The growth of virtualization technologies and cloud solutions has fundamentally transformed the management and utilization of computing resources. These advancements have empowered cloud providers to offer scalable, flexible, and cost-effective services to their customers. However, with the ever-increasing demand for cloud services, effective resource management has become a critical concern. To address this challenge, various strategies, such as workload consolidation, resource utilization prediction, and resource scaling and migration techniques, have been proposed to optimize resource management in virtualized environments. This thesis presents a pioneering set of adaptive multi-objective resource management techniques designed to maximize resource utilization, reduce energy consumption, and ensure compliance with Service Level Agreement (SLA) requirements.

Considering the inherent challenges of workload variability, application diversity, and conflicting optimization goals, the research encapsulates several significant contributions, each focusing on a specific aspect of resource management. **First**, the dynamic resource adaptation problem within Network Function Virtualization (NFV)-cloud environments is explored, incorporating resource scaling and migration strategies for service function chains (SFCs). This resource allocation problem is tackled from a novel perspective, formulated as Integer Linear Programming (ILP) model and developed to generate optimal solutions. **Second**, innovative multi-objective decision-making metaheuristic algorithms, based on NSGAII, CRO, and PSO, are proposed to enable real-time resource adaptation with sub-optimal solutions. **Third**, proactive resource reallocation is investigated through the development of a multi-resource and multi-step-ahead workload prediction model. By integrating the Kalman filter and support vector regression, this model accurately anticipates host resource utilization, including CPU, memory, and bandwidth. **Fourth**, building upon this predictive capability, an optimized consolidation approach is introduced, incorporating proactive host state estimation strategies for overload and underload detection. To validate the effectiveness of the proposed techniques, extensive experiments are conducted employing diverse datasets such as Planetlab, Materna, and Bitbrains, coupled with the Cloudsim simulator. The experimental results demonstrate the potential of these techniques to enhance resource management in virtualized environments.

Keywords: resource management and reallocation, resource scaling and migration, network function virtualization, cloud computing, meta-heuristics, genetic algorithm NSGAII, chemical reaction optimization CRO, Particle swarm optimization PSO, ILP model, Service chains, workload prediction, workload consolidation, Support vector regression SVR, Kalman filter, Cloudsim.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

ETS	École de Technologie Supérieure
VM	Virtual Machine
PM	Physical Machine
IaaS	Infrastructure as a Service
NFV	Network Function Virtualization
SLA	Service Level Agreement
SLO	Service Level Objective
SFC	Service Function Chain
VNF	Virtual Network Function
NSGA II	Non-Dominated Sorting Genetic Algorithm-II
CRO	Chemical Reaction Optimization
PSO	Particle Swarm Optimization
BPSO	Binary Particle Swarm Optimization
ILP	Integer Linear Programming
NP	Non-Polynomial
QoS	Quality of Service
CPU	Central Processing Unit
MIPS	Million Instructions per second
RAM	Random Access Memory
IoT	Internet of Things
HS	Horizontal Scaling

VS	Vertical Scaling
M	Migration
OD	Overload Detection
UD	Underload Detection
HUX	Half-uniform crossover
PE	Potential Energy
KE	kinetic energy
SVR	Support Vector Regression
SVM	Support Vector Machine
WS	Window size
RBF	Radial Basis Function
MMT	Minimum Migration Time
PABFD	Power Aware Best Fit Decreasing
MAD	Median Absolute Deviation
IQR	Interquartile Range
ARIMA	Autoregressive Integrated Moving Average
NSERC	Natural Sciences and Engineering Research Council of Canada

LIST OF ALGORITHMS

Page

INTRODUCTION

0.1 Context

In recent years, the rapid advancement of virtualization technologies and cloud solutions has brought about a significant revolution in the field of computing. Virtualization, a technique that enables the creation of virtual versions of physical resources, has paved the way for the development of cloud computing, enabling the efficient utilization of resources in data centers. It has transformed the way computational resources are provisioned, deployed, and managed. By abstracting physical resources such as servers, storage, and networks into virtual entities, virtualization enables multiple virtual machines (VMs) or containers to run on a single physical server. This consolidation of resources allows for better utilization of hardware, leading to increased efficiency and cost savings. With the revolution of these technologies over the past years, cloud computing has become a paramount platform for hosting enterprise systems or infrastructures and delivering a wide range of services and applications (i.e., IoT and 5G applications) to users over Internet. As a model that delivers on-demand access to shared computing resources over the internet, cloud solutions offer scalable and elastic services, enabling organizations to quickly adapt to changing demands and scale their resources accordingly. Virtualization facilitates this dynamic allocation and provisioning of resources, enabling on-demand scalability and agility in cloud environments. However, this paradigm shift has led to a growing need for proposing and implementing efficient resource management techniques. These techniques mainly aim to enhance resource utilization, minimize energy consumption, and ensure compliance with Service Level Agreement (SLA) requirements. By effectively allocating resources to meet the demands of applications and users, resource management techniques optimize the utilization of available resources, thereby reducing wastage and enhancing overall system efficiency.

0.2 Problem statement and challenges

In virtualized environments such as cloud infrastructures, efficient resource management have become crucial in addressing the challenges posed by the rapid growth of virtualization and cloud technologies. Resource consolidation approaches, resource utilization prediction, resource scaling, and migration techniques are interrelated mechanisms that play integral roles in managing resources effectively within these environments:

1. Resource Scaling (Al-Dhuraibi, Paraiso, Djarallah, & Merle, 2018) (Singh, Gupta, Jyoti, & Nayyar, 2019) is an essential aspect of resource management in virtualized environments. It involves dynamically adjusting the allocation of resources to meet varying workload demands. Scaling techniques encompass vertical scaling (increasing or decreasing the resources allocated to individual VMs or containers) and horizontal scaling (adding or removing VMs or containers). By dynamically scaling resources, virtualized environments can adapt to workload fluctuations, ensure optimal performance, and prevent resource underutilization or contention. Resource scaling can also be interrelated with resource consolidation and resource utilization prediction, as it can allow efficient resource provisioning based on predicted resource needs and consolidated resources.
2. Resource migration techniques (Choudhary et al., 2017) (Silva Filho, Monteiro, Inácio, & Freire, 2018) play a vital role in optimizing resource allocation and balancing in virtualized environments. These techniques enable the live migration of VMs or containers across physical servers without disrupting running applications. By leveraging migration algorithms and intelligent resource placement strategies, resources can be dynamically repositioned to rebalance workloads, optimize resource utilization, and accommodate variations in resource demands. Resource migration is also interrelated with resource consolidation and resource scaling, as it allows for efficient resource utilization by rebalancing resources across servers where VM or containers can be added or removed.

3. Resource consolidation approaches (Dias, Correia, & Malheiros, 2022) (Khan, Tian, Zhou, et al., 2022) involve analyzing the characteristics and resource requirements of individual workloads (e.g., of running VMs or containers) and consolidating them onto a reduced number of physical servers. By consolidating resources, redundant hardware utilization can be minimized, and underutilized resources can be efficiently redistributed. The main objectives of resource consolidation are to optimize resource utilization, reduce energy consumption, and minimize operational costs while meeting SLA requirements.
4. Resource utilization prediction techniques (Masdari & Khoshnevis, 2020) are used to forecast future resource demands in virtualized environments. These techniques leverage historical resource usage patterns, workload profiles, and predictive algorithms to anticipate future resource requirements. By accurately predicting resource needs, resource allocation decisions (scaling and migrations) can be made proactively, avoiding potential performance bottlenecks, ensuring SLA compliance, and enabling efficient resource provisioning (Radhika & Sadasivam, 2021). Resource utilization prediction is also crucial in workload consolidation, as it helps to anticipate the server state, whether it will be overloaded or underloaded in the near-future, for optimally consolidating workloads.

Employing resource consolidation approaches, resource utilization prediction, resource scaling, and migration techniques together is crucial in providing efficient resource management in virtualized environments like the cloud. These mechanisms are highly interrelated and complementary, and their combination enables providers to optimize resource utilization, reduce energy consumption, and meet SLA requirements. Coherence between these mechanisms is essential, and their collective utilization can contribute to the ongoing evolution and advancements in cloud computing technologies. However, the complexity of creating such an efficient resource management framework in the dynamic cloud environment arises due to multiple challenges. These challenges include:

1. Complexity: The process of reallocating resources in virtualized environments becomes complex due to the huge number of physical servers, virtual machines (VMs) and/or containers involved. This challenge is considered NP-hard, meaning that finding the optimal solution becomes computationally intensive and time-consuming. (Mai et al., 2021) (Laaziz, Kara, Rabipour, Edstrom, & Lemieux, 2019)
2. Heterogeneity and Compatibility: To create an efficient resource management system, cloud service providers need to ensure that their system is compatible with a wide range of virtualization technologies (VMs, containers, etc.), operating systems, and hardware configurations. Figure 0.1 shows the architectural difference between virtualization via VMs only, containers only and by combining both (VM hosting containers). This entails find generic and adaptive solutions able to manage resources across these different platforms. This can be challenging due to the complex and heterogeneous nature of modern data center environments.

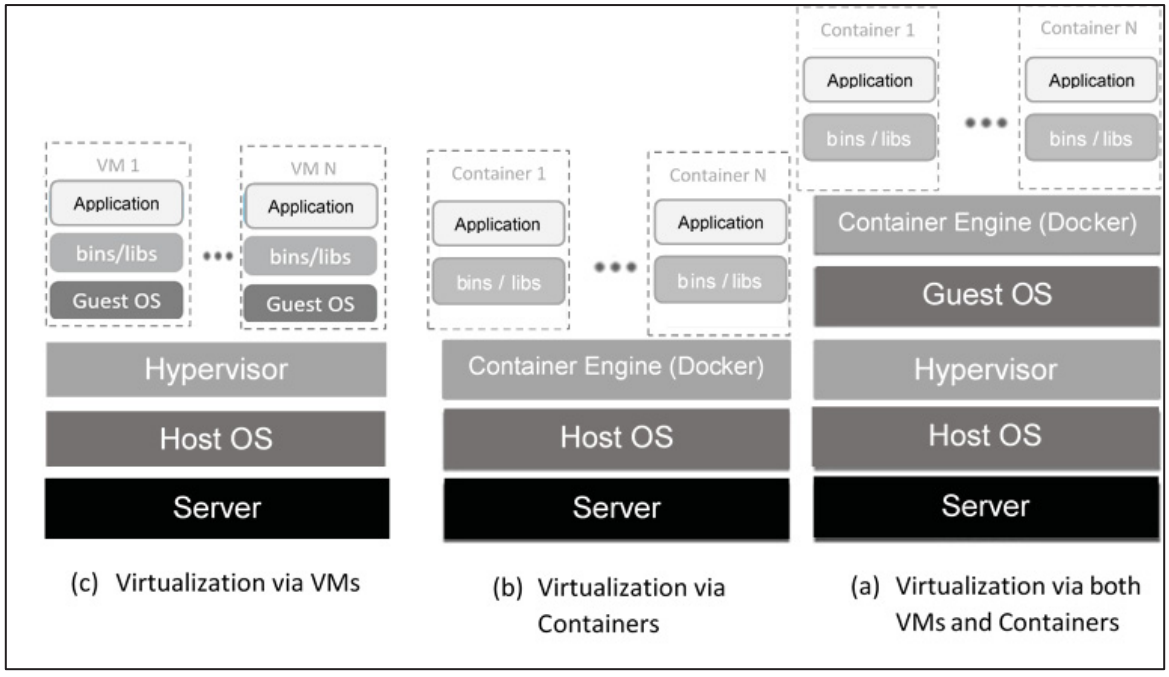

Figure 0.1 Virtualization technologies

3. Dynamic Workloads and SLA management: To meet SLA requirements and respond quickly to changes in demand, cloud service providers need to be able to manage resources in real-time. This means that the resource management system needs to be able to dynamically adjust resource allocation in real-time based on workload patterns,

user demand, and other factors. The main challenge is to maintain optimal resource utilization and meet SLA requirements under different and dynamic workloads.

4. Elasticity: Cloud environments are designed to be elastic, meaning that resources can be scaled up or down vertically or horizontally or migrated as demand changes. An efficient resource management system should be capable of selecting the optimal adaptation method for each scenario that can meet running application demands while minimizing resource waste.
5. Applications diversity: Cloud environments host a diverse set of applications or services with varying resource demands and characteristics. This diversity poses challenges in developing a resource reallocation framework that can accommodate these heterogeneous needs effectively. Additionally, in Network Function Virtualization (NFV)-enabled networks, an application involves a service function chain (SFC) of an interconnected set of virtual network functions (VNFs). These service function chains (SFCs) may vary in size and have different topologies, such as linear or non-linear forwarding graphs, introducing further challenges, constraints, and costs for their resource management. Figure 0.2 represents some examples of SFC topologies. Moreover, an SFC may have special connection constraints between its VNFs (e.g., affinity and anti-affinity constraints) that restrict their placement location and resource reallocation decisions. The resource management system should account for the diverse requirements of different applications or SFCs, optimizing resource allocation to ensure optimal performance and resource utilization for each service.
6. Conflicting objectives: Different cloud providers may have varying priorities and optimization goals. Even within the same provider, there may be a need to balance conflicting objectives and constraints. For example, service providers may aim to minimize resource utilization and energy consumption while meeting Quality of Service (QoS) requirements specified in SLAs. Designing a resource management system that can effectively address these conflicting objectives requires sophisticated algorithms and optimization techniques that consider multiple criteria and trade-offs.
7. Interdependent resource management techniques: Resource management techniques such as workload consolidation, resource scaling, resource migration, resource

utilization prediction, and monitoring are interdependent and need to be carefully coordinated and optimized to achieve the desired performance objectives. However, coordinating these techniques can be difficult due to the complex interdependencies between them. For example, a workload consolidation approach aimed at minimizing the number of active servers may inadvertently result in resource underutilization, necessitating resource scaling or migration to maintain performance levels. Balancing these interdependencies and optimizing resource management strategies in a dynamically changing cloud environment requires advanced algorithms, intelligent decision-making mechanisms, and effective coordination between different resource management techniques. This complexity makes it challenging to develop an efficient and effective resource management system. This challenge is not addressed in current implementations because our resource consolidation and adaptation strategies work separately to manage resources. However, this challenge is one of our future works.

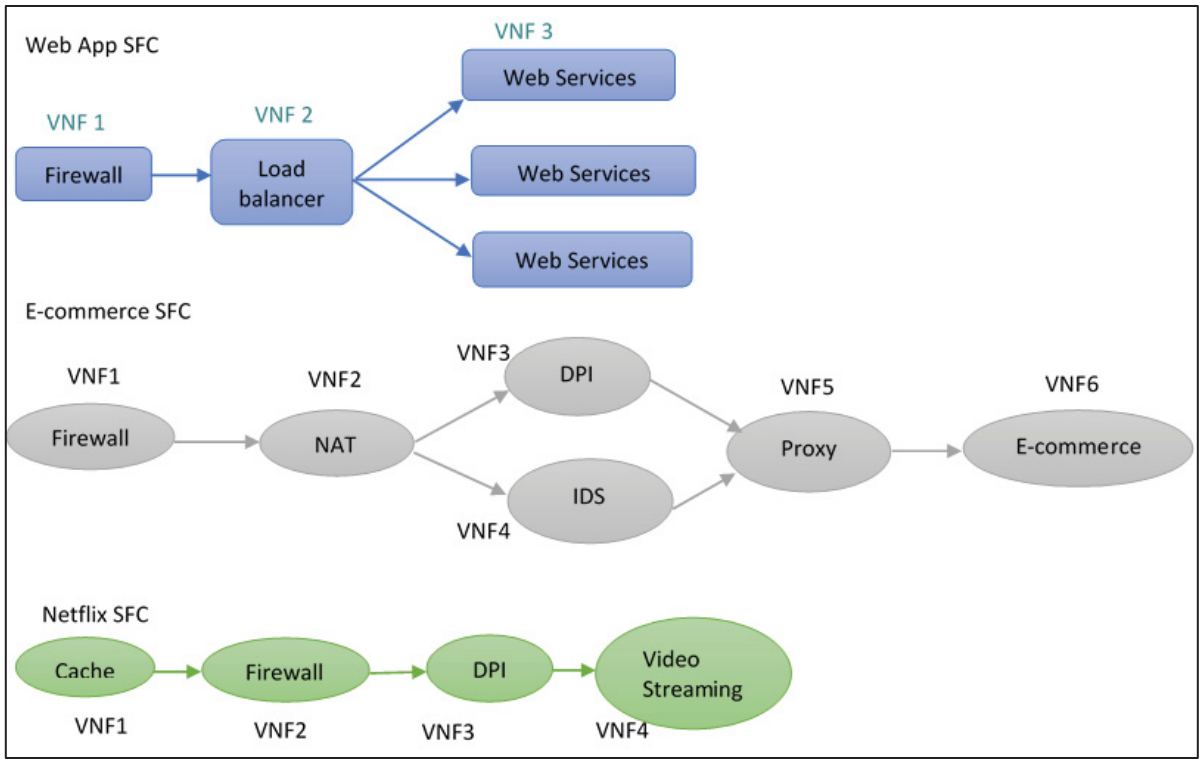

Figure 0.2 Examples of service function chains

Despite these challenges, the importance of designing an efficient resource management system for the cloud cannot be overstated. Cloud computing has become an integral part of modern computing, and efficient resource management is critical for ensuring optimal performance, reducing energy consumption, and meeting SLA requirements. Furthermore, with the increasing adoption of cloud computing, the need for energy-efficient resource management becomes more pressing, as the energy consumption of cloud data centers has a significant impact on the environment. Hence, there is a need for continued research on this topic to develop more efficient and sustainable resource management systems for the cloud.

0.3 Research Objectives

The primary aim of this thesis is to develop innovative techniques to manage resources dynamically and efficiently in virtualized and distributed environments. The research objective is to propose novel solutions that automate resource adaptation, ensuring compliance with Service Level Agreements (SLAs) while minimizing energy costs and resource consumption. To address this research problem, we approach it from different perspectives, introducing: i) efficient decision-making algorithms for dynamic adaptation of resources for SFCs requirements; ii) resource utilization prediction techniques to anticipate future resource demands and proactively reallocate the resources; iii) resource consolidation methods to optimize resource utilization in data centers using detection algorithms that leverage prediction techniques to proactively identify overloaded or underloaded servers, preventing SLA violations and saving energy. Each specific topic within this thesis has its own defined objectives, which are specified in the subsequent chapters. However, the overall research objective is to propose solutions that meet the following criteria:

1. Generic techniques: The choice of our proposed techniques and algorithms was motivated by the desire to provide portable and adaptable approaches suitable for deployment in diverse environments and compatible with various systems. They are intended to be independent of the workload, application types, virtualization technologies (such as VMs or containers), and host configurations, etc. All research

proposals put forth in this thesis have a generic nature, in the sense that these techniques can be adapted to be used in different virtualized systems or data centre architecture (e.g., SFCs, microservices, VMs or containerized applications, etc.)

2. Dynamic: the developed techniques possess the capability to dynamically adjust resource allocation, according to workload fluctuations. We tested our reactive and proactive techniques using randomized workloads (Chapter 2) and real-world datasets (Chapter 3 and 4) and validated the ability of these algorithms to dynamically make resource management decisions required to meet those workload requirements and update the infrastructure accordingly.
3. Proactive: by integrating reliable resource utilization prediction techniques, the proposed resource reallocation mechanism can proactively anticipate future demands and issues, allowing for proactive resource reallocation.
4. Scalable: As cloud computing environments continue to grow in size and complexity, a significant research objective is to address scalability issues associated with resource management. This involves developing techniques that can handle large-scale virtualized environments while maintaining performance, efficiency, and reliability. In chapters 3 and 4, we have tested our proposed mechanisms on infrastructures of up to 800 servers and up to 1500 VMs.
5. Multi-objective: Our key research objectives focus on reducing energy consumption, meeting SLA requirements, and optimizing resource utilization in virtualized environments. The proposed techniques aim to achieve a balance among these objectives, allowing for energy-efficient resource allocation while satisfying SLA commitments and maximizing resource utilization.
6. Time-efficient: The proposed algorithms generate sub-optimal solutions to the research problems within a reasonable execution time. The execution time is considered as one of the performance metrics in the experiments.

By adhering to these research objectives, this thesis aims to contribute to the advancement of resource management in virtualized and distributed environments, addressing critical challenges and proposing innovative solutions for efficient resource adaptation.

0.4 Contributions

Resource management and adaptation in virtualized environments encompass a broad and intricate field of research, encompassing various sub-topics. Within this realm, we can discern resource consolidation approaches, resource utilization prediction, resource scaling, and migration techniques as distinct strategies for resource adaptation. Each of these research topics represents a fertile ground for individual study and analysis, meriting complete theses in their own right. This thesis addresses and investigates the aforementioned topics, proposing distinct techniques that have yielded valuable insights and outcomes. Our contributions, as outlined in the subsequent chapters, have been distilled and succinctly summarized in this section.

Our investigations into resource scaling and migration techniques have led to the development of innovative methods for dynamically adjusting the resource allocation in a virtualized environment based on workload demands. For instance, our research work titled "SLO-aware dynamic self-adaptation of resources" (chapter 2), tackles the complex problem of dynamic resource adaptation in NFV-cloud environments. Dynamic resource management in NFV-cloud settings poses challenges, given the variability of workloads, the diversity of applications or service chains (SFCs), and the need to pick the appropriate method among horizontal scaling (HS), vertical scaling (VS), and migration (M) for adapting VNF resources, and to balance conflicting optimization goals.

While vertical scaling is limited by the capacities of physical machines, horizontally scaling all instances or migrating them can result in high operational costs. Existing research often focuses on one adaptation mechanism, neglecting the full range of possibilities. One of our key contributions lies in our innovative formulation of the problem, which adopts a novel and unique perspective, taking into account all three adaptation strategies, their associated costs, and subsequently determining the most appropriate approach for each given scenario.

To solve the resource allocation problem, we employ an Integer Linear Programming (ILP) model, which provides the optimal solution. However, the computational complexity of the ILP model is time-consuming. As a result, several decision-making metaheuristic algorithms are proposed based on Non-dominated Sorting Genetic Algorithm (NSGAII), Chemical Reaction Optimization (CRO), and Binary Particle Swarm Optimization (NBPSO). These algorithms offer efficient and effective solutions, enabling real-time resource adaptation decisions to manage resources of Service Function Chains (SFCs) based on real-time demands and performance requirements.

Moreover, we emphasize the importance of balancing conflicting optimization objectives in resource adaptation by integrating multiple objectives, including meeting Service Level Objectives (SLOs), optimizing resource utilization, and reducing energy consumption. SLO, a vital component of an SLA, comprises specific QoS measurements and constraints. In addition, our proposal addresses the variability of SFCs by considering different SFC sizes, and both linear and non-linear SFC topologies. The proposed algorithms are extensively evaluated through experiments conducted on various scenarios. The results demonstrate the effectiveness of the metaheuristic techniques in reducing SLO latency while approximating optimal solutions in terms of resource utilization and energy consumption.

By meticulously examining proactive resource reallocation approaches, we have also explored resource utilization prediction techniques. In our research work titled "Utilization Prediction-based VM Consolidation Approach", we develop a multi-step-ahead workload prediction model called K-SVR, which combines the power of Kalman filter and support vector regression (SVR). By integrating Kalman filter for data pre-processing, we achieve improved accuracy in predicting host CPU utilization and estimating their states. The primary objective of this research part is to overcome the limitations of existing approaches that solely rely on real-time workload variations to adapt resources and take the related decisions (Songara & Jain, 2023) (Xiao, Hu, & Li, 2019). These approaches often result in unreliable resource adaptation decisions, leading to energy waste, performance degradation, and violations of service-level agreements (SLAs). Table 0.1 represents some prediction errors of our technique

under different Planetlab workloads. It indicates that the prediction error is less than 10%. It can be improved by combining an adaptive window size technique that can handle variations in traffic loads. Such a technique can be considered as future work.

Furthermore, our research delves into the domain of workload consolidation strategies, shedding light on overload and underload detection techniques to accurately estimate the host state and efficiently trigger reliable migration decisions. Migration techniques facilitate the movement of virtualized workloads across physical hosts in order to consolidate the resources. Overload (OD) and underload detection (UD) algorithms enable migrations from overloaded servers to meet SLA requirements and migrations from under-utilized servers to conserve energy. Combining K-SVR prediction model with the proposed OD and UD algorithms, we build a predictive workload consolidation approach. Our consolidation framework dynamically determines overloaded and underloaded hosts by considering both current and near-future resource utilization. The main objective is to ensure reliable decision-making, avoiding unnecessary VM migrations and associated costs. Moreover, we have implemented an alternative consolidation approach employing an Autoregressive Integrated Moving Average (ARIMA) prediction model, replacing the K-SVR model. To evaluate the effectiveness of this approach, we have conducted simulations using real-world PlanetLab workloads on the well-known Cloudsim simulation platform. The evaluation is focused on essential metrics such as SLA violation rates, the number of VM migrations, and energy consumption in the data center. Compared to original and modified versions of benchmark algorithms (local regression, static threshold, Mean Absolute Deviation, Interquartile Range based consolidation approaches) and to ARIMA-based approach, the proposed consolidation technique exhibits a substantial reduction in SLA violations, VM migrations, and energy consumption. Besides the performed experiments, a detailed time complexity analysis for the entire framework is provided, and an analysis study is carried out on the energy consumption resulting from the execution of our proposed algorithms.

Lastly, building upon the previous work on "Utilization Prediction-based VM Consolidation Approach", the research titled " multi-resource predictive workload consolidation approach in

virtualized environments" aims to optimize and enhance the proposed consolidation technique by introducing a multi-resource and multi-step resource utilization prediction model. The optimized version of K-SVR model, called MSPR, forecasts the future workload of servers, taking into account CPU, memory, bandwidth received, and bandwidth transmitted. One limitation of our previous work, and of some existing consolidation schemes, is that they often consider only one type of resource, such as CPU utilization, while making decisions about server states. However, with the diversity of user applications and their variable workloads, this approach may not be efficient. Different applications may have different resource requirements, necessitating the consideration of multiple resource types for accurate decision-making. By considering a broader range of resources, the proposed approach becomes more versatile and applicable to various types of applications and workloads.

In this last work, the OD-MSPR and UD-MSPR algorithms consider all types of resources when making decisions. This adaptation ensures that the detection algorithms accurately assess the overall host state based on a combination of resource utilizations. Additionally, the previous static threshold for overload detection has been replaced with adaptive thresholds for each resource type, enabling more dynamic and responsive decision-making. This approach allows for distinct underload thresholds and prediction window sizes to be specified for each resource type. By accommodating this flexibility, we ensure adaptability to different scenarios and requirements. We have also refined our objective metrics (energy consumption and SLA violation) to encompass all considered resource types. To establish a fair and rigorous evaluation, we have updated and optimized ARIMA-based predictive consolidation technique and Cloudsim benchmark consolidation algorithms to incorporate multi-resource aspects. In our experimentation, we have used two real-world datasets, Materna and bitbrains, in place of the previously employed planetlab dataset.

The diagram depicted in Figure 0.3 illustrates the current interactions among the implemented entities in this thesis for resource management. For data collection, we have used existing real-world datasets and random workloads. Details on the potential enhancements of the proposed framework are given in section conclusion and future works. Overall, the culmination of all

these research efforts has resulted in the production of three distinct journal papers, each representing a significant contribution to the field of resource adaptation and optimization in

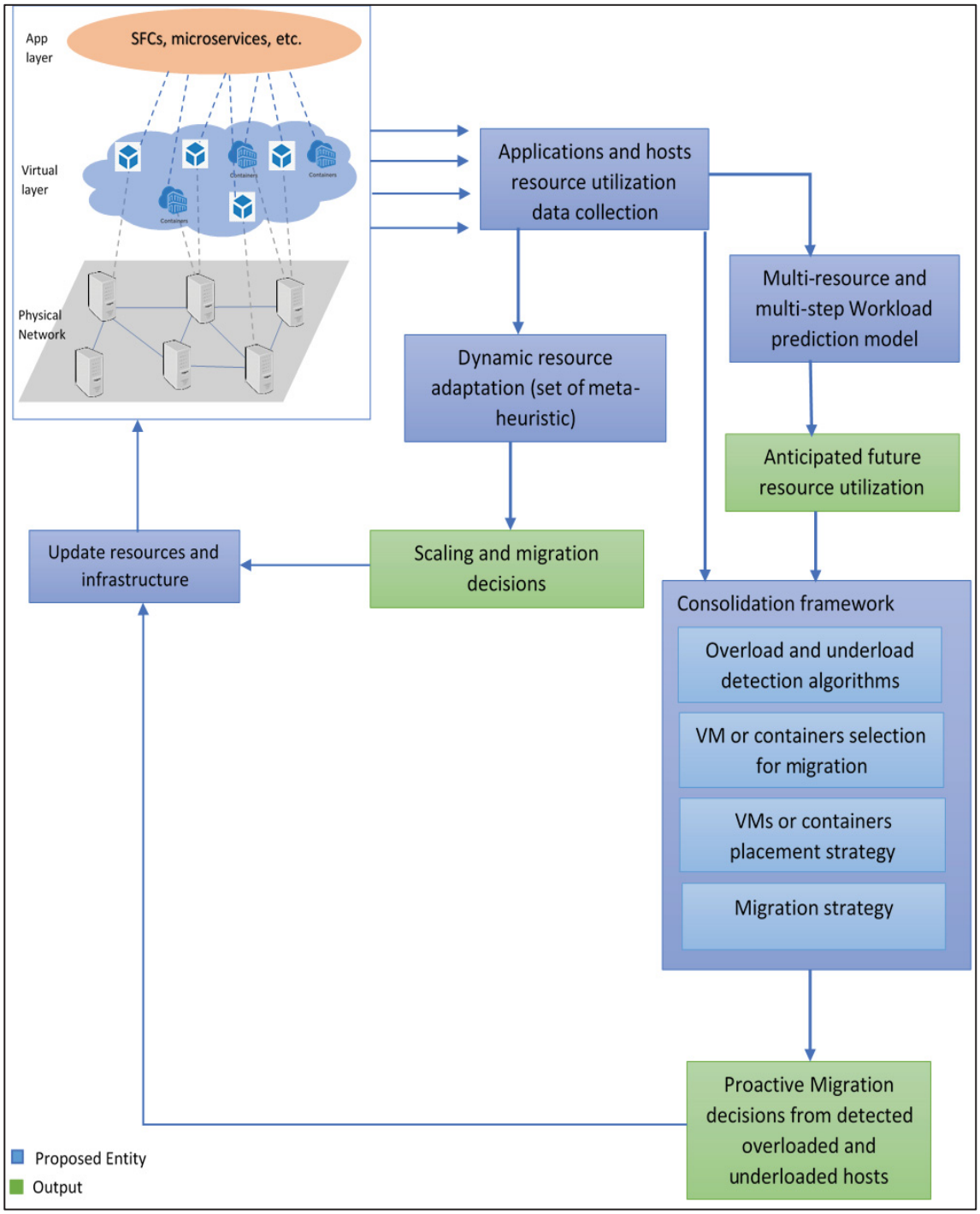

Figure 0.3 Architectural diagram illustrating the resource management components and their interactions

virtualized environments. The subsequent chapters delve into the specifics of our findings and the techniques employed, highlighting the significance and impact of our research contribution and the potential future optimizations.

Table 0.1 Prediction errors under different Planetlab workloads

Workload	**MAPE**	**MAE**	**MSE**	**MFE**	**Execution Time (ms)**
20110303	6.643	0.621	1.215	-0.053	13.617
20110306	7.016	0.614	1.227	-0.051	15.073
20110309	7.693	0.667	1.304	-0.076	17.982
20110322	7.329	0.571	0.955	-0.054	15.093
20110325	6.783	0.596	1.047	-0.047	17.789
20110403	6.555	0.654	1.258	-0.039	19.704
20110409	7.176	0.659	1.311	-0.047	13.575
20110411	6.837	0.667	1.335	-0.047	16.003
20110412	6.911	0.663	1.278	-0.054	19.649
20110420	8.028	0.768	1.780	-0.042	22.989

**Mean Absolute Percentage Error (MAPE); Mean Squared Error (MSE); Mean Absolute Error (MAE); Mean Forecast Error (MFE)

0.5 Methodology

The methodology pursued to perform this research work involves the following steps:

1. Literature review: This step involves conducting a comprehensive review of existing literature, research papers, books, and other relevant sources to gain a deep understanding of the research topic. The literature review helps identify gaps in knowledge, establish the current state of research, and gather insights that inform subsequent steps.
2. Problem formulation: Based on the findings from the literature review, the research problem or research question is formulated. This step involves formulating a mathematical model by clearly defining the objective functions and the constraints of

the study. The problem formulation stage ensures that the research is focused and addresses a specific issue or gap in the field.

3. Data collection: Once the research problem is defined, the next step is to collect relevant data. The data collected should be appropriate and sufficient to address the research problem and support the subsequent analysis. In this work, random workloads on resource utilization are generated, and existing real-world datasets are used. Three datasets are utilized to evaluate the proposed algorithms, namely, Planetlab, Materna and Bitbrains datasets.
4. Data preprocessing when needed: The collected data is preprocessed to ensure its quality and suitability for analysis. This step involves cleaning the data by removing duplicates, errors, missing values, or irrelevant information. Data preprocessing also involves normalizing data to make it consistent and ready for analysis (chapter4).
5. Algorithms Implementation: In this step, the proposed algorithms or techniques are developed to solve the research problem. The implementation stage is crucial for generating insights and results. In the context of this thesis, several resource adaptation algorithms have been proposed and implemented to address the research problem. These algorithms include:
    a. A set of meta-heuristic decision-making techniques for resource reallocation in the cloud: The thesis proposes and implements NSGAII-based, CRO-based, PSO-based, and CRO-PSO-based algorithms. These meta-heuristic algorithms are designed to optimize the allocation of resources for the service function chains (SFCs) hosted in the cloud environment (chapter 2).
    b. Mathematical model implementation using Gurobi solver: In addition to the meta-heuristic algorithms, the thesis also implements a mathematical model using the Gurobi solver. This approach aims to obtain the exact solution for the addressed research problem (resource adaptation for SFCs), providing a precise evaluation and optimal comparison with the proposed techniques (chapter 2).
    c. Resource utilization prediction technique: The thesis implements a multi-resource and multi-step-ahead prediction technique for resource utilization prediction. This technique leverages the Kalman Filter algorithm and SVR to

accurately predict resource utilization, enabling proactive resource reallocation based on anticipated needs (chapters 3 and 4).

d. Predictive workload consolidation mechanism: Another algorithm implemented in the thesis is a predictive workload consolidation mechanism. This mechanism uses multi-resource historical workload data and predictive analytics to identify opportunities for workload consolidation, optimizing resource utilization and enhancing overall performance in the data center (chapters 3 and 4).

6. Evaluation and validation: The proposed resource management techniques are evaluated and validated using appropriate performance metrics to ensure that they meet the defined objectives, such as improving resource utilization, enhancing SLA guarantees, reducing energy consumption, and enhancing overall performance within a moderate execution time, etc. Cloudsim simulator is used to test some of the proposed algorithms, specifically those related to the predictive workload consolidation mechanism. Validation ensures the reliability and credibility of the results.
7. Results interpretation and comparison with benchmarks techniques: Once the evaluation is complete, the results are compared with existing benchmarks or some previous studies from the state-of-the-art to assess the novelty or improvement of the research. For example, our workload consolidation technique based on Kalman-SVR prediction model is compared to the original and modified versions of existing Cloudsim benchmarks, specifically, static-threshold-based and adaptive-thresholds-based consolidation approaches, predictive ARIMA-based consolidation technique etc. (chapters 3 and 4). Whereas, the proposed set of meta-heuristic algorithms for SFC's resource adaptations are compared to each other's performance, and to the optimal solution generated by the Gurobi solver (chapter 2).
8. Complexity and other analyses: Detailed analyses are conducted to assess various aspects of some implemented algorithms or models. One key aspect examined is the computational time complexity of the algorithms. The time complexity analysis provides insights into the efficiency and scalability of the implemented approaches. By understanding the computational time complexity, the thesis aims to evaluate the

feasibility of the algorithms for large-scale resource adaptation scenarios. Furthermore, the thesis also delves into analysis related to power consumption resulting from the execution of the proposed algorithms. This analysis focuses on quantifying the energy requirements and power consumption associated with the resource reallocation process. Understanding the power consumption implications is crucial for optimizing resource utilization while considering energy efficiency goals. Both the computational time complexity analysis and power consumption analysis contribute to a comprehensive understanding of the implemented algorithms' performance and impact. These analyses are discussed in chapters 3 and 4.

9. Optimization and fine-tuning: Based on the performed analysis and the evaluation results, areas for improvement and optimization are identified. This step involves refining the algorithms, models, or methodologies to enhance their performance, efficiency, or accuracy. Fine-tuning may include parameters tuning, mathematical model adjustments by considering more objective metrics or constraints, or further optimization of the proposed techniques to consider missing aspects. Chapter 4 illustrates optimizations or enhancements of the techniques proposed in chapter 3.

Each step in the research methodology, discussed above, contributes to the overall process of conducting this thesis work and ensuring its validity, reliability, and contribution to this research field.

0.6 Publications

The research contributions discussed in this thesis are either published or submitted as follows:

1. **Mirna Awad** et al. (2022). SLO-aware dynamic self-adaptation of resources. Future Generation Computer Systems, 133, 266-280. **(IF-7.5)**
   This work has been done in collaboration with Ericsson Canada.
2. **Mirna Awad** et al. (2022). Utilization prediction-based VM consolidation approach. Journal of Parallel and Distributed Computing, 170, 24-38. **(IF-4.542)**

3. **Mirna Awad** et al. (2023). Multi-resource predictive workload consolidation approach in virtualized environments. Journal of Computer Networks. Journal Article under review **(IF-5.6)**

0.7 Thesis Organization

This work is presented in the form of a thesis by articles. An introduction showing the motivations behind this research subject and a general literature review are detailed in the first two chapters. Then, each journal article is presented in a dedicated chapter. Finally, the main insights are summarized, and some future works are suggested.

CHAPTER 1

LITERATURE REVIEW

This chapter aims to present and emphasize the newest approaches addressing the dynamic adaptation of resources in virtualized environments. It is divided into three parts: the first part discusses solutions proposed for resource adaptation techniques such as scaling and migration; the second part explores workload prediction techniques; and the third part delves into resource consolidation strategies. While each journal article in the subsequent chapters provides an extensive review of the existing literature, our discussion primarily focuses on more recent research works in this field.

1.1 Resource Adaptation

Resource adaptation plays a crucial role in ensuring efficient utilization of virtualized infrastructure while meeting the varying demands of applications and services. In the context of NFV, applications consist of SFC chains that connect a set of VNFs.

Three adaptation mechanisms can be used to address the resource adaptation issue including horizontal scaling (HS), vertical scaling (VS), and migration (M). Horizontal scaling involves adjusting the number of allocated virtual resources, such as VMs or containers hosting these VNFs, to dynamically accommodate the changing workload requirements. Vertical scaling, on the other hand, focuses on modifying the computing capacity of individual VNF instances. It involves scaling up, which refers to increasing the resources (e.g., CPU, memory) allocated to a VNF, or scaling down, which involves decreasing the resources allocated to a VNF. Vertical scaling is typically limited by the capacities of the physical machines hosting the VNF instances. Migration, the third adaptation mechanism, involves the movement of VNFs or entire SFCs from one physical machine to another. Migration techniques can be categorized into different types, such as post-copy, pre-copy, or hybrid migration, depending on the approach used to transfer the VNF or SFC from the source to the destination server. While

existing research has explored various aspects of resource adaptation problem, many proposals tend to focus on specific mechanisms or prioritize one adaptation technique over others.

Chouliaras et al. (Chouliaras & Sotiriadis, 2022) present a framework called PACE (Performance-aware Auto-scaler for Cloud Elasticity) for auto-scaling containerized cloud applications. This framework includes both reactive and proactive vertical auto-scaling techniques. The reactive approach uses threshold-based scaling rules to dynamically adjust cloud resources based on predefined thresholds, preventing system failures. The proactive approach utilizes convolutional neural networks (CNN) for time series forecasting and K-means for clustering. This approach clusters future workload demands into High, Medium, and Low categories and generates elastic scaling policies accordingly. However, this work addresses solely vertical scaling method.

Rahman et al. (Rahman, Ahmed, Huynh, Tornatore, & Mukherjee, 2018) propose a proactive machine learning approach for auto-scaling VNFs to improve QoS and reduce costs. They convert the auto-scaling problem into a supervised ML classification problem, training a classifier with past scaling decisions and network load data. The classifier predicts the number of VNF instances required to serve the traffic while meeting QoS requirements. The study compares four virtualization technologies (Xen, KVM, Docker, and LXC) and analyzes their impact on auto-scaling performance. However, their work focuses solely on horizontal scaling of VNFs.

Some studies have incorporated a combination of vertical and horizontal scaling to create a hybrid auto-scaling mechanism. Jeong et al. (Jeong, Baek, Park, Jeon, & Jeong, 2023) present an approach called Proactive Hybrid Pod Autoscaling (ProHPA) as a solution to pod autoscaling in cloud computing. ProHPA utilizes a bidirectional long short-term memory (Bi-LSTM) model with attention mechanism to forecast future CPU and memory usage. Then, based on these forecasted resource usages, ProHPA sequentially performs three steps: reducing excessive resource usage with vertical pod autoscaling (ReVPA), preventing overload with horizontal pod autoscaling (HPA) (PoHPA), and adjusting the initial resource allocation. The

evaluation shows that ProHPA significantly improves CPU and memory utilization compared to conventional HPA provided by Kubernetes. Other researchers concentrate on virtual resources migration or re-placement problem without considering the full spectrum of adaptation methods (T. Z. He, Toosi, & Buyya, 2021) (Duong-Ba, Nguyen, Bose, & Tran, 2018). Consequently, a holistic approach that takes into account all adaptation techniques (HS, VS, and M) is necessary to cover the diverse resource adaptation scenarios that can arise.

Other significant limitations of existing techniques are adapting the resources of individual VNFs without considering their connectivity or assuming them connected in a linear chain. SFCs can exhibit different topologies, including both linear and non-linear structures. A linear SFC topology refers to a chain-like structure where each VNF in the chain is connected to at most two neighboring VNFs. This topology is characterized by a sequential arrangement of VNF instances, creating a straightforward flow of data through the chain. On the other hand, non-linear SFC topologies allow for more complex interconnections among VNFs. In this type of topology, VNFs can have multiple instances and multiple connections with other VNFs within the chain. Nadjaran Toosi et al. (Nadjaran Toosi, Son, Chi, & Buyya, 2019) introduce an auto-scaling algorithm that optimizes end-to-end latency by considering vertical and horizontal scaling, migration, and flow scheduling. The main objective is to dynamically allocate CPU resources and network bandwidth for service chains while meeting latency requirements. It first attempts to vertically scale up the resources, if this is not feasible due to limitations in available resources, it explores horizontal scaling by adding more instances of VNFs. To handle bandwidth adaptation, the algorithm utilizes flow scheduling techniques. It redirects traffic to alternative network paths that can deliver the required bandwidth while maintaining the desired latency. In cases where no suitable path is found, VNF migration is employed.

In light of the aforementioned limitations in the existing literature, our work aims to contribute to the state-of-the-art by formulating the SFC resource adaptation problem as an ILP model that explicitly considers all three adaptation techniques (HS, VS, and M). This unique formulation sets our work apart from previous approaches that have often focused on one

adaptation mechanism or neglected to differentiate between them. By incorporating decision variables for each adaptation method, our ILP model enables us to evaluate and compare the associated costs and benefits of each approach. This allows us to make informed decisions on selecting the most appropriate resource adaptation method for a given scenario. Moreover, by defining objective functions based on these decisions, we can effectively optimize the allocation of resources and identify the optimal combination of adaptation techniques needed to satisfy the resource requirements of incoming SFC requests. Furthermore, we propose novel decision-making meta-heuristic algorithms based on NSGAII, CRO, Binary PSO, and a combination of CRO and Binary PSO. Our algorithms can handle both linear and non-linear SFC topologies of various size (number of VNFs), and strive to balance multiple objectives, including energy savings, improved CPU utilization, and minimized SLO violation (end-to-end latency).

1.2 Workload prediction

The workload prediction problem is extensively addressed in the state-of-the-art for various contexts, including proactive auto-scaling (S. Luo et al., 2022) (Radhika & Sadasivam, 2021), predictive resource consolidation (Chaurasia, Kumar, Vidyarthi, Pal, & Alkhayyat, 2023) (H. Sayadnavard, Toroghi Haghighat, & Rahmani, 2022), workload modeling (St-Onge et al., 2021), anomaly detection (Benmakrelouf et al., 2020) etc. Researchers are continuously proposing new methodologies to enhance accuracy and effectiveness in workload prediction. In this section, we discuss some new proposals in this area.

Devi et al. (Devi & Valli, 2023) develop a hybrid model for predicting future CPU and memory utilization in a cloud data center. The proposed model combines ARIMA and ANN (Artificial Neural Network) techniques to forecast both linear and nonlinear components of CPU and memory utilization patterns. The ARIMA model detects linear components in workload patterns, while the ANN leverages residuals derived from ARIMA model to capture nonlinear components. Dogani et al. (Dogani, Khunjush, & Seydali, 2023) aim to improve host workload prediction in cloud computing by proposing a hybrid approach that combines Discrete Wavelet

Transformation (DWT), Bidirectional Gated-Recurrent Unit (BiGRU), and an attention mechanism. The DWT is used to decompose the data into sub-bands, allowing for the extraction of patterns from nonlinear and nonstationary data. The decomposed data is then fed into a BiGRU model, enhanced by an attention mechanism to capture temporal correlation features. The study specifically focuses on predicting CPU usage. In the research work conducted by Malik et al. (Malik, Tahir, Sardaraz, & Alourani, 2022), a novel approach for predicting multi-resource utilization based on the Functional Link Neural Network (FLNN) is introduced. To enhance the prediction accuracy, the researchers develop a hybrid model that combines the genetic algorithm (GA) with the particle swarm optimization (PSO) algorithm to train the neural network. The fitness function for the GA is determined as the Mean Absolute Error (MAE). The experimental analysis primarily focuses on CPU and memory utilization of virtual machines (VMs). St-Onge et al. (St-Onge et al., 2021) propose a hybrid approach for workload modeling in cloud environments, aiming to generate generic CPU workload models that can fit various workload domains. The authors present two approaches: one combining Hull-White modeling with a genetic algorithm, and another combining a SVR model with Kalman filter. Janjanam et al. (Janjanam, Siram, & Kollu, 2023) utilize a SVR model combined with M/M/c queuing model, to predict the workload of web servers based on historical data. The Last Value model, Moving Average model, and Auto Regression model are compared with SVR models using different kernels. The obtained results conclude that the SVR-based models, especially those with RBF kernel, are better at forecasting server workload compared to basic forecasting models.

In our work, we propose, a multi-step ahead prediction model for forecasting the utilization of server resources, encompassing CPU, memory, received bandwidth, and transmitted bandwidth. We predict the future trend of each resource type instead of a single future value. By capturing the trends, we provide valuable insights into how the utilization of these resources is expected to evolve over time. Our model leverages a combination of SVR and Kalman Filter algorithms to accurately forecast future resource utilization. By integrating the Kalman Filter as a pre-processing step, we improve the accuracy of SVR predictions. The motivation behind our approach stems from the workload modeling research work mentioned earlier (St-Onge et

al., 2021), where the combination of SVR and Kalman Filter has shown promise. However, we target a different research context: estimating the state of hosts and optimizing resource consolidation in virtualized environments. Notably, our model predicts multiple resources simultaneously, and it is suitable for a variety of systems such as servers, virtual machines (VMs), and containers.

1.3 Resource Consolidation

Resource consolidation is a vital strategy for optimizing resource allocation and achieving energy efficiency in cloud environments. Its objective is to host workloads onto a reduced number of physical machines, maximizing resource utilization and minimizing energy consumption (Panwar, Rauthan, & Barthwal, 2022) (Bharany et al., 2022). Researchers have explored various approaches to effectively address this problem. These approaches include algorithms for VM or container replacement, techniques for detecting overloading and underloading states, and strategies for selecting VMs or containers for migration. The detection of host overloading and underloading conditions significantly impacts the performance and efficiency of the consolidation system. Overloaded hosts may suffer from resource scarcity, leading to degraded performance and potential violations of SLA. On the other hand, underloaded hosts indicate the underutilization of resources, resulting in resource wastage and unnecessary costs.

Some State-of-the-art approaches rely on actual resource utilization data to assess the current state of hosts, determining if they are overloaded or underloaded. Songara et al. (Songara & Jain, 2023) propose a multi-resource VM consolidation approach called MRA-VC. The underloaded hosts are classified into different categories: severe load, moderate load, or low load based on their current multi-resource utilization score and predefined thresholds. The overload detection algorithm assigns dynamic weights to each resource based on their importance in the decision-making process. If the calculated weighted score exceeds an upper threshold (80%), the host is considered overloaded. Regarding VM selection and placement, the authors propose a modified VM selection and placement algorithm based on a particle

swarm optimization. Yadav et al. (Yadav, Zhang, Li, Liu, & Laghari, 2021) suggest GradCent, an algorithm based on Stochastic Gradient Descent technique, for detecting overloaded hosts in cloud data centers. It determines an upper CPU utilization threshold based on CPU utilization history. They also introduce the Minimum Size Utilization (MSU) Algorithm, which prioritizes VMs with high CPU utilization and small sizes for migration from overloaded hosts. Hariharan et al. (Hariharan, Siva, Kaliraj, & Prakash, 2023) develop an adaptive beetle swarm optimization (ABSO) algorithm that combines the strengths of particle swarm optimization and beetle swarm optimization to optimize the placement and consolidation of virtual machines in a cloud environment. The fitness function considers energy consumption, migration cost, and utilization metrics.

In contrast, other approaches leverage the power of predictive models in resource consolidation. These models utilize historical data and machine learning algorithms to forecast whether a server is likely to encounter overloading or underloading conditions in the near future. By predicting future resource demands, these models enable proactive decision-making, helping to prevent unnecessary migrations and optimize resource allocation. Sayadnavard et al. (H. Sayadnavard et al., 2022) present a multi-objective approach for dynamic VM consolidation in cloud data centers. Their approach combines Discrete Time Markov Chain (DTMC) and Continuous Time Markov Chain (CTMC) for PM categorization, employs a heuristic VM selection algorithm based on task completion time, and utilizes a ε-dominance-based multi-objective artificial bee colony algorithm for VM placement. The proposed approach aims to reduce energy consumption, improve system reliability, and minimize resource wastage. Chaurasia et al. (Chaurasia et al., 2023) also utilize the Markov chain principle for server transition to optimize the consolidation process. Banerjee et al. (Banerjee, Roy, & Khatua, 2021) present a framework for efficient resource utilization in cloud environments by utilizing a multi-step-ahead workload prediction technique. The framework encompasses three key components: workload characterization, where agglomerative hierarchical clustering is used to identify VMs with similar resource usage patterns; workload prediction, employing and comparing a set of supervised machine learning models such as linear regression, k-nearest neighbor, decision tree, support vector machine, and gradient

boosting to forecast future resource consumption (CPU and memory); and VM placement, which utilizes a modified Best Fit Decreasing algorithm to allocate VMs based on predicted resource consumption values. Farahnakian et al. (Farahnakian et al., 2019) present a utilization prediction-aware VM consolidation approach in cloud data centers called UP-VMC. The problem is formulated as bi-dimensional vector packing problem as two types of resource are considered: CPU and memory. Two regression-based prediction models are used for resource utilization prediction: Linear Regression (LR) and K-Nearest Neighbor Regression (K-NNR). A PM is considered as overloaded if its current or its predicted CPU or memory utilization exceeds its resource capacity. Their approach identifies the underloaded PM by comparing the current load of the PMs and selecting the PM with the lowest load.

In our research, we propose a novel resource consolidation mechanism that combines the strengths of actual resource utilization analysis and predictive modeling. Building upon our MSPR prediction model discussed in the previous sub-section, our consolidation approach leverages both the current resource utilization and the predicted utilization trends to accurately identify overloading and underloading states in hosts. It considers multiple resource types, such as CPU, memory, and bandwidth, when making migration decisions. For overload detection, we calculate adaptive MAD thresholds that are specifically tailored to each resource type. Additionally, our approach offers flexibility by allowing the specification of distinct underload thresholds and prediction window sizes for each resource type. This adaptability allows for fine-grained customization to match the specific requirements of different resources. In addition to our primary approach utilizing the MSPR model, we implement an alternative consolidation approach that utilizes an ARIMA multi-resource prediction model. This alternative model serves as a replacement for the MSPR model, providing a comparative basis for evaluating the performance of our proposed mechanism. To evaluate the efficacy of our approach, we conduct extensive experiments comparing the MSPR-based consolidation approach with the ARIMA-based approach, as well as modified versions of benchmark consolidation algorithms in Cloudsim.

CHAPTER 2

SLO-AWARE DYNAMIC SELF-ADAPTATION OF RESOURCES

Mirna Awad [a], Nadjia Kara [a], and Claes Edstrom [b]

[a] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[b] Ericsson Canada,
8275 route Transcanadienne, Saint-Laurent, Québec, Canada, H4S 0B6

Published in *Journal of Future Generation Computer Systems*, March 2022.
(https://doi.org/10.1016/j.future.2022.03.018)

2.1 Abstract

Cloud computing and Network Function Virtualization (NFV) are two complementary technologies. Virtual network functions (VNFs) provided by NFV are connected in the form of service function chains (SFCs) and typically hosted on the cloud. Dynamic resource adaptation in NFV-cloud settings remains a challenging research problem. VNF resources can be adapted by performing either vertical scaling (VS), horizontal scaling (HS), or Migration (M). Deciding on the optimum strategy among these three approaches (VS, HS, M) may face several challenges, including the dynamicity of the cloud environment; the sheer multiplicity of SFC topologies (e.g., linear, or non-linear SFCs); potentially conflicting optimization objectives, and the substrate network configuration. Considering the challenges introduced, we propose decision-making algorithms that make the best adaptation decisions for the SFCs dynamically, while balancing a set of cost functions, such as energy consumption, resource utilization, and Service Level Objective (SLO) violation. We first formulate the problem as an integer linear programming (ILP) model to compute the optimal solution. Then, because solving an ILP model is time-consuming, we adopt multi-objective metaheuristic algorithms based on Non-dominated Sorting Genetic Algorithm (NSGAII), Chemical Reaction Optimization (CRO), Binary Particle Swarm Optimization (NBPSO), and the combination CRO-NBPSO to solve this problem. Experimental results demonstrate the effectiveness of the

proposed meta-heuristic algorithms in reducing the end-to-end latency while achieving performance similar to optimal solutions in terms of resource utilization and energy consumption.

Keywords: Dynamic resource adaptation, NFV, service chain, CRO, NSGAII, Binary PSO.

2.2 Introduction

Today, cloud computing has become an essential need and a popular technology both in academia and in industry. Among the main purposes of cloud platforms is the delivery of computing resources on-demand on a pay-for-use basis. The pay-for-use pricing model allows the user to only pay for the resource used. On-demand resource provisioning and release boil down to the ability to provide resources dynamically according to the application's needs (Al-Dhuraibi et al., 2018). Coupled with the pricing model offered, elastic resources constitute one of the main attractive benefits of cloud computing. On the other hand, the Network Function Virtualization (NFV) (Yi, Wang, Li, Das, & Huang, 2018) provides a wide range of network functions as virtual software components typically hosted on virtual machines (VM) or containers in the cloud, instead of on traditional hardware components. The ordered interconnection of these virtual network functions (VNFs) forms a service function chain (SFC) for each specific application (e.g., IoT-based applications, 5G applications) (Medhat et al., 2016). For example, a web application may consist of at least three connected VNF types, such as a load balancer, web services, and a database. Despite the cost savings and flexibility provided by the VNFs, several research problems related to dynamic resource management in NFV-cloud settings still need to be addressed. Indeed, the performance of running applications is mainly impacted by the dynamicity of the cloud environment (e.g., variability of workloads, diversity of applications, etc.). Having an automated system to dynamically assign cloud resources for the VNFs in real time according to the workload fluctuation still faces numerous challenges. A wrong resource allocation decision may lead to an over-provisioning state, leading to extra costs being paid to rent unnecessary amounts of resources exceeding application needs. Conversely, an under-provisioning state may degrade the service

performance due to a lack of resources required to process incoming and ongoing requests within a reasonable timeframe. Moreover, a dynamic resource adaptation system does not only maintain the service performance and decrease the expenses of its deployment on the cloud, but also by releasing unused cloud resources the providers can save energy consumption or can increase their revenue by using these resources to serve new requests.

One of the challenges in building such a system is to choose the appropriate method to adapt the VNF resources at each moment. Three possible mechanisms can be used to adapt resources for a VNF in the cloud: Horizontal scaling (HS), Vertical scaling (VS), and Migration (M). Horizontal scaling consists in scaling in (removing VNF instances) or out (adding new VNF instances). Vertical scaling consists in scaling up (increasing the VNF's computing capacity, such as the CPU) or down (decreasing the capacity of a VNF). Migration involves moving a VNF or an SFC from the current hosting server to another one. Relying on only one of these adaptation mechanisms will lead to a non-efficient resource adaptation technique. Vertical scaling is limited to the capacities of the physical machines, while horizontally scaling all the instances or migrating them may lead to high operational costs. Many researchers have addressed the dynamic resource allocation (Gil Herrera & Botero, 2016) issue in the cloud. However, some proposals prioritize one adaptation mechanism over another. For instance, they utilize vertical scaling as much as possible. Then, they apply horizontal scaling when the vertical one becomes impossible. Others focus on migration or re-placement of the VNF where the decision is either to migrate the VNF to another host or keep it in place. The re-placement decisions are most often made without considering other types of adaptation methods (VS and HS). Consequently, we need a decision-making approach that takes the three possible resource adaptation techniques (HS, VS, M) into account to cover various resource adaptation scenarios while selecting the best one for each of them. Additionally, deciding on the optimum strategy among those three mechanisms is not obvious because of potentially conflicting resource optimization goals. For instance, the service provider may need to minimize resource utilization and energy consumption, while providing the required Quality of Service (QoS) (Mostafavi, Hakami, & Sanaei, 2021) defined in the Service Level Agreement (SLA) established with the customer. Service Level Objective (SLO) is a key element of SLA

composed of one or more QoS measurements and constraints. For example, the service provider may be obliged to adhere to a maximum tolerated latency or service availability pre-specified in the SLO to avoid penalties (e.g., latency or availability of highly sensitive SFC). Another challenge is to adapt the resources of service chains of VNFs rather than individual VNFs. Each SFC may consist of a different number of VNFs connected in a linear or non-linear chain. The topology is *linear* when each VNF in the chain has only one instance and is connected to two VNFs peers at the most. Such limitations do not exist in non-linear chains, where each VNF may have many instances and multiple peers. Thus, resource adaptation requirements vary from one SFC to another. Some existing work on resource allocation treats individual VNFs without considering their full chain and connectivity, while other proposals only target linear SFC topologies. For this reason, it is important to consider the SFC topology, as well as its type (linear or nonlinear) and size (number of VNF instances), when adapting resources.

To contribute a solution to this problem, we consider the challenges introduced and propose multi-objective meta-heuristic scaling algorithms to help in automating resource adaptation for SFCs according to workload variations. The main objective of our algorithms is to select the best adaptation mechanism (VS, HS, or M) that meets the SFC's needs, while balancing a set of cost functions, such as energy consumption, SLO violation, and resource utilization. The main contributions of our work can be summarized as follows:

1. The resource allocation problem is NP-hard (Rodriguez, Alkmim, Da Fonseca, & Batista, 2017)(Houidi et al., 2017). We first formulate the problem as an Integer Linear Programming (ILP) model to find the exact solution. The solver Gurobi (http://www.gurobi.com) is used for implementation. In particular, the model takes a set of SFC requests as input and provides the optimal resource adaptation decisions to fulfill their resource needs. Contrary to existing work, in our problem formulation, we consider and distinguish the three possible adaptation strategies, including vertical elasticity, horizontal elasticity, and Migration, and calculate cost functions based on each. To the best of our knowledge, we are the first to formulate this problem from this

perspective. We also target both linear and non-linear SFC chains, of different sizes and graphs.

2. Because solving the ILP model is a time-consuming process (Rodriguez et al., 2017)(Junjie Liu Fen Zhou, Ping Lu, Zuqing Zhu, 2017)(Laaziz et al., 2019), it is more efficient to use heuristic-based techniques to solve the problem. We propose several meta-heuristic algorithms based on the multi-objective nondominated sorting genetic algorithm (NSGAII), the chemical reaction optimization (CRO) framework, the binary particle swarm optimization (NBPSO), and a combined technique CRO-NBPSO. To the best of our knowledge CRO, binary PSO and their combination have not been considered in the literature to address the discussed research problems.
3. We design all our approaches to find optimum resource adaptation decisions (VS, HS, M) for the SFC requests dynamically, irrespective of their types and their sizes, while balancing a set of objective functions composed of CPU utilization, energy consumption, and SLO violation.
4. We verify the effectiveness of our algorithms through extensive experiments on different scenarios, including different numbers of SFC requests, different SFC sizes, and considering both linear and non-linear topologies. We compare and analyze their performance in terms of mentioned objective functions, number of servers used, and their run time to find solutions. Through test experiments, our proposed meta-heuristic techniques demonstrate their ability in reducing SLO latency and prove their effectiveness in approximating the optimal solution in a much shorter runtime.

The rest of the paper is organized as follows: Section 2.3 reviews the literature on dynamic resource adaptation in the cloud and highlights the novelty of this work. Section 2.4 describes the problem formulation as an ILP model. Section 2.5 explains in detail the proposed meta-heuristic algorithms. Section 2.6 presents our experimental setup, discusses the results obtained, and compares the performance of our approaches. Finally, section 2.7 concludes this paper and summarizes the main insights.

2.3 Related work

VNF resource adaptation problem is computationally hard. Solving such complex optimization problems using exact methods such as Integer Linear/Non-linear Programming (ILP/NILP), Mixed Integer Linear/Non-linear Programming (MILP/MINLP), SMT solvers, etc. requires a long execution time, which makes these techniques unsuitable for dynamic or real-time problems. For this reason, researchers generally look instead for approximation methods to find near-optimal solutions in a shorter time. Nowadays, meta-heuristic algorithms have become a powerful choice to solve complex research problems (Siddique & Adeli, 2017) (Emmerich & Deutz, 2018)(Olivas, Valdez, Melin, Sombra, & Castillo, 2019) (Astudillo, Melin, & Castillo, 2015)(Melin, Astudillo, Castillo, Valdez, & Garcia, 2013)(Olivas, Valdez, Castillo, & Melin, 2016). Many research works have addressed the dynamic resource allocation problem in the cloud and proposed different mechanisms to solve it (Al-Dhuraibi et al., 2018)(Yang, Li, Trajanovski, Yahyapour, & Fu, 2021)(Gil Herrera & Botero, 2016)(Schardong, Nunes, & Schaeffer-Filho, 2021)(Singh et al., 2019). However, the existing contributions suffer from many limitations.

To adapt resources for a VNF in the cloud, three techniques are possible: Horizontal scaling (HS), Vertical scaling (VS), and Migration (M). Several proposals focus mainly on horizontal elasticity [(Bouabdallah, Lajmi, & Ghedira, 2016), (Santhosh & Binu, 2016), (F. Huang, Li, Yuan, & Li, 2017), (Y. Li & Xia, 2017), (Kan, 2016), (Hu, Bo, & Fuyang, 2016), (Shariffdeen, Munasinghe, Bhathiya, Bandara, & Dilum Bandara, 2016), (Meng, Rao, Zhang, & Hong, 2016), (Aslanpour & Branch, 2016), (Z. Luo & Wu, 2020), (X. Wang et al., 2016) and (Yi, Wang, & Huang, 2017)], where the number of allocated virtual resources (VMs or containers) should change dynamically according to application demands. For example, The author in (Kan, 2016) proposes DoCloud, a horizontal elasticity platform for web applications running on Docker containers. After predicting the incoming workload using the ARMA technique and estimating the number of containers needed, scale-out actions are triggered when the monitoring system detects that the resource utilization of some containers exceeds a specific threshold. They argue that scale-out actions should be fast enough to ensure QoS of web

applications, while scale-in should not be as fast to avoid oscillations in the number of containers. For this reason, the elasticity controller triggers scale-in actions only if the predicted number of containers is always less than the current number of running containers for k continuous periods. Results show that the proposed platform can dynamically allocate required resources to applications and improve container resource utilization. In (Yi et al., 2017), the authors target the scalable SFC provision problem in the sense of adding or removing functions to/from the service chains as needed. The problem is first formulated as an ILP model and then two heuristic schemes (reactive and proactive) are proposed to handle Scale-in (SI) and Scale-out (SO) requests. The reactive scheme scales the SFC without changing its Service Function Path (SFP) while the proactive scheme aims to optimize the SFP graph of SFC to a better one for the purpose of minimizing bandwidth consumption. Their objective function is to reduce the total cost including resource consumption, VNF deployment and SFC recomposition cost. Other researchers utilize only vertical elasticity to dynamically increase or decrease the resources allocated to computational units, such as the CPU, memory, storage, etc., to handle a varying workload [(Alzahrani et al., 2016), (Moghaddassian, Bannazadeh, & Leon-Garcia, 2017)]. For instance, an Energy-based Auto-scaling (EBAS) approach is presented in (Alzahrani et al., 2016) to proactively scale the number and frequency of CPU cores to containers. Their method incorporates the dynamic voltage and frequency scaling (DVFS) technique to dynamically adapt CPU frequencies, and a prediction model based on Autoregressive Integrated Moving Average (ARIMA) to anticipate future CPU utilization. The best allocation plan that has the lowest energy consumption and meets the SLA requirement (latency) is selected. In (Moghaddassian et al., 2017), the authors create VM scaling method based on the threshold-based approach using adaptive thresholds. Up and down vertical scaling thresholds are dynamically updated according to the real-time data utilization (CPU or memory) of VMs. Specifically, when the monitored data utilization of a VM is still greater than the initial threshold for β seconds, the algorithm increases the threshold value by α percent as long as the threshold doesn't attend a hundred percent. This process is repeated until the threshold can't increase anymore. At this point, the algorithm triggers the scale-up action if the metric crossed the threshold. Similarly, if the real-time data utilization of a VM is still smaller than the threshold for β seconds, the algorithm decreases the threshold by α percent

so long as the threshold is still greater than a minimum possible tolerance level (T_{min}). When the threshold can no longer decrease and the measured metric still below T_{min} for β seconds, the scale down action is triggered. Some existing works combine the two types of elasticity (VS and HS) and build a hybrid auto-scaling mechanism [(G. Huang et al., 2016), (Hirashima, Yamasaki, & Nagura, 2016), (Ye, Guangtao, Shiyou, & Minglu, 2017), (Hirashima, 2016), (Sotiriadis, Bessis, Amza, & Buyya, 2016), (Q. Zhang, Chen, & Yin, 2017), (Rankothge, Le, Russo, & Lobo, 2017)]. (Ye et al., 2017) designs a hybrid auto-scaling framework for containerized elastic applications. They employ a prediction technique based on Auto-Regression Moving Average (ARMA) to forecast applications' future resource demands. Then, because vertical elasticity is faster than horizontal one, they suggest scaling up the resources vertically to handle real-time load variations and scaling them out horizontally to meet future resource needs. However, horizontal scale-in is used to release resources in both scenarios by reducing the number of containers, because horizontal scaling is more cost-aware than vertical one. In their implementation, they focus on scaling containers' CPU resources specifically and they intend to minimize SLA violations (response time). (Rankothge et al., 2017) presents two resource allocation algorithms based on genetic programming, for VNFs initial placement and VNFs scaling. These algorithms are compared with the ILP model implemented in CPLEX. In the context of VNF initial placement, the main objective is to minimize the number of servers used and the network resources. For VNF scaling part, in addition to the previously mentioned objectives, the aim is to reduce the number of changes in server and links configurations. The algorithms support both horizontal and vertical scaling and target VNF chains. However, they assume that each VNF in a chain can have only one successor, which is not the case in non-linear SFC topologies.

Elasticity is not the only way to adapt resources in the cloud. Some proposals, such as [(Chaloemwat & Kitisin, 2016), (Al-Dhuraibi, Paraiso, Djarallah, & Merle, 2017), (Nadjaran Toosi et al., 2019), (Rankothge, Ramalhinho, & Lobo, 2019), (Jia, Wu, Li, Le, & Liu, 2018), (Liu, Lu, Zhou, Lu, & Zhu, 2017)], include a mix of elasticity and migration mechanisms. (Chaloemwat & Kitisin, 2016) introduces a combination of horizontal auto-scaling and migration techniques for cloud services with a skewness algorithm. Skewness algorithm aims

to measure the unevenness of resource utilization on a physical machine. Minimizing the skewness value leads to an improvement in overall resource utilization on the physical host. If the current resource utilization of a VM exceeds the predefined thresholds, it is considered overloaded, then the processes running on this VM should be migrated to another available VM or a new VM. An equation is also defined to measure the pressure which means the overloading degree of a VM. Processes running on the VM with the highest pressure should migrate to an idle VM whose skewness value increases the least due to this migration. If no potential destination VM is found, the processes are migrated to a new VM. On the other hand, the duration of being in an idle state is measured for the VM, and if the remaining processes running on this VM can be migrated to other available VMs without overloading them, the idle VM can be scaled down. The authors in (Al-Dhuraibi et al., 2017) create ElasticDocker system, which combines vertical elasticity and live migration for Docker containers. When there are not enough resources to scale containers vertically, live migration is triggered to move the container to another host. Vertical elasticity scales up and down both CPU and memory of containers. Live migration technique is based on CRIU functionality in Linux systems. Evaluation results demonstrate that ElasticDocker can minimize customer expenses, improve resource utilization, enhance the quality of experience (QoE) for end-users and achieve better results compared to Kubernetes elasticity. Extending this solution to support horizontal elasticity may produce a more efficient and a complete resource adaptation system for Docker containers. (Nadjaran Toosi et al., 2019) presents a heuristic end-to-end latency-aware auto-scaling algorithm called ElasticSFC that considers vertical and horizontal scaling, migration, and flow scheduling. They focus on the dynamic allocation of CPU resources and network bandwidth for the service chains while meeting the required latency requirements. To adjust the VNF computing resources, the algorithm first attempts to vertically scale up the resources. If vertical scaling is impossible due to resource constraints, it tries horizontal scaling by adding VNF instances. For bandwidth adaptation, flow scheduling is used to redirect traffic to an alternate network path capable of providing the requested bandwidth and satisfying the required latency. If no potential path is available, VNF migration is adopted to move either of the two end VNF instances of the link or both to new destination hosts. (Rankothge et al., 2019) formulates the resource reallocation problem of VNFs as an ILP model. Their optimization

goal is to minimize the *bandwidth dropped* defined as the requested bandwidth that cannot be fulfilled by the allocated resources. Because optimal solutions are time-consuming, they develop an approximation algorithm based on the meta-heuristic Iterated Local Search (ILS) to solve the VNFs scaling problem. Their ILS technique handles the three scaling methods (vertical scaling, horizontal scaling, and migration) focusing essentially on satisfying the bandwidth demands of the VNF policies. In (Liu et al., 2017), the authors study the problem of the deployment of new users' SFCs and the readjustment of in-service users' SFCs while finding a trade-off between resource consumption and operational overhead. The operational overhead for a new user equals the number of VNFs required in its SFC request, while for an in-service user, it equals the number of migrated and newly added VNFs since the last service time. The problem is first formulated as an ILP model to find the exact solution. Then, to reduce the time complexity of ILP, they design a column generation model and implement an approximation algorithm based on it to solve the problem. The optimization goal is to maximize the service provider's profit which is equal to the total profit gained from serving SFC requests minus the total deployment cost.

In the context of VM migration, some authors propose different migration techniques (e.g., post-copy, pre-copy, or hybrid migration) to move the application from one physical machine to another one that has enough capacity to host it [(S. He, Hu, Shi, Wo, & Li, 2016), (F. Zhang, Fu, & Yahyapour, 2017), (Level, 2016), (Wahab, Kara, Edstrom, & Lemieux, 2019), and (Eramo, Miucci, Ammar, & Lavacca, 2017)]. The authors in (Choudhary et al., 2017) present a critical review of state-of-the-art live VM migration techniques, their strengths, and weaknesses. One of the main challenges in VNF migration is the VNF placement or bin packing problem. This problem consists of embedding the VNFs into the physical infrastructure while selecting the best server to host each VNF. Specifically, such proposals decide if a VNF will remain on the same server or will be migrated to another one while specifying the new destination [(Silva Filho et al., 2018), (Tavakoli-Someh & Rezvani, 2019), (Khebbache, Hadji, & Zeghlache, 2018), (El Mensoum, Wahab, Kara, & Edstrom, 2020), (Zhiyong Li, Li, Yuan, Chen, & Jiang, 2019), (Laaziz et al., 2019), (Abdelaal, Ebrahim, & Anis, 2021), and (Mai et al., 2021)]. Many algorithms are suggested to select the destination

servers such as NSGAII [(Tavakoli-Someh & Rezvani, 2019), (Khebbache et al., 2018), (Laaziz et al., 2019)] and CRO [(Zhiyong Li et al., 2019), (El Mensoum et al., 2020)]. However, they don't differentiate the three-adaptation technique. On the one hand, we may decide to keep a VNF on its current server while scaling it vertically (increasing the resources of this VNF instance) or horizontally (adding a new VNF instance on the same server). On the other hand, we may choose a new destination server for the designated VNF instance if we decided to migrate it, or for the newly added VNF instance if we decided to scale it horizontally. Each adaptation technique involves different costs in terms of CPU, energy, etc. and we should distinguish between them.

In this paper, we formulate the resource allocation problem as an ILP model to find the exact solutions. Contrary to existing works, in our formulation, a decision variable is set for each resource adaptation method (HS, VS, and migration) and the objective functions are calculated according to these decisions. The output of our model is the set of resource adaptation methods decided to satisfy the resource requirements for the received SFC requests. To the best of our knowledge, we are the first to formulate the problem in this way. We also design decision-making meta-heuristic algorithms based on NSGAII, CRO, Binary PSO, and the combination (CRO - Binary PSO) techniques. Our algorithms benefit from several advantages compared to the state of the art, which are: (1) they consider the three possible resource adaptation mechanisms to cover various overloading server states while selecting the best one for each of them. (2) they target both linear and non-linear SFC topologies; (3) they aim to balance a set of objectives, including saving energy, improving resource utilization (CPU), and minimizing SLO violation (end-to-end latency); (4) the proposed idea is generic enough to be used in various SFCs and virtualized system or data center architecture. Finally, to the best of our knowledge CRO, binary PSO and their combination have not been considered in the literature to address these research challenges.

Table 2.1 Notations

Symbol	Description
S	Set of servers in the substrate network
R	Set of SFC requests
V	Set of VNF types in an SFC request
C	Set of VNF instances in an SFC request
M_c^r	Decision variable indicates if VNF instance $c \in C$ in an SFC request $r \in R$ will be migrated
VS_c^r	Decision variable indicates if VNF instance $c \in C$ in an SFC request $r \in R$ will be scaled vertically
HS_c^r	Decision variable indicates if VNF instance $c \in C$ in an SFC request $r \in R$ will be scaled horizontally
$X_s^{c,r}$	Decision variable that indicates if VNF instance $c \in C$ in an SFC request $r \in R$ will be placed on server $s \in S$
Y_s	Decision variable that indicates whether a server $s \in S$ is used
$t_{c,r}^{VS}$	The time taken while scaling a VNF instance $c \in C$ in an SFC request $r \in R$ vertically
$t_{c,r}^{HS}$	The time taken while scaling a VNF instance $c \in C$ in an SFC request $r \in R$ horizontally
$t_{c,r}^{M}$	The time taken while migrating a VNF instance $c \in C$ in an SFC request $r \in R$
$P_{c,r}^{CPU}$	Power consumed by a VNF instance on its current server
$P_{c,r}^{CPU^V}$	Power consumed by a server when scaling a VNF instance vertically
$P_{c,r}^{CPU^H}$	Power consumed by a server when scaling a VNF instance horizontally
$P_{c,r}^{CPUdestination}$	Power consumed by a VNF instance on destination server after migration
$P_{c,r}^{LB}$	The power consumed by the load balancer
P_s^{idle}	The power consumed by a server $s \in S$ in idle state
$CPU_{c,r}$	CPU utilization (%) of a VNF instance on its current server
$CPU_{c,r}^{V}$	CPU (%) consumed by a server when scaling a VNF instance vertically
$CPU_{c,r}^{H}$	CPU (%) consumed by a server when scaling a VNF instance horizontally
$CPU_{c,r}^{destination}$	CPU usage (%) when a VNF instance is running on a remote server after migration
$CPU_{c,r}^{LB}$	CPU (%) consumed by the load balancer
CPU_s^{idle}	CPU utilization of a server $s \in S$ in idle state
$CPU_{c,r}^{requested}$	Number of CPU cores requested by a VNF instance
$CPU_s^{available}$	The server available capacity (Number of CPU cores)
$X_{s\,(original)}^{c,r}$	Binary variable indicates if a VNF instance $c \in C$ in an SFC request $r \in R$ is initially placed on a server $s \in S$
$CPU_{c,r}^{allocated}$	Number of CPU cores allocated for a VNF instance on its current/original server
$L_{v,v'}^r$	Set of links connecting the instances of a pair of VNF types $v \in$ V and $v' \in$ V in an SFC
$d_{v,v'}^r$	Latency on a link connecting a pair of VNF types $v \in$ V and $v' \in$ V in an SFC
P	Set of paths in an SFC topology
LA	The worst latency found among the set of received SFC requests

2.4 Problem formulation

Let $G_p = (S, E_p)$ be an undirected graph representing the physical network, where S is the set of servers and E_p is the set of links connecting them. On the other hand, $G_v = (C, E_v)$ is a directed graph of an SFC, where C is the set of VNF instances and E_v is the set of virtual links among these instances. In the following, we describe our ILP formulation of the resource

adaptation problem in detail. The notations used in the problem formulation are described in Table 2.1.

2.4.1 Decision variables

A binary variable is used for each possible resource adaptation strategy, namely, migration, horizontal scaling, and vertical scaling. Two variables are added, one to specify the VNF instance replacement resulting from the decision taken, and another, to indicate whether a server is used.

$$M_c^r = \begin{cases} 1 \ \textit{if VNF instance } c \in C \textit{ in an SFC request } r \in R \\ \qquad \textit{will be migrated} \\ 0 \ \textit{otherwise} \end{cases} \tag{2.1}$$

$$VS_c^r = \begin{cases} 1 \ \textit{if VNF instance } c \in C \textit{ in an SFC request } r \in R \\ \qquad \textit{will be scaled vertically} \\ 0 \ \textit{otherwise} \end{cases} \tag{2.2}$$

$$HS_c^r = \begin{cases} 1 \ \textit{if VNF instance } c \in C \textit{ in an SFC request } r \in R \\ \qquad \textit{will be scaled horizontally} \\ 0 \ \textit{otherwise} \end{cases} \tag{2.3}$$

$$X_s^{c,r} = \begin{cases} 1 \ \textit{if VNF instance } c \in C \textit{ in an SFC request } r \in R \\ \qquad \textit{will be placed on server } s \in S \\ 0 \ \textit{otherwise} \end{cases} \tag{2.4}$$

$$Y_s = \begin{cases} 1 \ \textit{if server } s \in S \text{ is used} \\ 0 \ \textit{otherwise} \end{cases} \tag{2.5}$$

2.4.2 Constraints

We consider the following constraints:

$$M_c^r + VS_c^r + HS_c^r = 1 \quad \forall c \in C, r \in R \tag{2.6}$$

$$\sum_{s=1}^{|S|} X_s^{c,r} = 1 \quad \forall c \in C, \qquad r \in R \tag{2.7}$$

$$\left(X_{s\,(original)}^{c,r} - X_s^{c,r}\right) \times VS_c^r = 0 \quad \forall c \in C, \qquad s \in S, \qquad r \in R \tag{2.8}$$

$$\left(X_s^{c,r} \oplus X_{s\,(original)}^{c,r}\right) \times M_c^r = 1 \quad \forall c \in C, \qquad s \in S, \; r \in R \tag{2.9}$$

$$\sum_{r=1}^{|R|}\sum_{c=1}^{|C|}\left(CPU_{c,r}^{requested} \times X_s^{c,r}\right) \leq CPU_s^{available} + \sum_{r=1}^{|R|}\sum_{c=1}^{|C|} CPU_{c,r}^{allocated} \times M_c^r \times X_{s\,(original)}^{c,r} \qquad \forall s \in S \tag{2.10}$$

$$LA \leq Thresh \tag{2.11}$$

Constraint (2.6) ensures that only one resource adaptation strategy will be chosen for each VNF instance. Constraint (2.7) guarantees that only one server can be chosen as the destination of each VNF instance. Constraint (2.8) determines that the VNF instance should remain on its original server, in the case of vertical scaling. Contrary to the previous constraint, constraint (2.9) determines that the selected destination server to host the VNF instance in case of migration should be different from its original host. Constraint (2.10) guarantees that the amount of resources requested by VNF instances should not exceed the available capacity of the server. In this paper, we focus on one of the most critical metrics in cloud settings, CPU utilization. Other metrics such as memory, disk and I/O may be added to our formulation in future work. We consider that some resources will become free upon migration processes and will thus be added to the available server capacity. Note that in the case of migration, the requested CPU cores include not only the number of CPU cores needed by the VNF instance but also its allocated CPU cores that will be migrated to the destination server. Constraint (2.11) ensures that the worst end-to-end delay/latency LA found among the set of received SFC requests does not exceed the tolerance threshold based on SLO. The detailed calculation of LA is explained in the next sub-section.

2.4.3 Cost functions

Essentially, our model targets three different objectives: minimizing energy consumption, SLO violation, and resource consumption in terms of CPU.

2.4.3.1 Resource consumption

Resource consumption represents the total CPU utilization consumed by the target VNF, as well as the servers, in addition to the CPU consumed by the resources added during vertical scaling; the computational units added and the load balancer during horizontal scaling, and the migrated instance on the destination machine, in the case of migration.

$$G1 = \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} CPU_{c,r} + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} HS_c^r \times (CPU_{c,r}^H \times t_{c,r}^{HS} + CPU_{c,r}^{LB}) + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} VS_c^r \times CPU_{c,r}^V \times t_{c,r}^{VS} + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} M_c^r \times (CPU_{c,r}^{destination} + CPU_{c,r} \times t_{c,r}^M) + CPU_s^{idle} \times Y_s \quad (2.12)$$

2.4.3.2 Power consumption

Saving energy is a major concern for today's cloud data centers. Since greater CPU consumption will increase power electricity and cooling costs, and generate more heat in the cloud environment, in this article, we mainly focus on processing power. Optimizing the calculation of energy consumption while taking into account other resources (e.g., memory, I/O in addition to CPU) is one of our future work steps. In our formulation, energy consumption represents the energy consumed by the CPU of the current server and the target VNF, plus the energy consumed by the resources added during vertical scaling; the servers added, the VNF instances and the load balancer, in the case of horizontal scaling; and the migrated instance on the destination machine after migration.

$$G2 = \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} P_{c,r}^{CPU} + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} VS_c^r \times \left(P_{c,r}^{CPU^V} \times t_{c,r}^{VS}\right) + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} HS_c^r \times \left(P_{c,r}^{CPU^H} \times t_{c,r}^{HS} + P_{c,r}^{LB}\right) + \sum_{s=1}^{|S|}\sum_{r=1}^{|R|}\sum_{c=1}^{|C|} M_{v,c}^r \times \left(P_{c,r}^{CPU} \times t_{c,r}^{M} + P_{c,r}^{CPU_{destination}}\right) + P_s^{idle} \times Y_s \quad (2.13)$$

As the maximum (P_s^{max}) and the idle (P_s^{idle}) power consumption of a server are known, the power consumed by a VNF instance hosted on this server is calculated by the following formula (Beloglazov, Abawajy, & Buyya, 2012):

$$P_{c,r}^{CPU} = \left(\left(\frac{P_s^{max} - P_s^{idle}}{max_{cores}}\right) * CPU_{c,r}^{allocated}\right) \times \frac{u}{100} \quad (2.14)$$

Where max_{cores} is the total number of CPU cores of the server, $CPU_{c,r}^{allocated}$ represents the number of cores allocated to the VNF instance, and u is the CPU utilization of this VNF.

2.4.3.3 SLO violation

To avoid SLO violations, we minimize the end-to-end delay/latency of the SFC requests. Particularly, we calculate the total delay consumed on the physical links that connect the source VNF to the destination VNF of an SFC request. We consider linear and non-linear SFC topologies. In non-linear chains, each VNF type $v \in V$ in the chain may have many instances, and each of these instances may be hosted on a different server. We assume that the delay between a pair of VNF types is equal to the max delay on the links connecting all their instances (Eq. 2.15). Also, the VNF may have more than two successors, and so many paths are possible for traffic. We first calculate the total end-to-end delay of each path in the SFC topology DP_p^r. The total delay of a path DP_p^r in an SFC represents the sum of the delays between all VNF peers on this path (Eq. 2.16). Then, we take the max path delay as the SFC latency DS^r (Eq.

2.16). In the end, our objective is to minimize the max or the worst SFC latency LA found by the algorithm among the set of SFC requests (Eq. 2.18).

$$D^r_{v,v'} = \max_{L^r_{v,v'}} d^r_{v,v'} \quad \forall\, v, v' \in V,\ r \in R \tag{2.15}$$

$$DP^r_p = \sum_{v}^{|V|} D^r_{v,v'} \quad \forall\, v, v' \in V,\ r \in R \tag{2.16}$$

$$DS^r = \max_{P} DP^r_p \quad \forall p \in P, \quad r \in R \tag{2.17}$$

$$LA = \max_{R} DS^r \quad \forall r \in R \tag{2.18}$$

Consequently, the algorithm searches for a resource adaptation solution that has the least bad SFC latency, coupled with minimal CPU utilization and energy consumption.

2.5 Algorithms

Finding an optimal resource allocation solution is computationally hard. Solving the ILP problem is a time-consuming process and is consequently not suitable to meet traffic fluctuations in real-time. Hence, it is more practical to look for approximations using heuristic-based techniques. In this section, we provide a detailed presentation of our proposed multi-objective resource adaptation algorithms based on NSGAII (Deb, Pratap, Agarwal, & Meyarivan, 2002), Chemical reaction optimization (CRO) [(Lam & Li, 2012), (Islam, Saifullah, & Mahmud, 2019)], Binary Particle Swarm Optimization (NBPSO) [(D. Wang, Tan, & Liu, 2018), (Nezamabadi-Pour, Rostami-Shahrbabaki, & Maghfoori-Farsangi, 2008)] and the combination CRO-NBPSO. All proposed algorithms are population-based optimization techniques that start by the initialization of a set of random solutions, then iteratively manipulate some of them to generate new ones using specific operators, and generate the best solution found in terms of cost functions while respecting the constraints discussed in section 2.4. The termination condition is the max number of iterations. In this section, we first discuss the common strategies used in our algorithms, related to solution representation and population

initialization. Then, we explain the working principles and the operators used in each algorithm to manipulate the solutions.

2.5.1 Common strategies

2.5.1.1 Solution encoding

In our problem, a solution consists of resource adaptation decisions for all the SFC requests received. As mentioned earlier, each SFC is a chain of network functions (NFs) connected to each other. In the case of a non-linear chain, each network function may have many instances (copies) hosted on different virtual machines. For this reason, we represent a solution as a multidimensional array that includes a resource adaptation decision for each VNF instance in each SFC request. To represent the adaptation strategies, we used a binary representation with 2 bits, as follows: Do Nothing N = 00, Vertical Scaling V = 01, Horizontal Scaling H = 10, Migration M = 11. For instance, if the received SFC consists of 4 chained VNFs, namely, *Cache→Firewall→PDI→Video Streaming,* with Cache has one instance, Firewall and PDI have 2 instances each, and video streaming has 3 instances, a possible solution may look like [[11], [01, 10], [10, 11], [01, 01, 11]], with 11 (M) being the decision taken for the first VNF (Cache), 01 (V) and 10 (H) being the decisions for the Firewall instances, 10 (H) and 11 (M) being the decisions for the PDI instances and so on. Therefore, for each SFC, the solution's length is equivalent to *2 bits* SFC size*, where the SFC size represents the number of VNF instances composing the chain. In fact, the algorithm generates a solution for a set of SFC requests instead of one. If there are m number of SFC requests deployed in our environment, then a full solution contains m number of partial solutions, each representing the resource adaptation decisions for the VNFs sequence of each SFC. Consequently, the individual's length depends on the number of SFC requests to be adapted and on the size of each one: $\sum_{r=1}^{R} 2bits * SFC\ size$, where R is the number of requests.

To evaluate the feasibility of the solutions and compute the resulting costs, we need to know the destination servers for the decisions taken. In particular, to verify the latency constraint,

we need to calculate it according to the modified SFC topologies and the chosen hosting servers. The decisions taken can modify the SFC graph by adding for example new VNF instances in case of horizontal scaling. Also, In the case of vertical scaling, the instance remains on the same server, but for horizontal scaling and migration, a destination server may be needed to host the newly added or the migrated VNF instance. Thus, to test our decision-making proposals, we need a strategy to choose the destinations servers. Because SLO violation avoidance is one of our main objectives, we adopt a server selection strategy that intends to minimize the latency for each partial solution. Briefly, for each SFC request, we choose the first server that has enough capacity to host the first VNF instance of the chain. Then, for the next instance, we select the server that has not only a sufficient capacity but also, minimal latency with the one already chosen to host the previous instance of the same chain and so on. Although this placement strategy is used in our testing experiments, our decision-making algorithms can be combined with another more advanced placement strategy (host selection strategy) to optimize their performance and accuracy in selecting adaptation decisions.

2.5.1.2 Population initialization

The population consists of “n” possible solutions. At first, we initialized these solutions randomly. The algorithms were able to find sub-optimal solutions for our problem quickly unless we added the latency constraint (Equation 2.11). Using very stressful latency thresholds in some scenarios results in our algorithms being unable to find any feasible solution, while an ILP solution is found. To solve this discrepancy, we initialize the population with random, but feasible solutions. Specifically, we generate a random solution, adjust its adaptation decisions in terms of resource availability, and then check its feasibility in terms of latency before adding it to the initial population. This is done by the feasibility operator described in the next sub-section. Adopting this initialization strategy increases the runtime of our algorithms a little bit but avoids their failure.

2.5.1.3 Feasibility operator

The solutions in the initial population are manipulated randomly by operators (e.g., in NSGAII and CRO) or by equations (e.g., in NBPSO) to create new ones without considering the constraints imposed by our research problem. This operator is implemented to verify the feasibility of the newly generated solutions specifically in terms of resource availability and tolerated latency. In terms of resource availability, we check if there are inapplicable random vertical scaling decisions taken in the solution. If resources are insufficient, we randomly change this decision to horizontal scaling or migration. Once the decisions are adjusted and their destination servers are chosen, we calculate and check the latency constraint. Only feasible solutions are added to the population.

2.5.2 NSGA-based algorithm

2.5.2.1 Working principles

The non-dominated sorting genetic algorithm (NSGAII) aims to find a trade-off between the three cost functions defined in section 2.4, namely, Equation (2.12) for the CPU utilization, (2.13) for the Energy consumption, and (2.18) for the Latency. In each iteration, the algorithm applies two essential sorting mechanisms: n*on-dominant and crowding distance* sorting. The first mechanism ranks the individuals according to their fitness functions and divides the population into several subsets/fronts. Whereas the second sorting technique ranks the solutions in each front according to their crowding distance values. Based on the rank and the crowding distance parameters, the best individuals (called parents) are selected from the population. Then, the genetic operators, namely, the crossover and the mutation, which will be discussed in detail in a later section, are applied to the selected individuals to produce an offspring population. Each cycle will produce a new population that converges more toward the best solution. The output of the algorithm is a Pareto front of sub-optimal solutions with their cost function values.

2.5.2.2 Genetic operators

Crossover: To perform this process, we use a modified Half-uniform crossover (HUX) operator. This operator randomly swaps non-matching adaptation decisions between the two parents. For instance, there are four non-matching adaptation decisions between the parents P1 and P2, particularly on the following indexes: (0, 1), (1, 0), (1, 2), (2, 1), and (2, 2). The operator randomly chooses to swap the decisions of the indexes (0, 1), (1, 2), and (2, 2) to produce the children R1 and R2.

P1 : [[11, **01**, 10], [**10**, 01, **10**],[01, **11**, **11**]]
P2: [[11, **10**, 10], [**11**, 01, **11**],[01, **01**, **10**]]

R1 : [[11, **10**, 10],[10, 01, **11**],[01, 11, **10**]]
R2: [[11, **01**, 10],[11, 01, **10**],[01, 01, **11**]]

Mutation: For mutation, we adopt an operator similar to the concept of BitFlip mutation. It selects random positions in the chromosome and then changes its adaptation decisions. As shown in the example below, the decisions on the indexes (0, 0), (1, 1), and (2, 2) are randomly changed.

P : [[**11**, 01, 10],[10, **01**, 10], [01, 11, **11**]]

P' : [[**10**, 01, 10],[10, **10**, 10], [01, 11, **01**]]

As the end goal is to converge toward the best solution, crossover happens more frequently, and its probability is higher than mutation. These probabilities are specified as input and can easily be changed at any time.

2.5.3 CRO-based algorithm

2.5.3.1 Working principles

Our second algorithm is based on Chemical reaction optimization (CRO). Since its appearance, CRO has demonstrated its power to solve different kinds of optimization problems, which is what motivated us to choose it as a base technique to develop our decision-making algorithm. Unlike genetic algorithms, the population size in CRO varies through iterations. Each molecule (solution) in the population is characterized by its molecular structure (W), potential energy (PE), kinetic energy (KE), total hits number, minimum hits number, etc. In our algorithm, W is the multidimensional array of potential resource adaptation mechanisms for the received SFC requests. PE represents the objective function value of the solution W. In contrast, CRO is not a multi-objective algorithm as is NSGA, and so instead of using our three cost functions independently, we are minimizing the summation of these functions (Equation 2.19). KE states a tolerance of accepting a worse solution than the existing one. *Hits_number* records the total number of times a molecule has collided and *min_hits_number* indicates the number of collisions when it achieves the min PE. Collisions happen when molecules interact with each other due to chemical reactions in order to reach an equilibrium state of minimal PE. There are four essential elementary reactions, namely, On-wall ineffective collision, Inter-molecular ineffective collision, Decomposition, and Synthesis. The two ineffective collisions perform the *local search (intensification)* while decomposition and synthesis implement the *diversification* effect. The intensification and diversification are controlled by parameters α and β.

$$PE = G1 + G2 + LA \tag{2.19}$$

2.5.3.2 Operators

In the following, we explain in detail the four operators used in our algorithm to perform the CRO chemical reactions. As we mentioned, the solution is encoded in binary format, but for

simplicity, and to clearly explain the chosen operators, we used the symbols V = Vertical Scaling, H = Horizontal Scaling, and M= Migration in our examples.

1. *On-wall ineffective collision:* This occurs when a molecule hits an external substance (e.g., the wall of the container), resulting in a subtle change in its molecular structure from ω to ω'. For this operator, we randomly pick one VNF instance from each SFC chain in the solution and change its adaptation decision.

2. *Decomposition*: This takes place when a molecule hits a wall and then splits into several parts. For simplicity, we assumed each molecule ω breaks into two parts ω_1 and ω_2. In this reaction, we adopt the half-total change operator and apply it to manipulate the resource adaptation decisions taken for each SFC request. In general, this operator generates a new solution ω' from an existing one ω, by keeping one-half of its values and assigning the other half with new values. Specifically, we first copy all the decisions taken in molecule ω to ω_1 and ω_2. Then in each, we randomly select [n/2] VNF instances from each SFC chain and offer them new adaptation decisions. Note that 'n' represents the number of VNF instances in the SFC or what we called the SFC size. Let us suppose we try to adjust the resources for three SFC requests of the same size (e.g., four VNF instances each). In that case, after copying the values of ω to ω_1 and ω_2, we need to assign new decisions for two random VNF instances in each SFC. For example:

   ω : [[M, M, V, H], [H, M, V, H], [V, M, H, M]]

   ω_1: [[M, **H**, V, **M**],[**V**, M, V, **M**], [V, M, **V**, **H**]]
   ω_2: [[**V**, M, V, **M**],[H, **V**, **H**, H], [V, **H**, **M**, M]]

3. *Inter-molecular ineffective collision:* This refers to the situation when multiple molecules (let us assume two) collide with each other. As a result, their molecular structures ω_1 and ω_2 are perturbed and slightly change to ω'_1 and ω'_2, respectively. This

collision is performed in our algorithm by selecting two random positions in the vector of each SFC. Then, we exchange the adaptation decisions located between the selected positions in ω_1 and ω_2 to produce ω'_1 and ω'_2. As shown below, for the first SFC, the decisions between the two chosen indexes 0 and 3 are exchanged.

ω_1: [[M, H, H, M], [V, M, V, M], [V, M, V, H]]
ω_2: [[V, M, V, M], [H, V, H, H], [V, H, M, M]]

ω'_1: [[M, **M, V**, M], [V, **V**, V, M], [V, M, **M**, H]]
ω'_2: [[V, **H, H**, M], [H, **M**, H, H], [V, H, **V**, M]]

4. *Synthesis*: Contrary to the decomposition, synthesis is the situation when many molecules hit each other to fuse together into one new molecule. Thus, two existing solutions ω1 and ω2 are merged into a new solution ω', different from them individually. For this reaction, we used the Probabilistic select operator that randomly chooses values from ω1 and ω2 to generate ω'. Each decision ω'(i) is equal to either ω1(i) or ω2(i), with the same probability.

ω_1: [[M, H, H, M], [V, M, V, M], [V, M, V, H]]
ω_2: [[V, M, V, M], [H, V, H, H], [V, H, M, M]]

ω': [[M, M, H, M], [V, M, H, H], [V, H, M, H]]

2.5.4 NBPSO-based algorithm

Particle swarm optimization (PSO) is a population-based optimization technique inspired by the behavior of bird flocking. Due to our binary representation of the solutions, the binary version of PSO, discussed in (Nezamabadi-Pour et al., 2008), is adopted to solve the problem. Each solution or particle i in PSO is characterized by its position x_i^k and velocity v_i^k. The position is the binary set of adaptation decisions while the velocity is the probability of

changing the position bit values. The velocity boundaries are $[-v_{max}, v_{max}]$ where v_{max} is typically set to 6. Unlike genetic and CRO algorithms, PSO does not have operators to manipulate the solutions, it applies certain equations instead. In every iteration, each particle updates itself depending on the two best positions p_i^k and g_i^k. The first is the best position the particle has achieved so far, in terms of cost function values. The second is the best position obtained so far among the particles in the population. The cost function represents the overall costs calculated in (Equation 2.19). After updating the velocity of the particle and calculating the sigmoid (Equations 2.20, 2.21, and 2.22), we compare the *rand ()* value to $S'(v_i^{k+1})$ and then decide if each position bit value will be switched from 0 to 1 or vice versa, or will remain unchanged (Equation 2.23). These bit changes refer to manipulations in adaptation decisions.

$$v_i^{k+1} = wv_i^k + c_1 r_1^k \left(p_i^k - x_i^k\right) + c_2 r_2^k \left(g_i^k - x_i^k\right) \tag{2.20}$$

$$Sigmoid(v_i^{k+1}) = \frac{1}{1 + e^{-v_i^{k+1}}} \tag{2.21}$$

$$S'(v_i^{k+1}) = 2 \times |Sigmoid\left(v_i^{k+1}\right) - 0.5| \tag{2.22}$$

$$x_i^{k+1} = \begin{cases} exhange\left(x_i^k\right) & \text{if rand () } < S'(v_i^{k+1}) \\ x_i^k & otherwise \end{cases} \tag{2.23}$$

r_1^k , r_2^k and *rand ()* are random numbers in the range [0,1], c_1 and c_2 are positive constants, and w is the inertia weight which describes the effect of the previous velocity on the new one. w is calculated and updated in each iteration using the following formula:

$$w = w_{max} - \frac{w_{max} - w_{min}}{T_{max}} t \tag{2.24}$$

Typically, w_{max}= 0.9, w_{min}=0.4; t is the current iteration number; T_{max} is the maximum number of iterations.

2.5.5 CRO-NBPSO algorithm

A good optimization technique should perform well at both global (exploration) and local (exploitation) searches. In this section, we implement an algorithm based on balanced local and global search. The idea is inspired by (Nguyen, Li, Zhang, & Truong, 2014). This algorithm combines the PSO's strong global search ability with the CRO's local search performance. Contrary to (Nguyen et al., 2014), we combine CRO with a binary version of PSO and apply the technique to solve the resource allocation problem. We re-use the same CRO collision operators described in section 2.5.3 and the same objective function (Equation 2.19). However, CRO's global search operators (Decomposition and Synthesis) are replaced by the Binary PSO algorithm described previously. To the best of our knowledge, we are the first to adopt such a combination technique (CRO - binary PSO) to address this research problem.

Contrary to CRO, the population size in this technique doesn't vary through iterations. Each molecule represents a particle at the same time. All parameters related to CRO's global operators are removed, and new parameters are added including PSO parameters and a control parameter ɣ. Figure 2.1 shows a modified flow chart of the algorithm (Nguyen et al., 2014).

In each iteration, a random molecule is selected from the population. Then we compare its number of hits to ɣ. The number of hits means the number of times this molecule has undergone collisions (On-wall ineffective collision, Inter-molecular ineffective collision). According to the comparison result between its MolHits and ɣ, we decide whether the molecule will be manipulated by PSO or by CRO collisions. In the case of CRO manipulation, there are two types of collisions. To choose one of them, a random number r in the range [0,1] is compared to CollRate. If r is larger than CollRate, Inter-molecular ineffective collision is triggered, and the molecule will collide with another one randomly chosen. Otherwise, *On-wall ineffective collision* will take place. Each time the molecule collides, its *MolHits* increase by one. When its *MolHits* become higher than ɣ, PSO equations are applied to manipulate the molecule's structure and its *MolHits* are reset to zero.

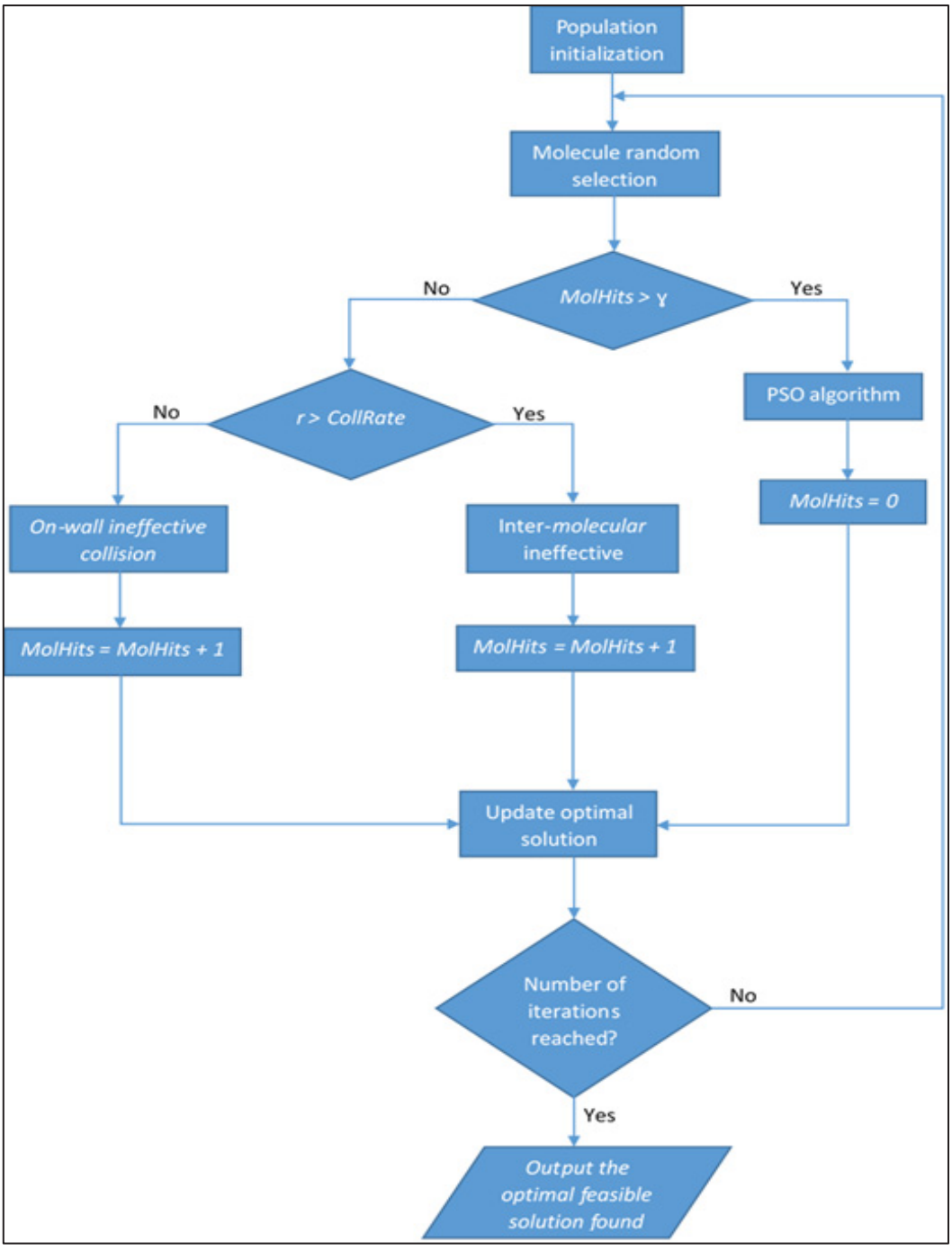

Figure 2.1 Flow chart CRO - Binary PSO

2.6 Experiments

2.6.1 Experimental design

In this study, we sequentially implemented an ILP model, NSGA-based, CRO-based, BPSO-based, and CRO_BPSO decision-making algorithms to dynamically find the optimal resource adaptation techniques for the service function chains in different scenarios. The algorithms were tested on a machine with an 8×Intel core i7-7700 CPU processor, 15.5 GB of RAM, and Ubuntu 18.04.2 LTS 64-bits. All programs were written using Python and executed on PyCharm IDE. NSGAII was implemented based on the Platypus Library (https://github.com/Project-Platypus/Platypus). Gurobi was used to solve the ILP formulation and find the exact solution to the problem. Due to the long convergence time required by the ILP approach and to compare the performance of our heuristic algorithms in finding a near-optimal solution, we limited the network size to 20 servers. The servers each have 16 CPU cores and peak and idle power consumption of 135W and 93.7W, respectively. We assumed that the average CPU utilization of servers is around 20% in their common state. Additionally, they consume 70% of their peak energy consumption in their idle state (Basmadjian, Niedermeier, & De Meer, 2012)(Hsieh, Liu, Buyya, & Zomaya, 2020). We generated the

Table 2.2 Meta-Heuristic Parameters

NSGAII	Mut_rate	Cross_rate			
	0.3	0.8			
CRO	InitialKE	*CollRate*	KELossRate	α	β
	100	0.2	0.2	100	100
NBPSO	c_1	c_2	w	v_{max}	
	2	1	(Eq. 2.24)	6	
CRO-NBPSO	ɣ				
	2				
Common criteria	Popsize	T_{max}			
	100	100			

Mut_rate = Mutation rate; Cross_rate = Crossover rate; Popsize = population size; T_{max}= maximum number of iterations

network topology using NetworkX library and set the delays on the links connecting the servers randomly between *0.01* and *0.06* seconds.

Moving on to the virtual network side, we varied the incoming number of SFC requests between 5 and 20 chains, and the SFC sizes between 3 and 10 VNF instances. The SFCs were generated using the same library NetworkX and initially placed in the infrastructure randomly. Each VNF in an SFC initially has one CPU core each. Both linear and non-linear SFC topologies were considered. Also, one load balancer was added to the network to balance the traffic between the VNF instances in the case of horizontal scaling. The parameters of our meta-heuristic algorithms are summarized in Table 2.2. Note that CRO-NBPSO takes as input the parameter ɣ as well as all NBPSO parameters and CRO parameters except α and β related to global search. It is worth mentioning that each experiment was repeated 10 times and we report the average results. Table 2.3 represents the worst or the highest standard deviation of the results of each algorithm in each test scenario.

Table 2.3 Worst Standard Deviation obtained

	Scenario 1	**Scenario 2**	**Scenario 3**	**Scenario 4**
NSGAII	2.145	2.001	1.671	1.918
CRO	0.213	0.243	0.281	0.248
NBPSO	0.174	0.128	0.151	0.119
CRO-NBPSO	0.318	0.245	0.251	0.291

2.6.2 Implementation

We evaluated the performance of our algorithms based on the cost functions defined in section 2.4, including, CPU utilization (Eq. 2.12), energy consumption (Eq. 2.13), and end-to-end SFC Latency (Eq. 2.18). Moreover, we compared their execution time to find solutions and the total number of servers used to adapt the requests received. We chose the following test cases for the assessments:

Scenario 1: in this scenario, we compared the performance of our algorithms while increasing the number of incoming SFC requests. We assumed that all SFC requests received consist of linear topologies of 5 VNF instances each. The number of incoming SFC requests was varied between 5 and 20 chains

Scenario 2: it is similar to scenario 1 in receiving 5 to 20 SFC requests for resource adaptation. However, the SFCs have non-linear graphs of connected VNFs (5 VNFs each).

Scenario 3: we used this scenario to compare the effectiveness of our algorithms while receiving SFC chains of different sizes. For this reason, we set the number of incoming requests to 10 SFCs and varied the SFC sizes between 3 and 10 VNF instances. All SFCs in this scenario consist of linear graphs.

Scenario 4: in this scenario, we combined both linear and non-linear SFC requests. We varied the number of requests between 4 and 20 chains of 5 VNFs each, with approximately 50% of them being linear SFCs and 50% non-linear. Table 2.5 illustrates detailed results of performance metrics for many test cases including the number of SFC requests received and the SLO delay threshold (D) in each. To represent more testing results and confirm our observations, we chose different test cases than previous scenarios.

All these scenarios were tested in the same test environment, including the physical and virtual network characteristics described in the previous sub-section. Note that our implementation is not limited to these scenarios, because our algorithms can accept a mixed set of linear and non-linear SFCs of different sizes. But we selected the testing scenarios described above to compare

Table 2.4 Latency Thresholds in scenarios 1, 2 and 3

Scenarios 1 & 2		**Scenario 3**	
Number of SFCs	**Thresh.**	**SFC Sizes**	**Thresh.**
5	0.25	3	0.4
10	0.45	5	0.45
15	0.5	7	0.7
20	0.6	10	0.9

Thresh = SLO latency threshold in seconds.

the performance of our algorithms easily and clearly. After performing many experiments with different latency thresholds in each study case, we adopted the thresholds given in Table 2.4 for scenarios 1, 2 and 3, and Table 2.5 for scenario 4. For scenario 3, we had to set higher SLO thresholds than other scenarios, because the SFC chains are longer (higher SFC size). In other words, due to the higher number of VNF instances connected linearly, the end-to-end latencies of SFCs increased and therefore we needed to increase the latency thresholds. In our tests, we also assumed that it takes one second to scale or migrate a VNF instance, but this input value can be changed easily. In the next sub-sections, we discuss the experimental results obtained in the testing scenarios and illustrated by figures 2.2 to 2.7 and Table 2.5. Note that in these figures, scenario 1 corresponds to the left-side figure, scenario 2 to the middle one, and scenario 3 to the right-side one.

2.6.3 Results and discussion

2.6.3.1 Execution time

Through the experiments, we compare the runtime taken by the algorithms to find a solution for the resource adaptation requests. It is worth mentioning that for the meta-heuristic algorithms, the total time includes both (a) the population initialization process, and (b) the solution searching process over 100 generations. Based on the results obtained in Figure 2.2, we can see the big difference in time between ILP computation and all meta-heuristic algorithms. For some cases, while the ILP takes about hundreds and maybe thousands of seconds to generate a solution, the metaheuristics (specifically NSGA, CRO, and CRO-NBPSO) run for a few seconds and even less than 1 second to solve the problem. NBPSO has the highest execution time among the meta-heuristics. We mention that when we increased the SFC size to 10 VNFs in scenario 3, the ILP approach ran for a long time (many hours) in the case of 10 incoming requests, which prevents us from illustrating it in the graph. However, NSGA, CRO, CRO-NBPSO, and NBPSO could normally find a solution for this case in 2.54, 2.25, 2.7, and 36.15 seconds respectively. For this reason, we argue that the ILP approach is not suitable to solve the dynamic resource adaptation problem. A clearer view of NSGA, CRO,

and CRO-NBPSO runtime is given in Figure 2.3. According to their results, CRO has the lowest execution time in most cases. We argue that this is quite normal as the non-domination optimization feature of NSGA requires additional time for the searching process, which is not the case for CRO. Also, the high computation time taken by the NBPSO algorithm increases the runtime of the CRO-NBPSO.

Another aspect to clarify is the significant variation of the algorithm runtime, depending on the scenario. We can notice that a higher number of SFC requests did not necessarily result in a higher execution time. Time is affected by the SLO latency threshold specified in each case. Targeting a stressful latency threshold increases the execution time of the algorithms to search for a feasible solution and avoid SLO violation. We have selected a stressful latency in the case of 5 SFC requests for scenarios 1 and 2 to demonstrate this fact. Figures 2.2 and 2.3 show the high execution time of the algorithms when receiving 5 SFC requests and the targeted latency threshold is 0.25 seconds. Thus, a higher number of SFC requests, coupled with a low-pressure SLO threshold, can be adapted in a lower runtime than what is needed to adapt a few requests with a stressful one. Furthermore, in Table 2.5, the case of 8 SFC requests received is tested with two different latency thresholds (0.35 and 0.4 seconds). The time taken by the algorithms to find solutions when the threshold is 0.35 sec is higher than when it is 0.4 seconds. In other words, the execution time of the algorithms decreases while processing the same number of requests but with a higher latency threshold.

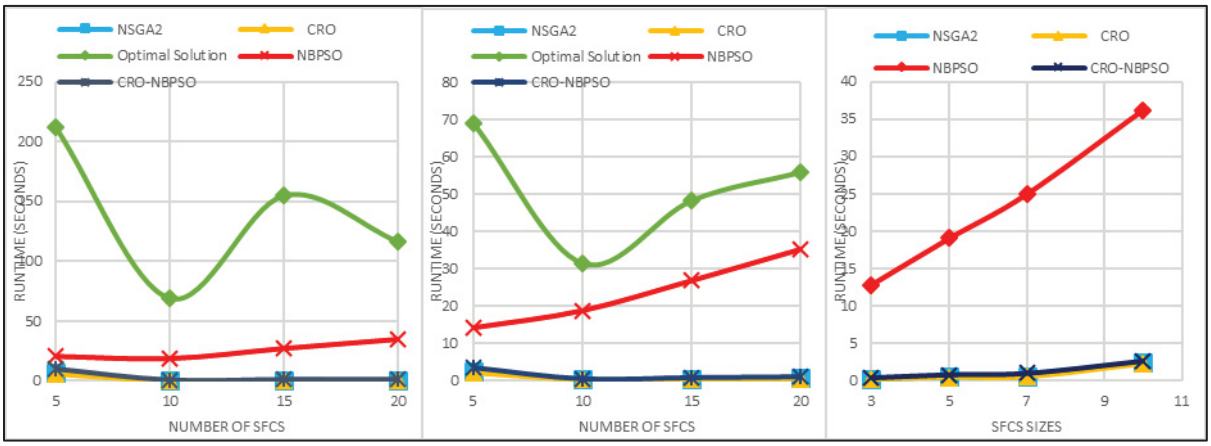

Figure 2.2 Average runtime of the NSGAII, CRO, NBPSO, CRO-NBPSO and ILP algorithms in scenarios 1 (left-side fig.), 2 (middle fig.) and 3 (right-side fig.), respectively

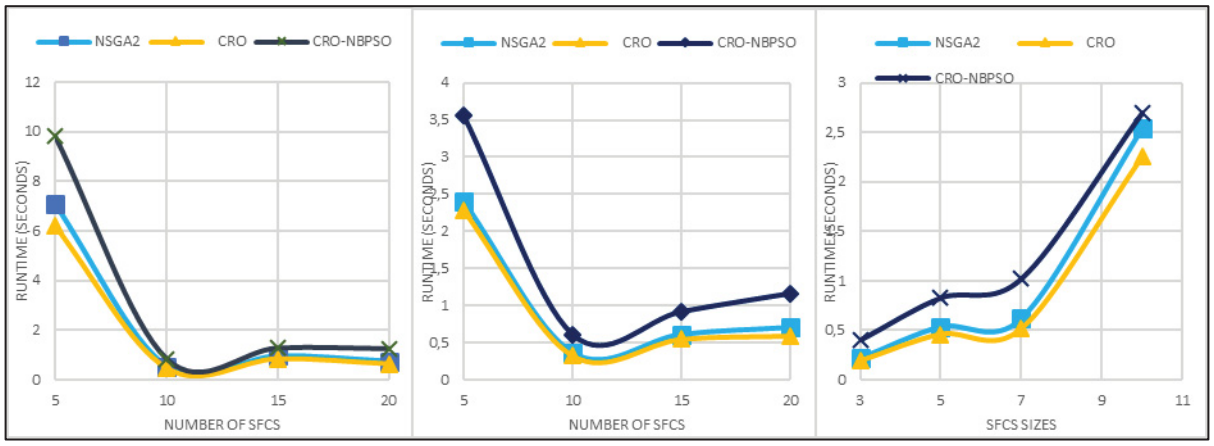

Figure 2.3 Clearer view of the average runtime for NSGAII, CRO and CRO-NBPSO in scenarios 1, 2 and 3, respectively

2.6.3.2 Solution accuracy

Besides the execution time, we compare the quality of the solutions generated by our five approaches in terms of the total CPU utilization, the energy consumption, and the end-to-end delay/latency. Section 2.4 already explained how we calculated these costs. In addition to the objective values, we analyze the number of servers used to adapt the resources in the solutions.

Regarding the CPU consumption, we can notice in Figure 2.4 the accurate performance of CRO, NBPSO, and CRO-NBPSO, as they generate solutions with CPU cost values very close to ILP solutions. They have approximately similar performance in terms of CPU consumption with slight differences. According to the detailed testing of scenario 4 in Table 2.5, NBPSO can achieve a little bit less CPU and power than CRO and CRO-NBPSO in almost all cases and even less than ILP when we increase the number of SFC requests (more than 4 SFCs). For its part, NSGA has higher CPU utilization, as compared to the other approaches. Similarly, CRO, NBPSO, and CRO-NBPSO exhibit approximately similar performance to the ILP model, while minimizing the energy consumption, as shown in Figure 2.5. Although NSGA exhibits the highest power consumption, it converges toward the CRO and ILP solutions while adapting 20 requests in scenarios 1 and 2 and adapting requests of 10 VNF instances in scenario

3. Table 2.5 also shows the high CPU and power consumption of NSGA compared to the other approaches in scenario 4. The difference in power consumption between NSGA and the others decreases with the increase in the number of requests. By adapting 20 SFCs, NSGA consumes 87.706% of power while ILP, CRO, NBPSO, and CRO-NBPSO consume 86.724%, 86.948%, 86.653%, and 86.916% respectively.

Conversely to the previous objective functions, Figure 2.6 and Table 2.5 show the ability of NSGA to generate solutions with minimal latency cost versus ILP and other meta-heuristics. In the second place, CRO, NBPSO and CRO-NBPSO can achieve lower latencies than ILP in many test cases and similar latency in some cases. For instance, they can considerably overcome the ILP in the second test scenario in Figure 2.6 while increasing the number of non-linear SFC requests. Table 2.5 also shows many cases where these algorithms can achieve lower latency than ILP such as testing 4 and 16 SFCs. Comparing these three meta-heuristics, they have variable and competitive performance in terms of latency. CRO-NBPSO achieves lower latency than CRO in scenario 3-Figure 2.6 while increasing the SFC Size. NBPSO has sometimes slightly higher latency compared to CRO and CRO-NBPSO, but it converges towards them at the end. Table 2.5 also shows that NBPSO can converge and overcome the latencies provided by CRO and CRO-NBPSO when increasing the number of requests (e.g., 20 SFCs). Note that the better performance of the metaheuristics algorithms in terms of latency compared to ILP can be attributed to the help from the server selection strategy used in their implementation. This strategy is described in section 2.5, “Solution encoding”. In contrast, it is not the case in ILP, as nothing there prioritizes latency minimization.

Concerning the number of servers, we notice that all meta-heuristics usually use the same number as the ILP model in their solutions. However, according to Figure 2.7 and Table 2.5, NSGA sometimes involves an additional server, and rather, converges to operate the same number as the other approaches, while increasing the number of VNFs to be adapted. It is important to mention that the server involved in many adaptation decisions (HS, VS, and M) was counted once. For example, migrating an instance does not necessarily increase the

number of servers used unless its destination sever has not been previously chosen for another decision.

To summarize, the experimental results confirm that the time to solve ILP models renders ILP not suitable for the dynamic resources adaptation of VNFs. On the other hand, they show that the meta-heuristic algorithms can provide solutions close to optimal in a significantly shorter amount of time. CRO first and NSGA second are the fastest in generating solutions while NBPSO is the slowest. In terms of objective functions, NSGA has a little bit higher resource and power consumption than others but can provide solutions with minimal end-to-end latency. CRO, NBPSO, and CRO-NBPSO perform close to ILP in terms of CPU and energy consumption while satisfying the latency tolerance threshold. They also achieve the same latency as ILP in some cases and lower latency in other cases depending on the testing scenario and the number of VNF resources to adapt. NBPSO provides slightly lower CPU and power consumption than CRO and CRO-NBPSO. It also has a little bit higher latency compared to them in some cases, but it converges towards them and can overcome them in terms of latency as the size or number of requests increases. Accordingly, depending on the service provider's optimization goals and priorities, the more suitable algorithm can be selected for resource adaptation. For example, if the priority is to strictly achieve the least end-to-end latency, NSGA can better serve this objective. If the priority is to reduce the resource utilization and energy expenditure as much as possible while meeting the latency tolerance threshold agreed in the SLO, other algorithms such as CRO, NBPSO or CRO-NBPSO can be used while considering their runtime.

Table 2.5 Scenario 4 test results

Test case	Performance metrics	ILP	NSGAII	CRO	NBPSO	CRO-NBPSO
4 SFCs D = 0.25 secs	CPU (%)	24.857	28.142	25.589	25.345	25.590
	Power (%)	48.995	55.425	52.424	52.342	52.424
	End-to-end latency (secs)	0.25	0.118	0.246	0.246	0.244
	Execution time(secs)	38.433	1.877	1.210	10.496	2.170
	Number of servers	14	15	14	14	14
8 SFCs D = 0.35 secs	CPU (%)	39.174	42.099	39.382	38.190	39.174
	Power (%)	66.128	68.953	66.222	65.871	66.167
	End-to-end latency (secs)	0.35	0.229	0.341	0.35	0.349
	Execution time(secs)	47.426	1.771	1.424	15.225	2.328
	Number of servers	17	18	17	17	17
8 SFCs D = 0.4 secs	CPU (%)	38.812	41.894	38.879	38.121	38.949
	Power (%)	66.017	69.204	66.077	65.844	66.101
	End-to-end latency (secs)	0.4	0.243	0.384	0.4	0.392
	Execution time(secs)	23.186	0.528	0.366	9.720	0.660
	Number of servers	17	18	17	17	17
12 SFCs D = 0.45 secs	CPU (%)	51.073	54.1195	51.036	50.101	50.991
	Power (%)	72.967	74.568	73.002	72.724	72.993
	End-to-end latency (secs)	0.44	0.311	0.424	0.433	0.426
	Execution time(secs)	48.608	0.539	0.486	20.762	0.926
	Number of servers	18	18	18	18	18
16 SFCs D = 0.55 secs	CPU (%)	62.907	66.328	63.576	62.476	63.445
	Power (%)	79.779	81.200	80.061	79.740	80.019
	End-to-end latency (secs)	0.55	0.361	0.531	0.523	0.522
	Execution time(secs)	58.050	0.69	0.556	29.025	1.014
	Number of servers	19	19	19	19	19
20 SFCs D = 0.6 secs	CPU (%)	75.078	78.124	75.571	74.507	75.456
	Power (%)	86.724	87.706	86.948	86.653	86.916
	End-to-end latency (secs)	0.55	0.383	0.539	0.533	0.544
	Execution time(secs)	66.918	0.756	0.628	35.282	1.582
	Number of servers	20	20	20	20	20

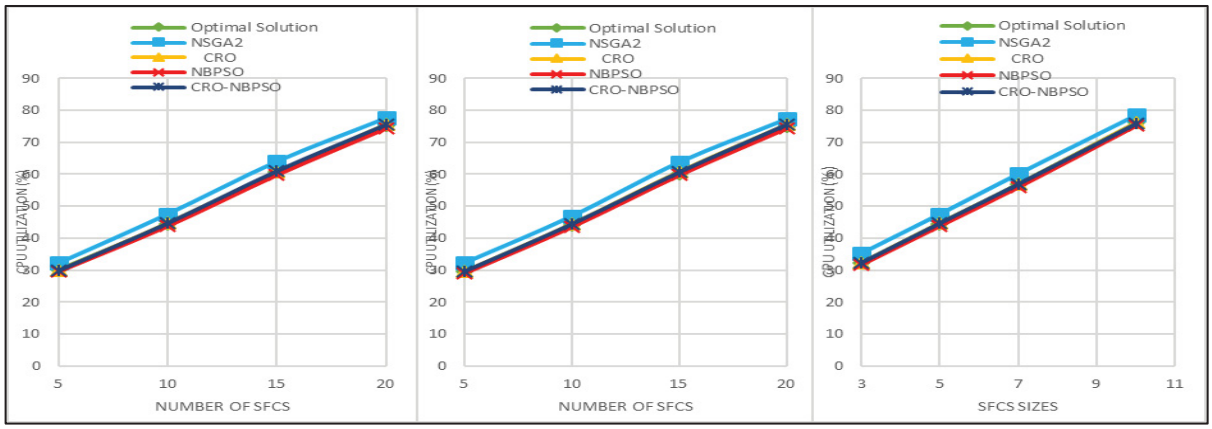

Figure 2.4 Comparison of the average CPU utilization for NSGAII, CRO, NBPSO, CRO-NBPSO and ILP in scenarios 1, 2 and 3, respectively

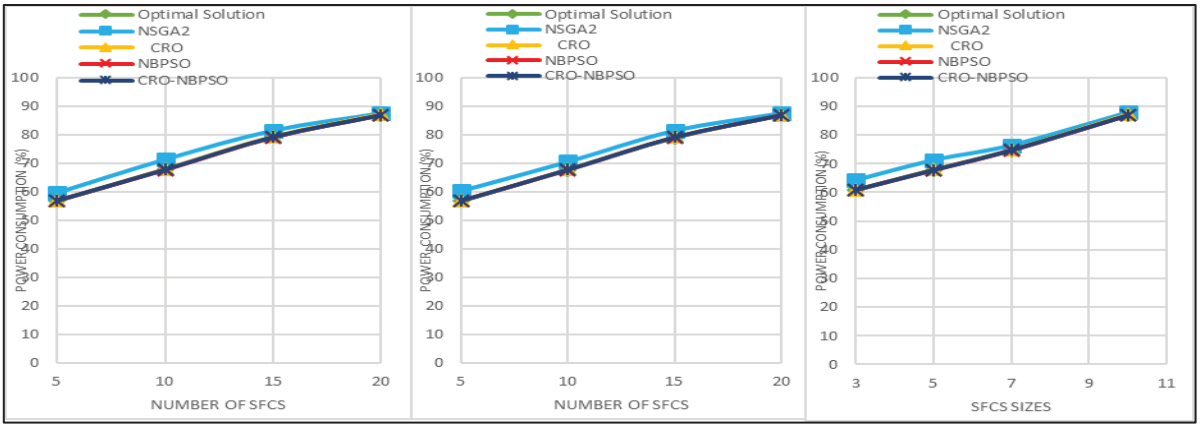

Figure 2.5 Comparison of energy consumption for NSGAII, CRO, NBPSO, CRO-NBPSO and ILP in scenarios 1, 2 and 3, respectively

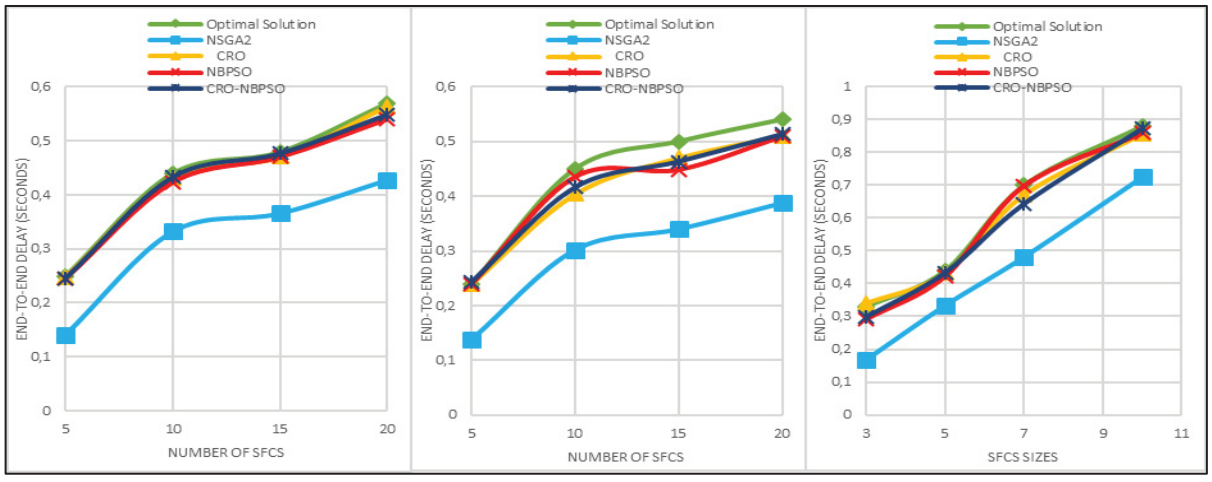

Figure 2.6 Comparison of the average end-to-end delay/latency for NSGAII, CRO, NBPSO, CRO-NBPSO and ILP in scenarios 1, 2 and 3, respectively

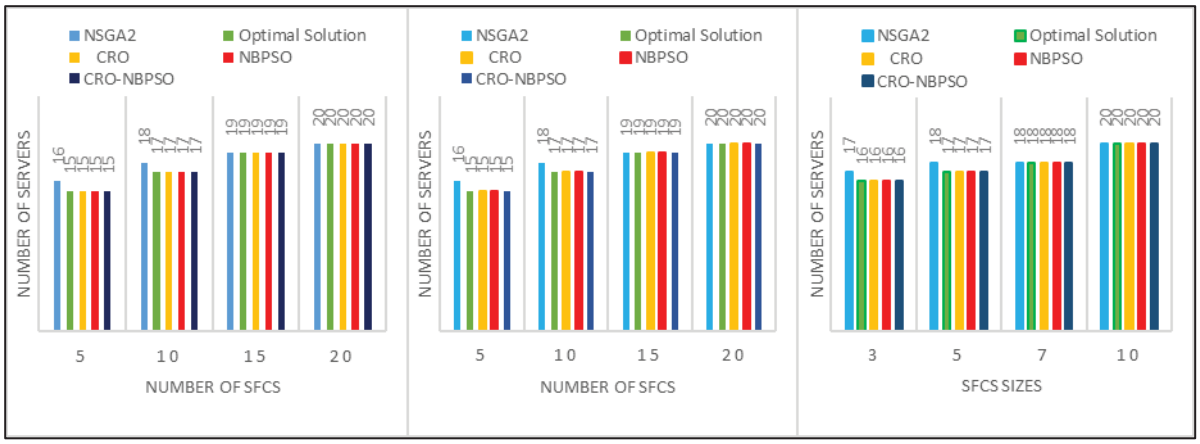

Figure 2.7 Comparison of the average number of servers used by the NSGA, CRO, NBPSO, CRO-NBPSO and ILP in scenarios 1, 2, and 3, respectively

2.7 Conclusion

In this article, we target the issues related to dynamic resource adaptation for service chains hosted in the cloud. We have proposed multi-objective metaheuristic-based approaches to dynamically find the optimum resource adaptation decisions that satisfy service needs while minimizing the cost metrics. These approaches are based on the Genetic algorithm NSGAII, Chemical Reaction Optimization (CRO), Binary Particle Swarm Optimization (BPSO), and the combination CRO_BPSO. Our proposal enjoys several advantages, namely: (1) it considers different potential resource adaptation techniques (Horizontal scaling, Vertical Scaling, and Migration) and dynamically selects the best that fulfills request demands according to traffic fluctuations; (2) its idea is generic enough to be used in any virtualized system or data center architecture; (3) it tackles SFCs of different sizes and topologies, including linear and non-linear ones; (4) it finds a trade-off between a set of cost metrics, such as energy consumption, SLO violation, and CPU utilization.

The experimental results show the effectiveness of the proposed algorithms in generating convenient adaptation solutions for incoming requests according to the network environment and imposed constraints. Additionally, the meta-heuristic algorithms demonstrate their ability

to approximate the ILP performance with a much shorter runtime. Concerning the metaheuristics techniques, we notice several observations. CRO had a lower execution time than other meta-heuristics in almost all cases while NBPSO had the highest. CRO-NBSO and NBPSO had competitive efficiency in terms of objective functions versus CRO. NSGAII sometimes had a little bit higher CPU utilization and Power consumption than other meta-heuristics. However, it could find feasible solutions with the shortest end-to-end delay compared to them and ILP. We also notice that the pre-specified SLO latency threshold had a significant influence on the results, especially on the algorithm's runtime.

For future work, we intend first to optimize our algorithms and ILP model to support multi-resources such as memory, disk, I/O, etc. instead of CPU only specifically in our constraints and Power consumption computation. Second, we will extend our algorithms by targeting more cost metrics, such as network bandwidth utilization, and constraints related to VNF dependencies that may be agreed to in the SLA (e.g., affinity and anti-affinity constraints). When two consecutive VNFs of the same SFC have a common affinity constraint, it means that these VNFs exchange a lot of loads and need to be located on the same server. In contrast, an anti-affinity constraint between two VNFs means that they are intensive-load VNFs, and therefore need to be hosted on different servers. Considering such constraints and metrics will influence the resource adaptation decisions suggested by our algorithms. Third, we want to test our algorithms while combining them with other VNF placement strategies to select destination servers for the adaptation decisions taken. Finally, we intend to combine our algorithms with a resource prediction technique to anticipate future resource demands and avoid delays in resource adaptation.

Acknowledgment

Special thanks to Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC) for their support.

CHAPTER 3

UTILIZATION PREDICTION-BASED VM CONSOLIDATION APPROACH

Mirna Awad [a], Nadjia Kara [a], and Aris Leivadeas [a]

[a] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Published in *Journal of Parallel and Distributed Computing*, August 2022.
(https://doi.org/10.1016/j.jpdc.2022.08.001)

3.1 Abstract

Reducing energy consumption and optimizing resource usage in large cloud data centers is still an essential target for the current researchers and cloud providers. The state-of-the-art highlights the effectiveness of VM consolidation and live migrations in achieving reasonable solutions. However, most proposals consider only the real-time workload variations to decide whether a host is overloaded or underloaded, or to trigger migration actions. Such approaches may apply frequent and needless VM migrations leading to energy waste, performance degradation, and service-level agreement (SLA) violations. In this paper, we propose a consolidation approach based on the resource utilization prediction to determine the overloaded and underloaded hosts. The prediction method combines a Kalman filter and support vector regression (SVR) to forecast the host's future CPU utilization. Simulations are conducted on Cloudsim using real PlanetLab workloads to verify the performance of our proposal against existing benchmark algorithms. Experimental results demonstrate that our consolidation technique significantly reduces the SLA violation rate, number of VM migrations, and energy consumed in the datacenter.

Keywords: Utilization prediction, Kalman filter, Support vector regression, VM consolidation, Cloud computing.

3.2 Introduction

The large-scale deployment of cloud data centers has led to a dramatic increase in energy consumption and environmental pollutants. Implementing effective resource management in the Cloud has become an increasingly urgent issue to be addressed. The challenge consists of the dynamic workload fluctuations of the running applications which in turn causes a continuous variation in resource utilization of virtual and physical machines. Thus, the main objective in this research field is to reduce the energy consumed by these data centers and utilize their physical resources efficiently, while maintaining the performance of the hosted applications. According to the state-of-the-art, virtual machine (VM) consolidation is one of the most productive methods that can achieve this goal (Helali & Omri, 2021a)(Chaurasia, Kumar, Chaudhry, & Verma, 2021)(Zhou et al., 2020). At the same time, using live VM migration techniques (F. Zhang, Liu, Fu, & Yahyapour, 2018)(Noshy, Ibrahim, & Ali, 2018)(Silva Filho et al., 2018), the VM consolidation strategy can be reinforced by hosting the active VMs on fewer physical machines (PMs) and switching off idle servers. This VM reallocation strategy should be employed dynamically to effectively redistribute load among the servers, enhance the cloud resources utilization and meet the time-varying resource requirements of the applications.

The main concern when jointly addressing the VM consolidation and VM migration is to decide whether a host is overloaded or underloaded depending on the real-time VM workload fluctuations. Migrating VMs from overloaded hosts helps reduce SLA violations. Whereas, migrating VMs from under-utilized servers and then switching them into sleep mode avoids energy waste. Several proposals have considered only the current host state and resource utilization to take such decisions and trigger VM migrations. These approaches may lead to an aggressive consolidation with frequent needless migrations. Excessively migrating the virtual machines from one host to another to satisfy their resource demands may affect their performance due to additional delays such as migration time and downtime. This decrease in Quality-of-service (QoS) may violate the service level agreement (SLA) and cause some penalties.

Consequently, more efficient solutions are needed to correctly make decisions regarding VM migrations. In other words, overloaded and underloaded hosts should be reliably determined to limit the frequency of VM migrations. Simple reactive methods that consider only the current CPU utilization may lead to unreliable VM migration decisions. Sometimes, the CPU utilization of the server may exceed a max threshold at the current time, but the load rapidly decreases in the next time slots. In such a situation, the host should not be considered overloaded and there is no need to migrate VMs from this server to release resources. Figure 3.1 borrowed from (Hieu, Francesco, & Yla-Jaaski, 2020) illustrates an example of a CPU utilization trace of a server measured every 5 minutes over 24 hours. If we assume that the max threshold is set to 80%, we can recognize many false overloading detection points (marked by small circles) in which the reactive approach will make inefficient VM migration decisions and eventually increase costs. For this reason, it is crucial to consider both current and future utilizations of the server before taking such decisions. In this paper, we propose a prediction-based VM consolidation approach that predicts the future host state and assigns VMs to hosts based on both current and near-future resource utilization. For example, in the trace of Figure 3.1, our method will report an overloading state for the server only in the period between 600 and 670 minutes (marked by a rectangle) because its utilization exceeds the max threshold in both the current and future period of time.

To do so, in this work, a multi-step prediction model that combines Kalman filter with Support Vector Regression (SVR) is used to forecast the host's future CPU utilization. Then, a VM consolidation framework based on prediction is presented to decide the host state (overloaded or underloaded) and then performs the required VM migrations. Our main contributions can be summarized as follows:

1. An efficient multi-step-ahead workload prediction method called K-SVR based on Kalman Filter and Support vector regression to forecast the host's future CPU utilization is proposed. Kalman filter is integrated for data pre-processing to reduce prediction error.

2. A dynamic VM consolidation approach is combined with the proposed prediction model, which consists of host overloading and underloading detection algorithms. In particular, the workload prediction model is distributed on all hosts to dynamically anticipate the future utilizations of each host according to its historical data. Then, overloading and underloading detection algorithms decide whether a host is underloaded or overloaded depending on its current and predicted utilization. According to the decisions taken about the host state, VM live migrations are performed.
3. A trade-off between SLA violation and energy consumption is pursued while minimizing the number of VM migrations and the runtime of resources allocations.
4. A comparative study is provided about the impact of prediction windows size (WS) on the performance of our consolidation approach in terms of SLA violation and energy consumption. This analysis shows the possibility to select the value of WS according to the deployment constraints or the objective priorities of the service provider. The service provider may aim for minimum power consumption, or to reduce SLA violation as much as possible, or save energy while satisfying a tolerance threshold agreed upon in the SLA. Host overloading and VM migrations may degrade the performance of the service and violate some agreements related to the quality of service and service availability. Respecting these constraints is mandatory to avoid penalties.
5. An analysis study is carried out on the energy consumption resulting from the execution of the simulation of our consolidation approach and our prediction technique and compared to existing benchmarks. Energy measurements are performed using the JoularJX tool(Noureddine, 2022).
6. Several simulations are conducted on Cloudsim using PlanetLab real-world workloads to validate the effectiveness of our proposed approach compared to the existing state-of-the-art algorithms.

The rest of the paper is organized as follows: Section 3.3 reviews the literature on resource utilization prediction and dynamic VM consolidation in the cloud. Section 3.4 explains in detail the prediction model proposed. Section 3.5 describes the utilization prediction-aware consolidation approach and analyses its time complexity. Section 3.6 presents our experimental setup, discusses the results obtained and compares the performance of our proposal against existing approaches. Finally, section 3.7 summarizes the paper and highlights our future directions for this work.

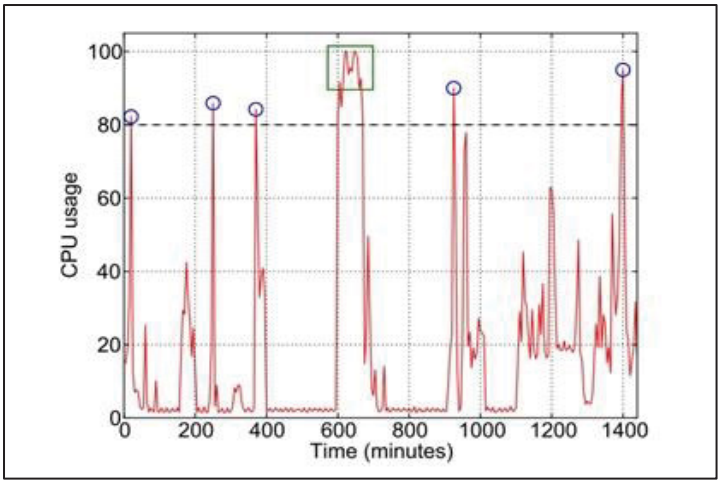

Figure 3.1 CPU utilization trace of a cloud server
Taken from Hieu et al. (2020, p. 190)

3.3 Related work

Resource management and reallocation in Cloud is a wide research problem that has been divided into sub-problems and addressed by researchers from various perspectives (Silva Filho et al., 2018)(Awad, Kara, & Edstrom, 2022) (B, Gounaris, & Sioutas, 2016) (Nadgowda, Suneja, & Kanso, 2017)(Taherizadeh & Stankovski, 2018)(Nadgowda, Suneja, Bila, & Isci, 2017)(Hoseinyfarahabady, Taheri, Zomaya, & Tari, 2021).

VM consolidation is a resource reallocation problem that relies on replacing VMs in the physical infrastructure in order to satisfy their resource needs while consolidating their workloads on a fewer number of physical machines. Research works tackle the VM consolidation problem from various perspectives such as VM placement (Bharanidharan &

Jayalakshmi, 2021)(Zhihua Li, Yu, Yu, Guo, & Chang, 2020)(Silva Filho et al., 2018), overloading and underload detection (Beloglazov & Buyya, 2012) (Hsieh et al., 2020), VM selection for migrations (Melhem, Agarwal, Goel, & Zaman, 2017)(Moghaddam, Piraghaj, O'Sullivan, Walker, & Unsworth, 2018) etc. In our paper, we focus mainly on host overload and underload detection. This decision-making problem is challenging and mainly sensitive to the workload type and its variation over time. Several proposals decide whether a host is overloaded or under-loaded by comparing its current resource utilization to two static thresholds (hot and cold thresholds). For example, XIAO et al. (Xiao et al., 2019) present a VM consolidation method based on thresholds and an Ant Colony system. The upper threshold is set to 80% and the lower to 40%. If the current CPU utilization exceeds 80%, the host is overloaded. Whereas, if it is lower than 40%, the host is considered under-utilized. Ant Colony system is used to map the migrated VMs to appropriate destination hosts. Other researchers suggest the usage of dynamic thresholds instead of static ones. According to these approaches, static thresholds are not effective methods to deal with the workload dynamicity in cloud data centers. For instance, Beloglazov et al. (Beloglazov & Buyya, 2012) present two adaptive threshold methods based on a statistical analysis of the host historical data, namely, Interquartile Range (IQR) and Median Absolute Deviation (MAD). In this case, the current utilization of the host is compared to thresholds computed dynamically, to evaluate its state over time. On the contrary to these approaches, several works propose predictive dynamic VM consolidation algorithms. Indeed, relying on the current host state only to make decisions regarding VM migrations may lead to unreliable decisions with excessive needless migrations. Frequent VM migrations can affect their performance, impose additional delays and downtime, and therefore risk more SLA violations. Hence, prediction techniques are widely investigated in the pertinent literature to anticipate the future load variations (Benmakrelouf, Kara, Tout, Rabipour, & Edstrom, 2019 ; Qiu, Zhang, & Guo, 2016), (Abdullah, Li, Al-Jamali, Al-Badwi, & Ruan, 2020) and to adapt resources in the cloud according to the future resource demands (Radhika & Sadasivam, 2021). In (Amiri & Mohammad-Khanli, 2017) a detailed survey of application prediction schemes is presented, including the different prediction models proposed in the state of the art, their main characteristics, and challenges. In general, in predictive consolidation techniques, a workload forecasting model is used, and then VM live

migration is triggered considering the predicted host state, or both its current and future state. For example, Hsieh et al. (Hsieh et al., 2020) present a utilization prediction-aware VM consolidation strategy. In particular, a Gray model and Markov chain are combined to forecast the short-term future CPU utilization of hosts. Then, overloaded and under-utilized hosts are determined by comparing their current and estimated CPU utilization to given thresholds. A dynamic threshold based on MAD is used for overload detection, and a static one is applied to detect underload states. Li et al. (L. Li, Dong, Zuo, & Wu, 2019) employ an SLA and energy-aware consolidation process based on the Robust Simple Linear Regression (RobustSLR) prediction model. This model anticipates the host's future CPU usage, and it amends the prediction by adding the error directly or indirectly to the prediction value. If the historical data length is not sufficient to predict, the current CPU utilization of the host is compared to a static threshold to verify whether it is overloaded or not. Otherwise, the predicted CPU utilization is adopted to make the decision. Ding et al. (Ding et al., 2020) propose a performance-to-power-ratio (PPR) aware VM consolidation approach. Their short-term workload prediction model consists of the moving average (MA) and the interquartile range (IQR) techniques. Host overload detection is based on the available residual computing capacity (RACC) evaluation model. The RACC of a host is calculated based on its power consumption and PPR. The RACC of the host with the maximum PPR is called optimum RACC and is used as the threshold to detect overload. That is to say, a host is considered as overloaded if its RACC is lower than the optimum RACC. Underload detection relies on a multi-criteria Z-score algorithm in which a score is calculated for each host based on its CPU utilization and PPR, and so the host with the highest score (minimum CPU usage and PPR) is declared as under-utilized. Unlike resource utilization prediction methods, Li et al. (L. Li et al., 2018) rather predict the future host states (overloaded or not) using Naïve Bayesian classifier. However, underloaded servers are simply selected as the hosts with the minimum utilization compared to the others. Their main target is the reduction of SLA violation level and power costs. Mahdhi et al. (Mahdhi & Mezni, 2018) suggest predicting the host states based on the resource utilization history of VMs and the past VM migration traffic. They create a weighted graph to model the history of migration traffic between hosts. A Kernel Density Estimation (KDE) technique is used to forecast the future resource usage of each VM, and an AKKA framework is adopted as actor-

based to allow exchanging information about the host's states. The host is detected as overloaded when one of its estimated requested resources (CPU, RAM, and Storage) exceeds the available capacity. Otherwise, it is considered under-loaded. The new placement for the migrated VMs is then decided according to the migration history. Shao et al. (Shao et al., 2020) discuss a dynamic VM consolidation technique based on a Gray model and an improved discrete particle swarm algorithm (GM-DPSO). Their objective is to minimize energy consumption, SLA violations, and number of migrations. The Gray prediction model is used for load detection, while an improved discrete particle swarm algorithm is employed for the VM placement process. A host is determined to be overloaded if both the current load rate and the predicted load rate are higher than the threshold. An adjustment mechanism based on MAD is adopted to dynamically modify the upper threshold according to the load condition. Regarding underload detection, the host with the lowest utilization rate is selected at each time step. Finally, Witanto et al. (Witanto, Lim, & Atiquzzaman, 2018) implement an adaptive selector based on neural network to select the appropriate consolidation technique adaptively according to the cloud provider's goal priority and environment parameters. It chooses between the following four classes: (1) No migration; (2) Migration only for underloading; (3) Migration for underloading and overloading, in this case, local regression (LR) is used for detection and Guazzone for VM placement; (4) Migration for underloading and overloading, using LR for detection and Shi-AC for VM placement. In contrast to the predictive consolidation approaches previously discussed, Hieu et al. (Hieu et al., 2020) estimate the long-term resource utilization of servers using multiple linear regression. They define a server as overloaded if its multiple predicted resource usage exceeds a hot threshold h ($U_{t+k} > h$); and as underloaded if its multiple predicted resource utilization is equal or less than its current resource usage ($U_{t+k} \leq U_t$). Sometimes, the workload trace of a host may occur false hot increase detection points, and the load rapidly decreases in the short-term future. For this reason, adopting a multi-step or a long-term prediction model to anticipate a sequence of future host utilization values can be helpful to avoid unreliable decisions. Khoshkholghi et al. (Khoshkholghi, Derahman, Abdullah, Subramaniam, & Othman, 2017) develop an iterative weighted linear regression method that uses two utilization thresholds to detect overload situations. This strategy does k iterations to find K future values of host utilization. A host is

declared as overloaded if its predicted utilization at step t+1 is higher than the total capacity (100%). However, if other future values in host utilization (steps t+2 to t+k) are detected to be higher than the total capacity, the host is considered under pressure and does not accept any new VM. For underloading part, they propose an algorithm using vector magnitude squared of multiple resources. The host is underloaded when its CPU, RAM, and BW utilizations are lower than a threshold. The lower threshold is calculated as the lower quartile of the previous host utilizations.

Resource reallocation and workload prediction are not limited to VMs. With the remarkable evolution of Cloud computing solutions for hosting services, from the usage of virtual machines (VMs) to containers, and recently to serverless platforms, more resource management mechanisms are needed and still under study. Serverless platform benefits from many advantages over the IaaS cloud (Rajan, 2018)(McGrath & Brenner, 2017). First, it allows users to deploy highly scalable containerized services decomposed into a workflow of event-driven stateless functions. Second, it offloads all management responsibilities to the cloud provider, including resource provisioning, scheduling, scaling, etc., so organizations and developers do not have to worry about these issues. Third, unlike the IaaS cloud where users are charged for the number of rented VMs and their resources even if they are idle or unused, in serverless platforms users are charged granularly for the compute and storage resources needed to execute their computing tasks. This fact reduces the hosting costs but complicates the billing estimation. Serverless billing models are multi-dimensional because a set of functions are deployed individually and executed over heterogeneous resources. This performance variance exhibits costs variance which motivates some researchers to introduce tools for performance characterization, runtime prediction, and workload costs estimation for Function-as-a-Service (FaaS) platforms (Cordingly, Shu, & Lloyd, 2020). In (Apostolopoulos, Tsiropoulou, & Papavassiliou, 2019), the authors propose a distributed approach to determine the optimal resource allocation strategy for users by helping them decide whether to offload their computing tasks to virtual machines and/or serverless functions offered in the social cloud computing environment. In (Bhattacharjee et al., 2019), the authors present a distributed and scalable resource management system called Barista for serverless computing, specifically for

deep learning prediction services. The objective is to manage the compute resources for these services while maintaining their SLO requirements (e.g., required prediction latency). Their methodology includes: (a) an online rolling window based forecasting technique to predict workload based on historical data; (b) a greedy heuristic to identify suitable compute resource configuration; (c) an intelligent agent to proactively scale container resources horizontally and vertically based on predicted workload. In (Ali, Pinciroli, Yan, & Smirni, 2018), the authors focus on burstable performance instances, which are low-cost instances that can ramp up their CPU performance using spare resources to treat a burst of heavy load for certain amounts of time. These instances are typically used for applications like microservices that do not require consistently high computational power but may need higher computational power from time to time to satisfy a heavy load for a short time. In their work, they present an autonomic scheduling framework for burstable performance instances that can maximize the efficiency of the spare resources' usage while meeting SLOs. In (Saha & Jindal, 2018), A resource management system for serverless cloud computing is suggested with the goal to improve memory allocation among containers. They built their solution on top of OpenLambda, an open-source serverless platform.

In this article, we propose a novel multi-step-ahead workload prediction model that combines Support Vector Regression with Kalman filter. Kalman is used as data preprocessing step to improve the prediction accuracy. This proposal can be used to forecast any workload whether it is received by servers, VMs, containers, microservices, serverless functions, etc. In this work, we adopted and tested this prediction model to forecast the incoming workload of the servers for the purpose of estimating their future states. In addition, a predictive consolidation framework is represented to proactively make the necessary resource management decisions and ensure that the SLOs are met. This framework is based on Overload and underload detection algorithms that consider both the current and the set of predicted CPU utilization values of hosts while making decisions. Moreover, we present a comparative study about the selection of window size value according to the deployment constraints (e.g., meet the required quality of service, avoid crossing tolerance thresholds agreed in SLA) and the cloud provider's objective priority (save energy or decrease SLA violation). Hence, we aim to find the best

trade-off between energy consumption and SLA violation, while minimizing the number of VM migrations and the execution time of the resource re-allocation process. Furthermore, an analysis study is performed to compare the energy consumption resulting from running the simulation of our consolidation approach against existing benchmarks.

Serverless platforms allow users to deploy highly scalable and event-driven applications that can be decomposed into a set of short-running and stateless functions. These functions are typically hosted on a set of containers running on a cluster of VMs. For example, Kubernetes host containerized applications into PODs which are allocated to nodes cluster. A POD consists of a group of containers, while a node can be a virtual machine. Thus, A workload consolidation technique or resource management requirements may involve the need of migrating workloads from one VM to another in the same cluster, and even from one cluster to another. The proposed consolidation framework can be optimized to support the migration of PODs hosting microservices or serverless functions from one VM to another while taking into account the deployment constraints and SLO metrics of these functions.

3.4 Prediction strategy

In this section, we introduce a novel multi-step-ahead CPU utilization prediction model called K-SVR that combines Support Vector Regression (SVR) and Kalman Filter to proactively forecast the future CPU utilization of servers hosted in the cloud. We argue that both selected techniques are suitable for the dynamic cloud environment and the fluctuant loads in resource usage of hosted applications. Kalman Filter (Kalman, 1960)(Zhang-Jian, Lee, & Hwang, 2013)(Kalyvianaki, Charalambous, & Hand, 2014) is originally developed to estimate time-varying states in dynamic systems and is suitable for Cloud application's load estimation. In our work, Kalman is adopted as a data pre-processing step to enhance prediction accuracy. It is used to filter the CPU historical data H_CPU_s of the host s as shown in the pseudocode of Algorithm 3.1. Then, the data is divided into training and testing sets and SVR is used to perform multi-step ahead CPU load prediction. WS represents the training data size and n is the number of prediction steps. At each time t, we predict n CPU utilization values for each

host using its historical data. Forecasting multiple values allows us to estimate the future trend in the CPU consumption of hosts and make the necessary resource management decisions before encountering serious issues (e.g., SLA violation). In the following, we introduce the working principles of the two aforementioned techniques.

3.4.1 Kalman Filter

Kalman Filter essentially provides estimates of unknown variables using a set of measurements observed over time. The evolution of the state x from time $K-1$ to time K is defined in (1).

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.1}$$

This process model is paired with the measurement model given in (2).

$$z_k = Hx_k + v_k \tag{3.2}$$

where A is the state transition matrix from step $k-1$ to k. B is a matrix that relates the optional control vector u to the state x. H is a matrix that describes the relation between the state x_k and the measurement z_k. In our test experiment, there is no control input (B=0), and measurements are of the state directly (H=1). We assume that the state does not change from one time step to another (A=1). w_{k-1} and v_k are white noises and represent process and measurement noise respectively. They are random variables assumed to be independent of each other, with $w_{k-1} \sim N(0, Q)$ and $v_k \sim N(0, R)$. Q is the process noise covariance matrix, while R is the measurement noise covariance matrix. We assume that Q and R are constant.

The algorithm iteratively applies two phases of computation: prediction and correction. The prediction step projects the current state estimate ahead of time from $k-1$ to k. The correction phase adjusts the projected estimate by an actual measurement at that time.

Prediction phase

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.3}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.4}$$

Correction phase

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (3.5)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (3.6)$$

$$P_k = (I - K_k H) P_k^- \quad (3.7)$$

Where $\hat{x}_k^-$ represents the priori state estimate at time k, $\hat{x}_k$ is the posteriori state estimate at time k, P_k^- denotes the priori estimate error covariance matrix, P_k is the posteriori estimate error covariance matrix, and K_k is called the Kalman Gain matrix. Kalman gain illustrates the weight given to the measurements and the priori state estimate $\hat{x}_k^-$. A high gain means that the filter places more weight on the accurate measurements to estimate the state $\hat{x}_k$. Conversely, a low gain means that the state estimate mostly depends on the model predictions derived from the prediction phase $\hat{x}_k^-$.

3.4.2 SVR regression

Support Vector Machine (SVM) is a popular statistical learning technique widely used to solve classification problems in machine learning. Support Vector Regression (SVR) has the same principles as SVM, but it is developed specifically for regression problems. SVR is the methodology by which a function $f(x)$ is estimated using a dataset that trains the SVM. We treat CPU utilization prediction as a time series prediction problem. To forecast the future CPU values, the time series workload dataset is split into input and output vectors. Each input vector x_i represents a finite set of sequential CPU utilization measurements of these series. The output vector y_i includes the $x_{(n+1)}$ observation, where n denotes the amount of historical data. Each combination (x_i, y_i) is used as a training point. Eq. (3.8) defines the regression prediction function.

$$f(x) = w\emptyset(x) + b \quad (3.8)$$

Where $\emptyset$ is a mapping function (kernel function) to non-linearly map x from input space to multi-dimensional feature space. The Radial Basis Function (RBF) kernel is used for its easier computation and fewer parameters. $f(x)$ is the predicted value, w is a weight coefficient, and

b is a bias. The basic idea behind SVR is to construct a hyperplane (best-fit line) that has maximum data points. The key to achieve this goal is to find the flattest function that allows the error to remain within a threshold epsilon ε. The flatness of the weights can be measured by the Euclidean norm. Hence, the objective is to minimize the l2-norm of the coefficient vector (minimize $\frac{1}{2} \parallel w \parallel^2$). Moreover, the error or the empirical risk R_{emp} generated by the estimation process should be minimized. Thus, the overall objective is to reduce the regularized risk that combines the two sub-objectives previously explained as follows:

$$Minimize\ R_{reg} = \frac{1}{2} \parallel w \parallel^2 + R_{emp} = \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{N} L^{\varepsilon}\big(y_i, f(x_i)\big) \tag{3.9}$$

Where

$$L^{\varepsilon}\big(y_i, f(x_i)\big) = \begin{cases} |\, y_i - f(x_i) - \varepsilon| & if\ |\, y_i - f(x_i)| \geq\ \varepsilon \\ 0 & otherwise \end{cases} \tag{3.10}$$

R_{emp} is calculated using the ε-insensitive loss function L^{ε}. ε and C are user-defined parameters. C is a constant used to control the trade-off between the empirical and regularized risk. Finally, Slack variables, ξ_i and ξ_i^*, are introduced to estimate the errors for underestimation and upper estimation (above and below ε). Adding these variables optimizes the equations as displayed in (3.11) and (3.12).

$$Minimize\ R_{reg} = \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{N} (\xi_i^* + \xi_i) \tag{3.11}$$

$$Subject\ to \begin{cases} f(x_i) - w\phi(x_i) - b \leq \varepsilon + \xi_i^* \\ w\phi(x_i) + b - f(x_i) \leq \varepsilon + \xi_i \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{3.12}$$

Algorithm 3.1 K-SVR prediction algorithm

1: ***Input****:* H_CPU_s, WS, n
2: ***Output****:* $\widehat{CPU_s}(t+n)$
3: */* Preprocess data */*
4: *Filtered_history = Kalman_filter (*H_CPU_s*)*
5: */* extract and divide training dataset into X and Y*/*
6: *Training_data = extract (Filtered_history, WS)*
7: *Xtrain, Ytrain = divide (Training_data)*
8: */* Train SVM model */*
9: *svmTrain(Xtrain, Ytrain)*
10: */* extract testing data */*
11: *Xtest = extract (Filtered_history, n)*
12: $\widehat{CPU_s}(t+n)$ *= svmPredict (Xtest)*
13: ***Return*** $\widehat{CPU_s}(t+n)$

3.5 Prediction-aware consolidation approach

According to (Beloglazov & Buyya, 2012), VM consolidation consists of four essential steps: (a) overloaded hosts detection; (b) underloaded hosts detection; (c) VM selection for migration and (d) VM placement. The workflow diagram presented in Figure 3.2, illustrates step by step how our predictive consolidation approach works. Specifically, it is executed periodically to optimize the reallocation and consolidate the resources. Firstly, it starts by detecting the overloaded hosts based on their resource usage prediction (Algorithm 3.2). To maintain the necessary QoS of running services and avoid SLA violation, some VMs from those overloaded hosts are chosen for migration (Algorithm 3.4). Then, new destination hosts are selected for the migrated VMs (Algorithm 3.5). If no active host with enough capacity is found by the placement algorithm, an inactive host is initiated to place the migrated VM. On the other hand, the underutilized hosts are also identified (Algorithm 3.3) and all VMs on those hosts are migrated to others if possible (Algorithm 3.5). Then, the idle hosts are switched to a low-power state to save energy. In this way, the VMs are consolidated into a minimum number of active hosts. In this article, we focus mainly on the first two parts of the consolidation problem (steps a and b). Precisely, we present overloading and underloading detection algorithms based on the prediction strategy discussed previously in section 3.4. In this section, we describe the different algorithms that constitute our VM consolidation framework, then we analyze its time complexity.

3.5.1 Overload detection algorithm

The proposed overload detection technique is presented in Algorithm 3.2. It takes as input an active server $s \in S_{active}$, and outputs a Boolean decision to indicate whether this server is overloaded or not.

The algorithm starts by obtaining the recorded CPU utilization history H_CPU_s of the server, its current utilization $CPU_s(t)$, the prediction window size (WS), the number of prediction steps (n), and the upper threshold (O_TH). It is worth mentioning that the historical data is recorded at a 5-minute interval. To forecast $\widehat{CPU_s}(t+n)$, sufficient historical data is required to train the prediction model. If data is insufficient ($Length\ (H\_CPU_s) < WS$), the decision is taken based on the current utilization only. Thus, the host is considered overloaded if its current CPU utilization is higher than the threshold ($CPU_s(t) > O\_TH$) (Steps 3-10). Otherwise, $\widehat{CPU_s}(t+n)$ are predicted using our time-series prediction mechanism K-SVR presented in section 3.4. In this case, the host is considered overloaded if the average of the predicted n CPU values $Avg(\widehat{CPU_s}(t+n))$ is higher than O_TH (Steps 11-15). In another word, the server can be overloaded in two possible scenarios: (1) if it is overloaded in both the current and the future period of time (e.g., $CPU_s(t) > O\_TH\ \ and\ Avg(\widehat{CPU_s}(t+n) > O\_TH$); or (2) it is currently operating normally but will be overloaded in the future period of time (e.g., $CPU_s(t) < O\_TH\ \ and\ Avg(\widehat{CPU_s}(t+n) > O\_TH$). According to the overload decisions taken by this algorithm, migration actions will be taken. Precisely, VMs will be selected from overloaded hosts and migrated to new destinations by the algorithms described in the following sub-sections.

Algorithm 3.2 Overload_detection

1: ***Input***: $s \in S_{active}$
2: ***Output***: *Boolean decision if s is overloaded or not*
3: *Get* $CPU_s(t)$
4: *Get* H_CPU_s
5: *Get* WS , n , O_TH
6: ***if*** *Length (*H_CPU_s*)* < WS ***then***
7: ***if*** $CPU_s(t) > O\_TH$ ***then***
8: $return\ true$
9: ***end***
10: ***else***
11: $\widehat{CPU_s}(t+n)$ = *K-SVR(*H_CPU_s*,* WS*,* n*)*
12: ***if*** *Avg (*$\widehat{CPU_s}(t+n)$*)* > O_TH ***then***
13: $return\ true$
14: ***end***
15: ***end***
16: ***Return*** $false$

3.5.2 Underload detection algorithm

The main objective of algorithm 3.3 is to identify the underloaded servers. Their identification will help us minimize the number of active hosts by switching under-utilized servers to a low-power state. Thereby, the total energy consumption in the data center will reduce.

Its input includes a set of active hosts S_{active}, and its output is the detected under-utilized server. Note that the list of active hosts should exclude the overloaded hosts detected in algorithm 3.2. If no sufficient historical data is available for prediction, the under-utilized servers are simply those having minimum CPU utilization (Steps 6-9). Otherwise, the future n CPU values are predicted by our K-SVR technique described in section 3.4, while taking as input the CPU historical data of the server, the prediction window size (WS), the number of prediction steps (n), and the lower threshold (U_TH). Then, the server with an average of $\widehat{CPU_s}(t+n)$ lower than the threshold is considered underloaded (Steps 10-14).

Algorithm 3.3 Underload_detection

```
1:  Input: S_active
2:  Output: Underloaded server U_server
3:  Get WS , n , U_TH
4:  U_server = NULL
5:  Get Length (H_CPU_s) /* s ∈ S_active */
6:  if Length (H_CPU_s) < WS then
7:        U_server = min_Utilization_host(S_active)
8:        Return U_server
9:  else
10:    foreach s ∈ S_active do
11:       CPU_s(t + n) = K-SVR(H_CPU_s, WS, n)
12:       if Avg (CPU_s(t + n)) <= U_TH then
13:          U_server =s
14:          Return U_server
15:       end
16:    end
17: end
18: Return U_server
```

3.5.3 VM migration and placement

To move a VM from one host to another, we have adopted the pre-copy VM live migration algorithm. Migration time is calculated as the utilized RAM of VM divided by the available network bandwidth. The time taken to migrate a VM v can be formulated as $T_v^{migration} = \frac{RAM_v}{B_v}$.

For VM selection and placement, we have reused the techniques proposed by (Beloglazov & Buyya, 2012) to perform a formal comparison between our approach and theirs. Precisely, to select the VMs to be migrated from an overloaded host, the Minimum Migration Time (MMT) is used. MMT selects the VM that has the least migration time, as calculated by the equation above, for migration. According to the formula, the VM that has the least RAM will also have the least migration time. The VM selection algorithm is presented in Algorithm 3.4. After selecting the candidate VM for migration among the set of VMs V^s hosted on the server s, it

checks if this server will remain overloaded after deallocating the selected VM or not. The function overloadedAfterDeallocation will simulate the VM deallocation from the server and call our overload detection algorithm (Algorithm 3.2) to verify the server state. If the server *is* still overloaded, another VM will be selected for migration. The output is the list of VMs to migrate from the overloaded server $s \in S_{over}$.

Algorithm 3.4 MMT VM selection

```
1:  Input: s ∈ S_over
2:  Output: List vmsToMigrate
3:  While (true) do
4:      Set min_ram = MAX
5:      CandidateVM = NULL
6:      foreach v ∈ V^s do
7:          ram = v.getRam()
8:          if ram < min_ram then
9:              min_ram = ram
10:             CandidateVM = v
11:         end
12:     end
13:     vmsToMigrate.add (CandidateVM)
14:     if overloadedAfterDeallocation (s, v) = false then
15:         Break
16:     end
17:  end
18: Return vmsToMigrate
```

Once the VMs to be migrated are selected from all detected overloaded servers, the Power Aware Best Fit Decreasing (PABFD) placement strategy is applied to find a destination server for each migrated VM and then return the complete migration Map. PABFD in Algorithm 3.5 searches for each VM, a destination host that will be suitable in terms of resource capacity requirements (Step 8), will not become overloaded after hosting the target VM (Steps 9-11) and will have the least increase in its power consumption caused by this allocation. The function overloadedAfterAllocation simulates the allocation of the VM on the server and then calls our overload detection algorithm (Algorithm 3.2) to check its state after this allocation.

Algorithm 3.5 PABFD

```
1: Input: S_active, List vmsToMigrate
2: Output: Migration Map
3: foreach v ∈ vmsToMigrate do
4:     minPower = Max
5:     destinationServer = NULL
6:     foreach s ∈ S_active do
7:         if s.isSuitableForVM(v) then
8:             if overloadedAfterAllocation(v, s) then
9:                 Continue
10:            end
11:            oldPower = s.getPower()
12:            newPower = estimatePowerAfterAllocation(v, s)
13:            powerDiff = newPower – oldPower
14:            if powerDiff < minPower then
15:                minPower = powerDiff
16:                destinationServer = s
17:            end
18:        end
19:    end
20:    if destinationServer is not NULL then
21:        migrationMap.add(v, destinationServer)
22:    end
23: end
24: Return migrationMap
```

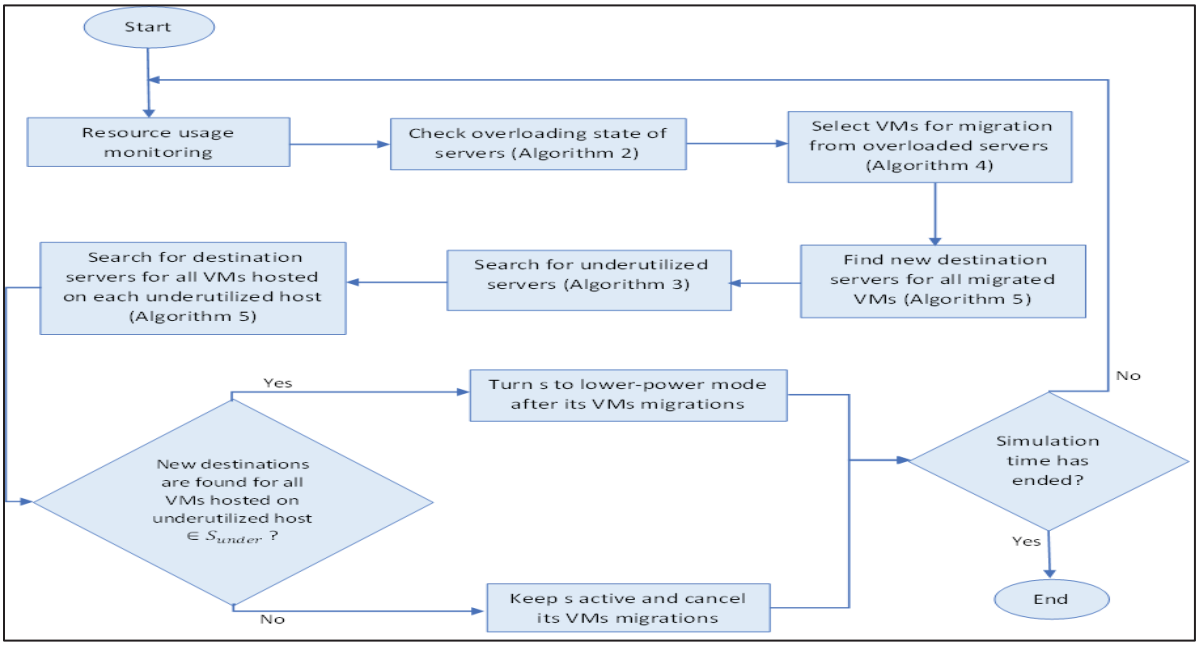

Figure 3.2 Workflow diagram of the proposed VM consolidation approach

3.5.4 Complexity Analysis

The overall VM consolidation algorithm is described in the diagram in Figure 3.2 and is presented in detail by Algorithm 3.6. It consists of two phases: (1) VM migrations due to overloading detection (Steps 2-8); and (2) VM migrations due to underloading detection (Steps 9-25). In the first phase, the framework detects the overloaded servers, selects the VMs to migrate from these servers, and constructs the migration map by finding new destination servers for the migrated VMs. We have explained the algorithms responsible for each of these steps in the previous sub-sections. Once overloading solutions are found, we move to the second phase of underload detection. We exclude from our searching list, the overloaded servers and the destination servers chosen for migrated VMs in phase 1 because those servers cannot be turned off. Each detected under-utilized server is switched to low-power mode if and only if all its running VMs can be migrated to other destination hosts. In the following, we discuss the complexity time of each phase. Let us define A as the number of active servers S_{active} in the system; N as the total number of VMs in the datacenter; N_{V^s} as the number of VMs hosted on a server s; $N_{V^{migrate}}$ is the number of VMs selected for migration; and H_s as the CPU historical data length of a server.

3.5.4.1 Complexity –Phase 1

Starting with line 2, the time complexity of the for loop is the same as the number of active hosts $O(A)$.

Inside the loop, algorithm 3.2 of overload detection is called. Its complexity is mainly based on the complexity of K-SVR prediction approach (Algorithm 3.1). According to (Valade, Acco, Grabolosa, & Fourniols, 2017), the Kalman filter complexity is $O(4n^3)$ where n is the state vector size. As we are using kalman to filter or preprocess the CPU historical data, its complexity in our case is $O(4{H_s}^3)$. In (Abdiansah & Wardoyo, 2015), a time complexity analysis of Support Vector Machine in LibSVM is provided. LibSVM is the library that we

used to implement the SVR prediction part. Their analysis shows that the worst complexity for svmPredict and svmTrain is $O(n^3)$ where n is the amount of data. In our work, the number of training dataset is WS and the number of prediction steps is n. Thus, the K-SVR complexity is $O(4{H_s}^3 + WS^3 + n^3)$. In line 5, Algorithm 3.4 is called to select the VMs to migrate from the server s. This algorithm loops over the set of VMs hosted on s and calls Algorithm 3.2 indirectly to check if the server remains overloaded after deallocation. Its time complexity can be calculated by $O(N_{V^s} \cdot (4{H_s}^3 + WS^3 + n^3))$. The total time complexity of the for loop becomes $O(A \cdot N_{V^s} \cdot (4{H_s}^3 + WS^3 + n^3)^2)$.

Algorithm 3.6 Predictive VM consolidation

```
1:  Input: S_active, V
2:  foreach s ∈ S_active do
3:      if overload_detection(s) then
4:          S_over.add (s)
5:          vmsToMigrate.add(MMT_VM_selection ( s ))
6:      end
7:  end
8:  migrationMap = PABFD (S_active, vmsToMigrate)
9:  S'_active = S_active − (S_over ∪ S_destinations)
10: While (true) do
11:     U_server = underload_detection(S'_active)
12:     if U_server = NULL then
13:         Break
14:     end
15:     Exclude U_server from S'_active
16:     S_under.add(U_server)
17:     MigrationMap2 = PABFD (S_active, V^s)
18:     if migrationMap2 is complete then
19:         migrationMap.addAll(migrationMap2)
20:         U_server can be turned off after migrations
21:     else
22:         Discard migrationMap2
23:         U_server will remain active
24:     end
25: End
```

After collecting all VMs to migrate, Algorithm 3.5-PABFD is called in line 8. PABFD loops over VMs to migrate and the active servers to find their appropriate destination. It also checks if a server will become overloaded after allocating the target VM by an indirect call to

Algorithm 3.2. Its complexity is $O(N_{V^{migrate}} \cdot A \cdot (4{H_s}^3 + WS^3 + n^3))$. Consequently, the total complexity of Phase 1 (Steps 2-8) is $O\left(A \cdot N_{V^s} \cdot \left(4{H_s}^3 + WS^3 + n^3\right)^2\right) + O(N_{V^{migrate}} \cdot A \cdot (4{H_s}^3 + WS^3 + n^3))$. However, H_s, WS, and n are typically small numbers. Therefore, the complexity can be simplified to $O(A \cdot N_{V^s}) + O(N_{V^{migrate}} \cdot A)$.

3.5.4.2 Complexity –Phase 2

In the second phase (Steps 9-25), we start by calling Algorithm 3.3 for underload detection. Its complexity can be represented by $O(A \cdot (4{H_s}^3 + WS^3 + n^3))$. Then, for each underutilized host, Algorithm 3.5-PABFD is called to find new destination servers for its set of VMs V^s. In this case, $N_{V^{migrate}} = N_{V^s}$ and the complexity of PABFD is $O(N_{V^s} \cdot A \cdot (4{H_s}^3 + WS^3 + n^3))$. Thus, the total complexity of this phase is $O(A^2 \cdot \left(4{H_s}^3 + WS^3 + n^3\right)^2 \cdot N_{V^s})$. Again, this complexity can be simplified to $O(A^2 \cdot N_{V^s})$.

3.5.4.3 Overall Complexity

The overall complexity of Algorithm 3.6 will be the summation of the complexities of phases 1 and 2. Therefore it is $O(A \cdot N_{V^s}) + O\left(N_{V^{migrate}} \cdot A\right) + O(A^2 \cdot N_{V^s})$. We can approximate the number of active servers in the system by the total number of VMs divided by the number of VMs that can be allocated on a server $A \approx \frac{N}{N_{V^s}}$. In this case, the complexity becomes equal to $O(N) + O\left(N_{V^{migrate}} \cdot \frac{N}{N_{V^s}}\right) + O(\frac{N^2}{N_{V^s}})$. Hence, the worst-case complexity is $O(N^2)$.

3.5.5 Performance Metrics

As stated in the previous sections, our approach aims to find a trade-off between energy consumption and SLA violation. Hence, the following metrics are defined to evaluate the performance of the algorithms:

1. SLA violation:

Service level agreement (SLA) is a contract established between the cloud service provider and the customer about the required Quality of service (QoS). Service Level Objective (SLO) is the key element of SLA that includes one or more QoS measurements and constraints. Meeting the QoS requirements agreed upon in the SLA is extremely important for assessing the quality of cloud service and avoiding penalties. As these requirements can vary from one application to another, a workload independent metric called SLA violation ($SLAV$) is used to evaluate the SLA violation rate (Beloglazov & Buyya, 2012). It is measured by combining two SLO parameters: SLA violation due to host overloading ($SLAVO$) and SLA violation due to VM migration ($SLAVM$).

$$SLAV = SLAVO \times SLAVM \tag{3.13}$$

$SLAVO$ indicates the average ratio for the period when the host is fully utilized. If the CPU utilization of a host reaches 100%, it might not provide VMs with the required resources, which will negatively affect the performance level. This metric can be calculated as follows:

$$SLAVO = \frac{1}{M}\sum_{i=1}^{M}\frac{T_{s_i}}{T_{a_i}} \tag{3.14}$$

Where M represents the number of hosts; T_{s_i} denotes the total time during which the host i has experienced 100% CPU utilization leading to an SLA violation; T_{a_i} is the total time in which host i has been in an active state.

Whereas $SLAVM$ measures the overall performance degradation caused by VM migrations and can be calculated as shown as follows:

$$SLAVM = \frac{1}{N}\sum_{j=1}^{N}\frac{C_{d_j}}{C_{r_j}} \tag{3.15}$$

Where N is the number of VMs; C_{d_j} denotes the performance degradation of VM j due to migrations; C_{r_j} is the total CPU utilization demanded by VM j during its lifetime. C_{d_j} is set as 10% of the CPU utilization during all migrations of VM j.

2. Energy consumption:

We consider the total energy consumed by the physical machines of a data center. Most studies have determined that CPU consumes more power than memory, disk storage, and network interface (Leivadeas, Papagianni, & Papavassiliou, 2015). CPU consumption is one of the critical metrics in cloud environments that have an impact on power electricity and cooling costs as well as on heat emission. In this article, we have mainly focused on processing power, but optimizing the calculation of energy consumption while taking into account other resources is one of our future works. Here, the energy consumption measurements are based on real data provided by SPECpower benchmark results (« The SPECpower Benchmark », s.d.). Table 3.1 illustrates the power consumption of HP G4 and G5 servers at different load levels. Notably, when under-utilized servers switch to sleep mode, energy consumption decreases significantly. Therefore, reducing the number of active hosts is mandatory to minimize the energy consumption of the data center.

3. Number of migrations

Live migration negatively affects the performance of applications running on a migrating VM. It involves additional costs such as extra CPU utilization on the source host, network bandwidth between the source and destination host, downtime of the applications during VM migrations, and total migration time. Thus, reducing the number of VM migrations is essential to avoid SLA violations.

4. Execution Time

In addition to the previously mentioned metrics, we compare the algorithms in terms of execution time. Precisely, we consider the average execution time of each algorithm to perform a VM consolidation cycle including the four steps: a) overloaded hosts detection; (b) underloaded hosts detection; (c) VM selection for migration, and (d) VM placement.

Table 3.1 Power consumption of servers according to their CPU utilization (in watts)

Server	Sleep	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	10	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	10	93.7	97	101	105	110	116	121	125	129	133	135

3.6 Experiments

3.6.1 Setup

3.6.1.1 Environment

We have tested our proposed algorithms through simulations executed on CloudSim toolkit (Calheiros, Ranjan, Beloglazov, Buyya, & De Rose, 2011). Our testing environment involves 800 heterogeneous servers as follows: 400 HP ProLiant ML110 G4 machines of dual-core with 1860 MIPS each, and 400 HP ProLiant ML110 G5 of dual-core with 2660 MIPS each. Both server types have 4 GB of memory and support 1 GB/s of bandwidth. The power consumption characteristics of these servers, based on the SPECpower benchmark, are given in Table 3.1. The VM instances correspond to Amazon EC2, and their characteristics are listed in Table 3.2. Our K-SVR prediction model is implemented in Java using the LibSVM library (Chang & Lin, 2011). The parameters used to test our algorithms are summarized in Table 3.3. In our testing, we have set the upper threshold (U_TH) to 70% and the lower (O_TH) to 30%. To select these thresholds, we have referred to the state of the art and tested our algorithm under different inspired threshold values between 70% and 90% for overloading detection, and between 20% and 30% for underloading detection. Then, we have selected the values that enhance the efficiency of our algorithm, but these inputs can be easily modified. Also, we set the number of prediction steps n to 3 to predict three future CPU utilization for the host. However, our implementation is not limited to 3 and the value of n can be modified easily. The best historical data length required for training the SVR model is chosen after a careful fine-tuning of the data window size as later shown in sub-section 3.6.2. It is worth mentioning that each experiment was repeated 10 times to report the average execution time of each algorithm.

Table 3.2 VM instances characteristics

VM Instance Type	CPU (MIPS)	RAM (GB)
High-CPU medium instance	2500	0.85
Extra-large instance	2000	3.75
Small instance	1000	1.7
Micro instance	500	0.613

Table 3.3 Algorithms parameters

Kalman	A	H	Q	R
	1	1	0.01	1
SVR	ε	kernel	γ (parameter of RBF)	C
	0.1	RBF	0.0625	1
Predictive consolidation	U_TH	O_TH	WS	n
	0.3	0.7	20	3
Arima	p	d	q	
	1	0	1	

3.6.1.2 Workload

Our simulation uses real-world workloads publicly available in the form of PlanetLab data. PlanetLab data is provided as a part of the CoMon project, a monitoring infrastructure for PlanetLab (Park & Pai, 2006). It comprises CPU utilization of more than a thousand VMs hosted on servers located in more than 500 places around the World. The workload traces are collected during March and April 2011. Each VM has 288 CPU utilization records measured at 5 minutes intervals. The datasets tested in our experiments and their characteristics are shown in Table 3.4. We have selected datasets according to the number of VMs deployed and monitored in the infrastructure. In particular, we have selected the dataset with the highest number of VMs (dataset W1), the least number of VMs (dataset W4), and two datasets in-between (W2 and W3). These datasets allow us to verify if our approach performs well when increasing the number of VMs to consolidate and limiting the capacity of the physical infrastructure.

Table 3.4 Planetlab workloads characteristics (CPU utilization)

Dataset	Number of VMs	Number of servers	Date	Mean	St.dev.	Quartile1	Median	Quartile3
W1	1516	800	22/03/2011	9.26%	12.78%	2%	5%	12%
W2	1052	800	03/03/2011	12.31%	17.09%	2%	6%	15%
W3	1033	800	20/04/2011	10.43%	15.21%	2%	5%	12%
W4	898	800	06/03/2011	11.44%	16.83%	2%	5%	13%

3.6.1.3 Benchmarks comparison

To illustrate the efficiency of our approach, we compare it with the consolidation techniques proposed in (Beloglazov & Buyya, 2012). These techniques are integrated into the Cloudsim toolkit. In particular, the following four overload detection strategies are considered: Static Threshold (THR), InterQuartile Range (IQR), Median Absolute Deviation (MAD), and Local Regression (LR). Moreover, the proposed K-SVR based approach is tested using only SVR (without Kalman filtering step) and using Arima for prediction for both overload and underload detections instead of SVR.

Regarding VM selection, different methods are suggested in (Beloglazov & Buyya, 2012) to select VMs for migration from overloaded hosts: (i) Minimum migration time (MMT); (ii) Random Selection (RS); (iii) Maximum correlation (MC). Their extensive experiments to compare the different combinations between overload detection techniques and VM selection methods demonstrate that LR combined with MMT had the optimal results. To better verify the effectiveness of our algorithms in a formal comparison with theirs, we reuse the same MMT VM selection and VM placement methods explained in sub-section 3.5.3. Concerning the parameter configurations of the benchmark algorithms, we applied the same values used in their experiments. Note that by default, underloaded hosts in these consolidation policies are simply those having the least CPU utilization. We have performed different experiments to compare our consolidation approach to the default version and a modified version of the Cloudsim benchmarks.

Consequently, our K-SVR based consolidation approach is compared against the following policies: THR-MMT with an upper threshold of 80%, IQR-MMT, MAD-MMT, LR-MMT, Arima-MMT, and SVR-MMT. Experiments are explained in the next sub-section.

3.6.2 Results and discussion

Four experiments are conducted to evaluate the performance of our consolidation approach against the four benchmark algorithms introduced in the previous sub-section 3.6.1.3. Experiment 1 analyses the impact of the prediction window size on the performance of our consolidation technique. Its main target is to carefully select a WS value that can satisfy the deployment constraints (e.g., avoid crossing the tolerance threshold of SLA) and the objective metrics (save energy, reduce SLA). Experiment 2 compares the proposed consolidation approach to the existing consolidation algorithms in Cloudsim, in addition to Arima-based and SVR-based approaches. Note that the original implementation versions of the Cloudsim benchmark algorithms are tested in this experiment, where underloaded hosts are those having the least CPU utilization. Whereas experiment 3 performs this comparison but uses a modified version of the cloudsim benchmarks. Precisely, their default underload detection method is replaced by our prediction-based underload detection algorithm discussed in section 3.5.2. Adopting a similar underload detection mechanism in the benchmarks strengthens the comparison between the different overload detection techniques and allows a clearer interpretation of the obtained results. All mentioned comparisons are performed according to the performance metrics described in section 3.5.5. Finally, in experiment 4, we analyze the energy consumption of executing the proposed predictive consolidation approach against the benchmarks discussed in experiment 2.

3.6.2.1 Experiment 1- WS selection

In this simulation, our consolidation technique is tested under variable prediction window sizes of lengths 4, 8, 12, 16, 20, 24, and 28. The purpose is to study the influence of WS on the performance of our approach because it is based on the utilization prediction of the hosts to

determine overloading and underloading states and trigger VM migrations. It should be noted that the same experiment is repeated for all datasets presented in Table 3.4. However, no particular difference was observed. To this end, testing results for dataset W1 are given in Table 3.5 as an example. Based on the results, we notice the contradiction between the two resource optimization objectives (SLA and energy) and the importance of finding a solution that provides a trade-off between them. Decreasing the energy consumption and reducing the SLA violation at the same time is difficult and probably not possible. The SLA violation rate of our approach increases progressively from 0.00004% to 0.00015% with the increase of the WS value. Conversely, the total energy consumption in the datacenter is the highest (200.86 Kwh) when the WS is equal to 4. Then, it decreases progressively to 158.81 Kwh with the increase of WS to 24. At the end, the energy re-increases slightly for WS 28 showing a convergence of the energy consumption with respect to the WS. The observations demonstrate that a trade-off between the SLA violations and energy consumption is occurred. Accordingly, if the main target of the cloud service provider is to strictly achieve the least SLA violation rate and avoid its resulting penalties, it is better to go with a small WS value. On the contrary, if the priority is to reduce energy expenses in cloud data centers, higher WS can better serve this objective. In this article, we do not prioritize an objective metric over another. However, we intend to find the best trade-off between SLA violation and energy consumption. Through our experiments, the consolidation technique achieves this trade-off and performs optimally at WS 20 under different workloads. Thus, WS=20 is used for testing in the following experiments.

Table 3.5 Testing results of different windows size values

WS	Energy (Kwh)	SLAV (%)	SLAVO (%)	SLAVM (%)	Number of migrations
4	200.86	0.00004	0.69	0.01	2796
8	191.71	0.00004	0.78	0.01	2949
12	179.73	0.00005	0.87	0.01	3042
16	167.60	0.00007	1.12	0.01	3708
20	159.29	0.00012	1.5	0.01	4786
24	158.81	0.00013	1.55	0.01	4986
28	160.60	0.00015	1.59	0.01	5287

3.6.2.2 Experiment 2 – proposed approach vs. original benchmarks

Figs. 3-8 present the comparison results between our K-SVR based consolidation approach and the four benchmarks discussed in subsection 3.6.1.3: THR-MMT, IQR-MMT, MAD-MMT, and LR-MMT, in addition to our approach based on Arima for prediction, and our approach based on SVR only without Kalman filtering.

Based on the results obtained in Figure 3.3, our K-SVR based consolidation approach outperforms its energy results using SVR only, in all workloads. It also outperforms slightly Arima based approach in most cases. Arima-based approach occupies the second place in minimizing energy and outperforms SVR-based approach in most cases except for workload W4. Moreover, we notice that our K-SVR based approach reduces the total energy consumed in the datacenter by an average of 7.54 %, 18.86 %, 20.65 %, and 20.56 % compared to LR, MAD, IQR, and THR respectively. LR outperforms all the threshold methods in terms of energy. By employing our proposed algorithms to
select underloaded and overloaded hosts, such hosts are identified more precisely. Once underloaded hosts are selected, all their hosted VMs can be migrated to other machines, and then energy can be saved by switching idle hosts to sleep mode. Moreover, estimating the trend of future CPU consumption of the hosts helps to anticipate the overloading situations and to react proactively before a violation occurs. For this reason, our approach significantly decreases the SLA violation due to host overloading (SLAVO) compared with Cloudsim benchmarks, as shown in Figure 3.4. K-SVR has the best results in SLAVO reduction, then, SVR-based approach, while Arima approach comes third. After identifying overloaded hosts, some VM migration plans can be applied to re-adjust resources and overcome the issue. Regarding the SLA violation due to migrations, Figure 3.5 reveals that K-SVR, SVR and Arima based approaches achieve the lowest SLAVM versus the others for all tested datasets. The reason refers to the substantial reduction in the number of VM migrations when using predictive approaches (this metric is discussed later). Since SLAV represents the multiplication of SLAVO and SLAVM, our proposed K-SVR approach considerably decreases SLAV as well. Figure 3.6 demonstrates that K-SVR approach minimizes the SLA violation rate by an

average of 97.36%, 96.17%, 96.12%, and 96.21% compared with LR, MAD, IQR, and THR respectively. It also reduces SLAV by an average of 24.5728% compared to SVR, and by 33.5233% compared to Arima.

Figure 3.7 illustrates the comparison results in terms of the number of migrations. Our K-SVR approach dramatically reduces the number of migrations in all workloads compared to Cloudsim benchmarks. For instance, K-SVR approach initiates 3632 migrations for dataset W2, while the other techniques perform 27632 (LR), 26292 (MAD), 26476 (IQR), and 26634 (THR) migrations. It also has a lower number of migrations than SVR and Arima which perform 4546 and 4896 migrations respectively for the same dataset (W2). Live VM migration may cause overhead on the system, extra expenses, and more violations. Thus, a consolidation mechanism that requires fewer migrations is much preferable. In addition, avoiding unnecessary VM migrations minimizes the runtime of our re-allocation process (e.g., selection of VMs to migrate and destination hosts). Figure 3.8 also indicates a lower execution time for K-SVR and SVR based techniques compared to the others. K-SVR has a slightly lower runtime compared to SVR in all workloads except W1. Using Arima for prediction to detect overloaded and underloaded hosts longer time than SVR-based approaches. Overall, K-SVR approach provides a successful trade-off between energy cost and SLA violation. It also outperforms benchmark algorithms in almost all targeted cost metrics: energy, SLA, number of migrations, and execution time.

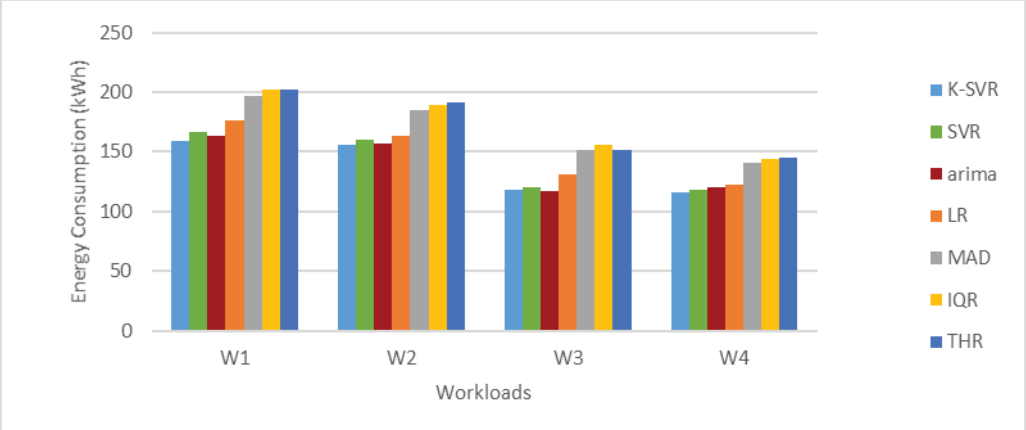

Figure 3.3 Comparison of energy consumption for 4 workloads - experiment 2

Figure 3.4 Comparison of the SLAVO metric for 4 workloads - experiment 2

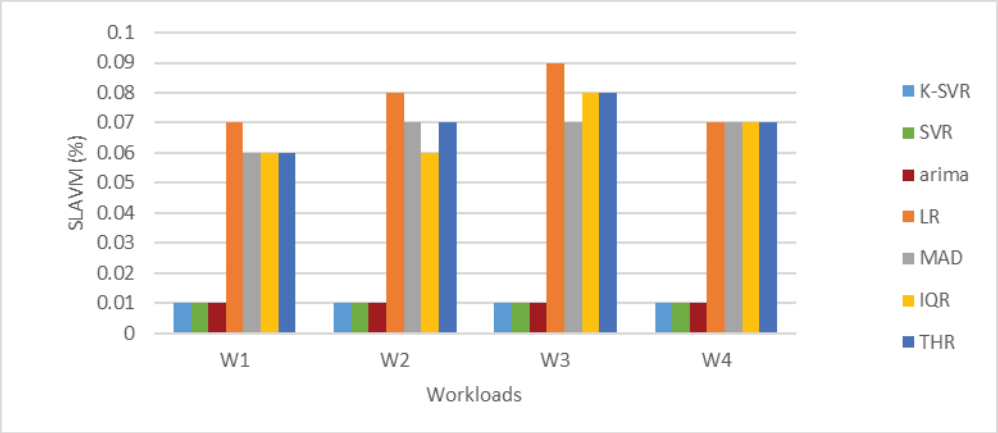

Figure 3.5 Comparison of the SLAVM metric for 4 workloads- experiment 2

Figure 3.6 Comparison of the SLAV metric for 4 workloads - experiment 2

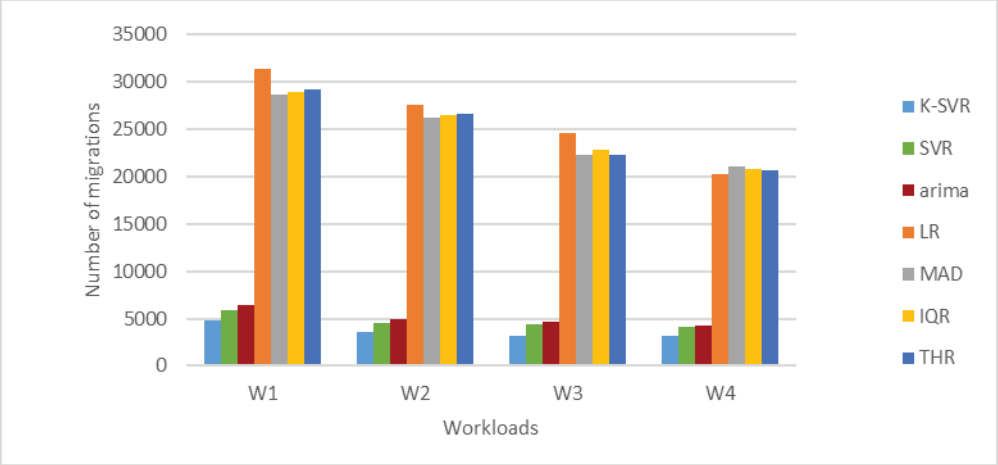

Figure 3.7 Comparison of number of VM migrations for 4 workloads - experiment 2

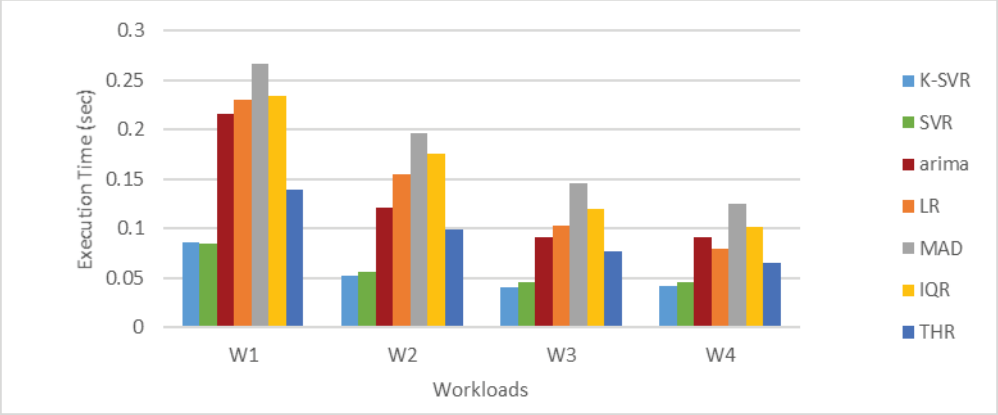

Figure 3.8 Comparison of runtime for 4 workloads - experiment 2

3.6.2.3 Experiment 3 – proposed approach vs. modified benchmarks

Figs. 9-14 illustrate the comparison results of our consolidation approach against the modified version of Cloudsim benchmark algorithms. Note that testing results of K-SVR, SVR, and Arima-based approaches are the same as experiment2. However, Cloudsim benchmarks (LR, MAD, IQR, and THR) are modified to utilize our proposed underload detection algorithm explained in sub-section 3.5.2. Therefore, the overload detection part only differentiates them. That way we can focus our comparison on the overloading part and verify if the benchmark algorithms combined with prediction-based underloading detection can outperform our

approach. Figure 3.9 exhibits competitive performance in terms of energy consumption, specifically between LR, K-SVR, and Arima-based approaches. While K-SVR approach consumes less power in W1 and W4, LR has lower consumption in W2 and Arima has lower power in W3 with a very slight difference for both. Threshold techniques consume the highest energy (THR, IQR, and MAD). Figure 3.10 demonstrates that K-SVR technique outperforms others in terms of SLAVO in almost all datasets (except in W3). In W3, threshold techniques (THR and MAD) provide a better SLAVO rate where they slightly outperform K-SVR approach. Thus, our overload detection algorithm selects over-utilized machines more efficiently than other algorithms and causes less SLAVO rate in most cases. Concerning SLAVM, Figure 3.11 indicates that K-SVR, SVR, and Arima approaches provide the least SLAVM in all datasets. Combining the two metrics SLAVM and SLAVO, the greater performance of K-SVR approach in terms of SLAV compared to the benchmarks, is evidently observed in Figure 3.12. In addition, it is important to realize the significant decrease in SLAVO, SLAVM, and SLAV for all benchmark techniques when combined with our underload detection algorithm, compared to the results in experiment 2. Figure 3.13 compares the number of VM migrations. Explicitly, K-SVR based approach applies the lowest number of VM migrations to re-allocate resources. The average difference in the number of migrations is quite significant between our approach and the other techniques: 49.31% compared to LR, 48.78% compared to MAD, 49.38% compared to IQR, and 46.37% compared to THR. However, the number of migrations for benchmark algorithms is considerably reduced compared to experiment 2. In terms of execution time, Figure 3.14 shows competitive results, especially between K-SVR approach, LR, and THR. Sometimes, K-SVR approach outperforms THR in terms of runtime with a very slight difference (e.g., in W3 and W4), and sometimes the inverse. LR has achieved a lower execution time than K-SVR approach in W1 only. We can also notice the reduction in runtime of the benchmark algorithms when combined with our K-SVR based underload detection algorithm, compared to experiment 2. This combination allows these benchmarks to outperform Arima-based approach in terms of execution time. To summarize, adopting our underload detection algorithm in benchmark approaches, improves their performance results, and considerably minimizes their SLA violation rates, the number of VM migrations, and their runtime. Despite this, our approach

achieves the best results and outperforms these approaches specifically in terms of SLA violation (SLAV) and the number of migrations, while maintaining an appropriate balance with the power consumption.

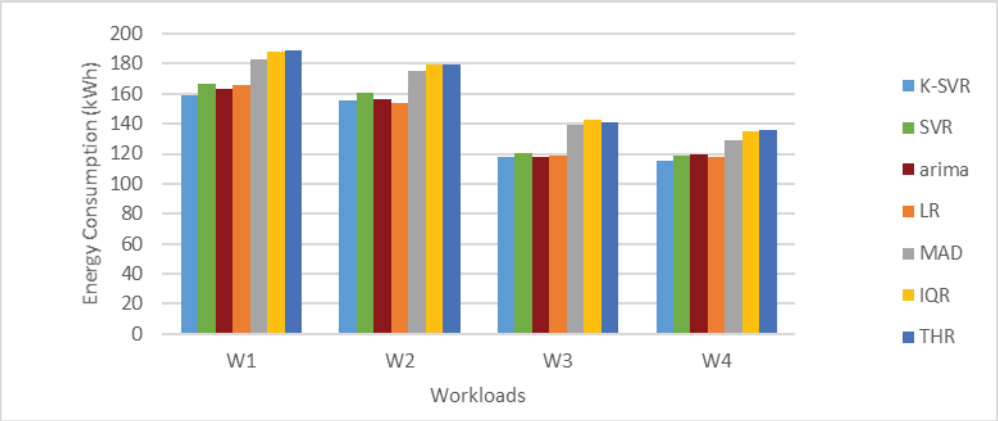

Figure 3.9 Comparison of energy consumption for 4 workloads-experiment 3

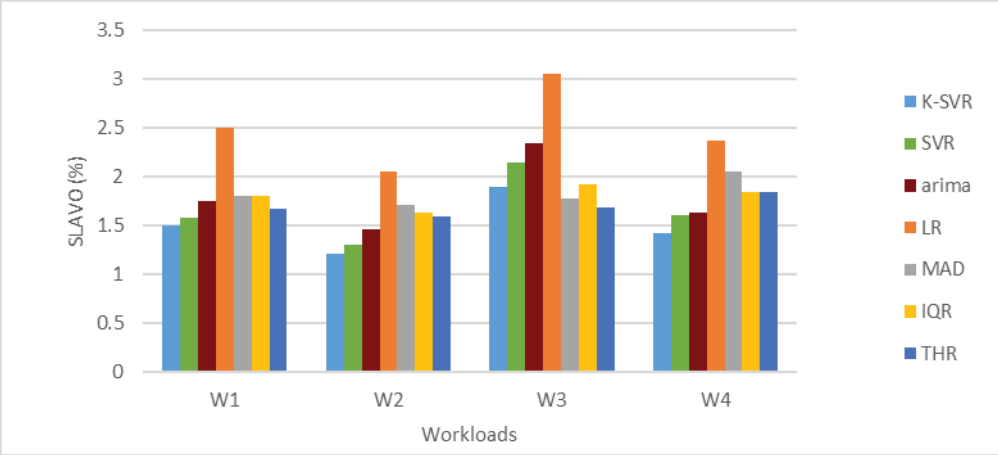

Figure 3.10 Comparison of the SLAVO metric for 4 workloads-experiment 3

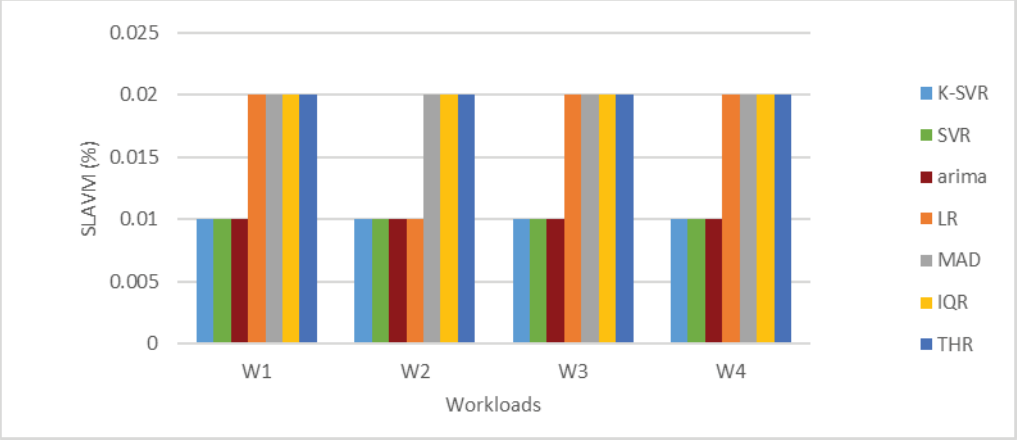

Figure 3.11 Comparison of the SLAVM metric for 4 workloads-experiment 3

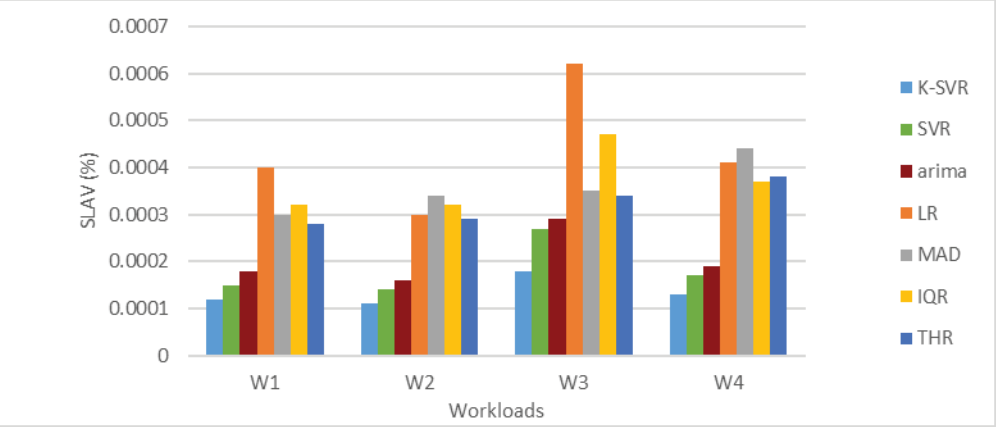

Figure 3.12 Comparison of the SLAV metric for 4 workloads - experiment 3

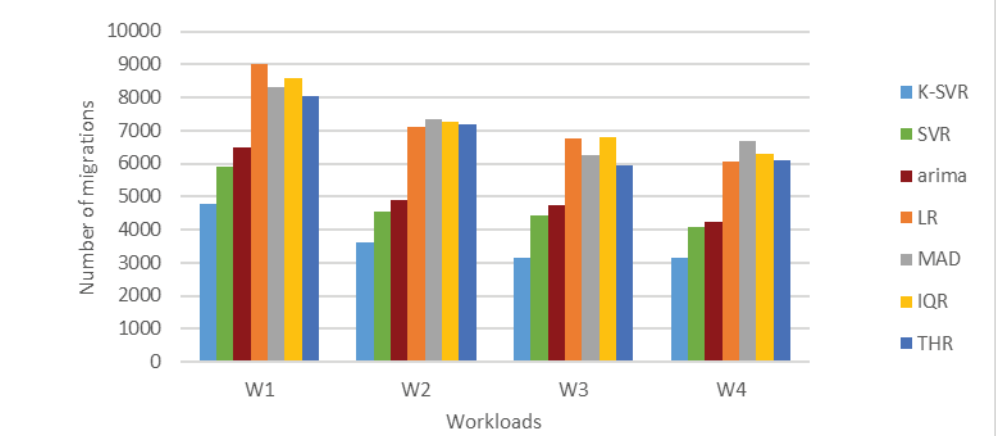

Figure 3.13 Comparison of number of VM migrations for 4 workloads – experiment 3

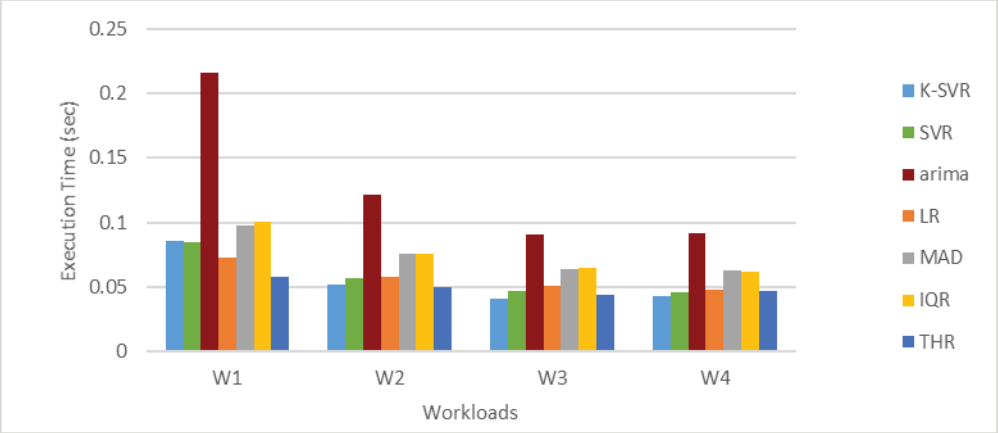

Figure 3.14 Comparison of runtime for 4 workloads - experiment 3

3.6.2.4 Experiment 4 – Energy consumption of our proposed performance profiling and prediction

To measure the energy consumed by executing our proposed approach, JoularJX tool (Noureddine, 2022) is used, a power monitoring tool at the source code level. It uses Intel RAPL (powercap interface) to get the power consumed by a running java program using its PID. On program exit, it outputs its total energy consumed in joules, in addition to the overall energy consumed by every java method executed inside it.

We use this tool to obtain the energy consumed by the entire simulation on Cloudsim from the instantiation phase until the termination time. Figure 3.15 illustrates the comparison of the total energy consumed by the simulation using our K-SVR based consolidation methodology and the benchmarks discussed in experiment 2. On the other hand, Figure 3.16 compares the overall energy consumed by the VM consolidation algorithm (Algorithm 3.6) during the simulation. As discussed previously, Algorithm 3.6 includes all decisions related to the consolidation process (Overloaded Hosts detection, Underload detection, Selection of VMs to migrate and their destination server). The complete simulation includes the instantiation phase, the execution of Algorithm 3.6 and all its sub-algorithms, and the application of the decisions made by this algorithm by updating the infrastructure and its resources. It is worth mentioning that the simulation performs many consolidation cycles and so Algorithm 3.6 is executed many

times to take the relevant decisions. Precisely, in our testing results, the simulation performs 288 consolidation cycles which is equal to the PlanetLab dataset size taken as input. Based on the results obtained, our approach reveals less energy consumption of running the complete simulation or Algorithm 3.6 specifically compared to other benchmarks for all workloads. As shown in Figure 3.7 of experiment 2, the other benchmarks may generate false overload and underload detection decisions than our approach which involves a much higher number of VM migrations. A higher number of VM migrations means more modifications in the infrastructure, resource allocations and deallocations processes, more calls for VM selection and VM placement algorithms, etc. and consequently a higher energy for executing these tasks.

In Table 3.6, we represent more details about the overall energy consumed by our Kalman-SVR prediction part during the simulation, specifically Algorithm 3.1. Note that Algorithm 3.6 calls algorithm 3.1 each time it needs to predict host resource utilization for overloading and underloading checking purposes. In the table, we also specify the number of times the prediction algorithm has been called during the simulation. As mentioned previously, in our simulation we execute Algorithm 3.6 for 288 times but in the first 20 times, there are no prediction calls due to insufficient historical data for the hosts (Window Size for prediction=20). For instance, Table 3.6 shows that for workload w1, the prediction algorithm is called 101992 times during the simulation (for 288-20=268 cycles), and its overall energy consumed is 52.825 joules which represent 36.72% of the energy consumed by Algorithm 3.6. If we convert the energy unit to KWh, the results are too small, demonstrating that running our proposed consolidation approach does not result in high energy consumption or overhead despite its data filtering and prediction tasks.

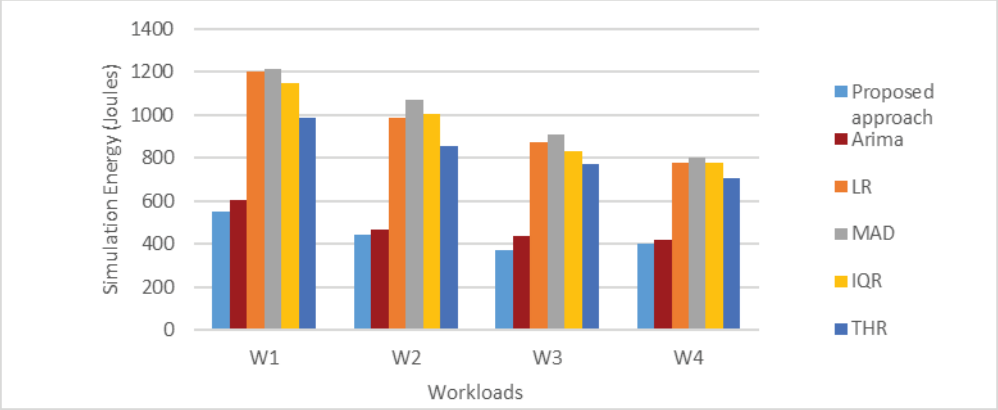

Figure 3.15 Comparison of energy consumed by the simulation execution for 4 workloads - experiment 4

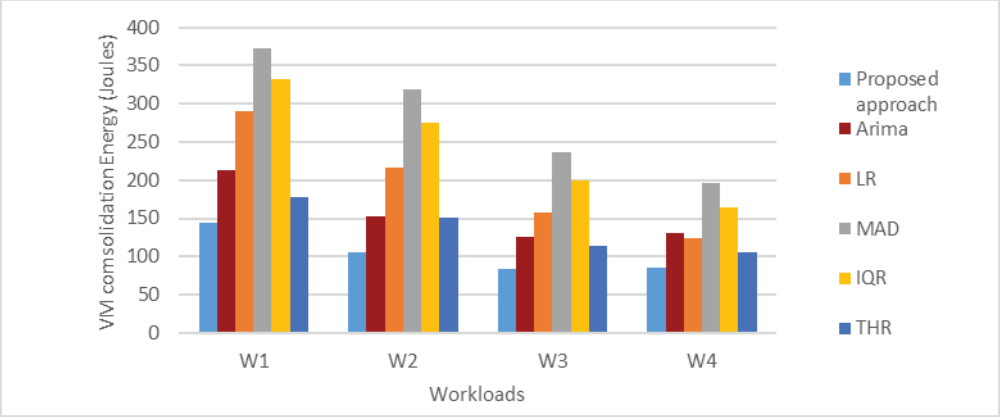

Figure 3.16 Comparison of energy consumed by the VM consolidation algorithm (Algorithm 3.6) for 4 workload-experiment 4

Table 3.6 Detailed energy measurements of executing our technique

Workloads	Overall Energy measurement (in Joules)			Number of K-SVR prediction (Algo. 3.1) runtimes		
	Whole simulation	K-SVR Prediction Algorithm 3.1	VM consolidation Algorithm 3.6	For overload checking	For underload checking	Total
W1	550.72	52.82	143.86	84613	17379	101992
W2	441.85	46.89	104.95	58276	18452	76728
W3	370.24	23.92	84.32	31640	10635	42275
W4	401.73	35.29	85.50	43859	12641	56500

3.7 Conclusion

In this paper, a predictive dynamic VM consolidation algorithm is proposed. Our multi-step prediction model combines Kalman Filter with Support Vector Regression (SVR) to forecast the future CPU utilization of the hosts. Considering current and predicted utilization, our underload detection, and overload detection techniques make decisions about the host state (overloaded or underloaded). Based on these decisions, VM re-allocations are planned, and migrations are triggered. The main target is to find a trade-off between energy consumption and SLA violation while minimizing the number of VM migrations and the runtime needed for resource re-allocations.

Various Simulations are conducted using PlanetLab's real-world workload traces. The experimental results demonstrate the ability of the proposed approach to significantly reduce the SLA violation rate and the number of migrations, with an appropriate balance with the energy consumption and very good execution time, compared to the existing benchmark algorithms. For future work, we intend to optimize the costs computation formulas taking into account additional factors and resources (memory, I/O, etc.) that may increase energy consumption and SLA violation during migrations. We also plan to extend our prediction to support multiple resources (e.g., memory, storage, and bandwidth) instead of a single resource (CPU). Considering multiple resource types to determine accurately the host state may limit the frequency of VM migrations and avoid hasty decisions.

Acknowledgment

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

CHAPTER 4

MULTI-RESOURCE PREDICTIVE WORKLOAD CONSOLIDATION APPROACH IN VIRTUALIZED ENVIRONMENTS

Mirna Awad [a], and Aris Leivadeas [a]

[a] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Submitted to *Journal of Computer Networks*, June 2023.

4.1 Abstract

The revolution of virtualization technologies and Cloud computing solutions has emphasized the need for energy-efficient and Service level agreement (SLA)-aware resource management techniques in cloud data centers. Workload consolidation in Infrastructure-as-a-Service (IaaS) providers allows for efficient utilization of hardware resources and reduced energy consumption by consolidating workloads onto fewer physical servers. To ensure successful workload consolidation, it is crucial for IaaS providers to carefully estimate the host state and identify overloaded and underloaded hosts, thereby avoiding overly aggressive consolidation. Existing proposals determine the host state depending on its current resource utilization or a single anticipated resource utilization value, and often consider only a single resource type of the host, such as CPU. These limitations may lead to unreliable host state estimations, resulting in excessive and needless service migrations between physical machines (PMs). This, in turn, can lead to extra delays in service execution, degraded performance, increased power consumption, and SLA violations. To address these challenges, we propose a workload consolidation approach that leverages a multi-resource and multi-step resource utilization prediction model. Based on this model, our overload and underload decision-making algorithms consider the forecasted future trend (sequence of future value) of each host resource's utilization, including CPU, memory, and bandwidth. Through extensive experimentations conducted with two real-world datasets, we demonstrate that our approach

can significantly reduce power consumption, SLA violation rate, and the number of migrations compared to existing benchmarks.

Keywords: Multi-resource, workload prediction, Kalman filter, Support vector regression, consolidation approach, Cloud computing.

4.2 Introduction

With the remarkable evolution of cloud computing solutions for hosting services, from the usage of Virtual Machines (VMs) to containers, and recently to serverless platforms, an effective resource management remains a significant challenge (Khan, Tian, Zhou, et al., 2022)(Awad, Kara, & Edstrom, 2022)(McGrath & Brenner, 2017). The dynamic workload fluctuations of running services leads to high variations in virtual and physical resource consumption in the cloud. By consolidating workloads onto a minimal number of physical servers, Infrastructure-as-a-Service (IaaS) providers can optimize the utilization of their hardware resources while reducing energy consumption through the powering off of under-utilized servers(Panwar et al., 2022). In virtualized data centers, this consolidation is possible through the live migration of VMs or containers hosting the running services, between physical machines (PMs) (F. Zhang et al., 2018). While workload consolidation optimizes resource utilization and reduces the energy consumed, it can negatively affect the performance requirements of applications defined in the Service level agreement (SLA). Thus, striking the right balance is essential.

For instance, implementing an excessively aggressive consolidation approach can lead to violations and performance degradation for the applications running on the servers. Accordingly, many concerns should be addressed to tackle such a research problem. First, to achieve an effective workload consolidation, it is crucial to carefully estimate the host state and identify overloaded and underloaded hosts in order to make informed decisions about workload redistribution. Underloaded hosts that have excess available resources can be utilized for consolidating additional workloads from overloaded hosts to improve overall resource

utilization and mitigate the risk of SLA violations. Alternatively, they can be turned off for minimizing energy consumption. The second concern involves selecting the right virtual resources (VMs or containers) to migrate from a server (Melhem et al., 2017)(Moghaddam et al., 2018). Lastly, a placement strategy is needed to select the best destination servers that can handle these migrated workloads (Nath, Addya, Chakraborty, & Ghosh, 2020). Each of these concerns poses its own challenges.

Regarding overload and underload detection concern, existing workload consolidation approaches often rely on current resource utilization of servers to determine their state and trigger the required migrations (Xiao et al., 2019)(Beloglazov & Buyya, 2012). However, such proposals may result in unreliable host state estimations and excessive needless migrations. An increase in the current server utilization does not necessarily reveal an overloading state, as the load may rapidly decrease in the next time slot. To make accurate estimations and limit the frequency of migrations, it is crucial to consider not only the current host workload but also its future resource utilization. Moreover, relying on a single future resource utilization value to judge the host state, may be also insufficient to perform reliable estimations. Therefore, it is important to anticipate and consider the future trend (as a sequence of multiple future values) of the host's resource usage.

Furthermore, some existing consolidation schemes focus on the utilization of a single resource type (i.e., CPU) while deciding whether a server is overloaded or underloaded (L. Li et al., 2019)(Hsieh et al., 2020). Due to the sheer multiplicity and heterogeneity of running applications (e.g., IoT-based applications, 5G applications, web etc.) that may be hosted in the cloud, and the variability of their workloads, considering only one resource type can lead to a non-efficient decision-making strategy. Different applications may have different resource requirements, such as CPU-intensive, memory-intensive, or bandwidth-sensitive workloads, and so on. Therefore, it is necessary to consider all these resource types in order to build a technique able to correctly detect overloaded and underloaded servers across different application types and workloads.

In this article, we present a predictive workload consolidation mechanism based on prediction model called MSPR. This mechanism incorporates overload detection and underload detection algorithms, which take into account the current and predicted trend (instead of a single future value) of resource utilization to determine if a host is experiencing overloading or underloading issue. Unlike some existing techniques, our approach considers many resource types, including CPU, memory, and bandwidth, when making host state estimation. For overload detection, we calculate adaptive Mean Absolute Deviation (MAD) thresholds tailored to each resource type. Moreover, our approach offers flexibility by allowing the specification of distinct underload thresholds and prediction window sizes for each resource type. Our evaluation involves comparing our approach with an optimized multi-resource versions of benchmark consolidation algorithms integrated into Cloudsim, including Mean Absolute Deviation MAD-based, Interquartile Range IQR-based, Static threshold THR-based, and Local Regression LR-based approaches, in addition to our alternative consolidation approach that uses Autoregressive Integrated Moving Average (Arima) multi-resource prediction model, replacing MSPR model. In summary, our contributions include:

1. A multi-resource and multi-step workload prediction model called MSPR is proposed, to anticipate the future resource utilization trends of servers in terms of CPU, memory, bandwidth received, and bandwidth transmitted. This model combines the well-known algorithms Support Vector Regression (SVR) and Kalman Filter. Kalman Filter is used as a data filtering pre-processing step to enhance the accuracy of SVR prediction process.
2. A workload consolidation approach is combined with the MSPR predictive model to reduce the total energy consumption in data center, limiting the frequency of virtual resource migrations, and decreasing SLA violations. This approach includes OD-MSPR and UD-MSPR algorithms for overload and underload detection, considering both current and predicted multi-resource utilization trends of servers. Our approach also offers the flexibility to specify different overload and underload thresholds for each resource type. Adaptive thresholds for overload detection, based on MAD, are calculated for each resource type based on historical utilization data.

3. Extensive experiments are conducted on Cloudsim using Bitbrains (Shen, Van Beek, & Iosup, 2015) and Materna (Kohne, Spohr, Nagel, & Spinczyk, 2014)(Kohne, Pasternak, Nagel, & Spinczyk, 2016) datasets to validate the effectiveness of our proposed approach compared to optimized multi-resource versions of state-of-the-art consolidation techniques, in addition to an alternative consolidation approach that replaces the MSPR model with an ARIMA multi-resource prediction model. Moreover, a detailed time complexity analysis of our predictive workload consolidation approach is provided

The rest of the article is organized as follows: Section 4.3 discusses the existing workload consolidation approaches in the state-of-the-art and highlights their limitations. Section 4.4 describes the proposed resource utilization prediction technique. Section 4.5 explains in detail the predictive workload consolidation approach proposed and analyzes its time complexity. Section 4.6 presents the experimental setup and compares the results obtained by our proposal with existing benchmarks. Section 4.7 summarizes the main insights and the future directions for this work.

4.3 Related work

Workload consolidation and resource utilization prediction constitute two major research problems that can be tackled separately. In this section, we review some related proposals that address workload consolidation problem alone, workload prediction problem alone, or both combined.

Numerous studies about workload prediction has been provided in the pertinent literature (Masdari & Khoshnevis, 2020). Qui et al. (Qiu et al., 2016) present a novel method for predicting the CPU utilization of VMs in a cloud computing environment. The proposed approach utilizes a deep learning model consisting of a Deep Belief Network (DBN) and a logistic regression layer. The parameters of the entire model are fine-tuned using the Backpropagation (BP) algorithm. The proposed approach is compared to other prediction

methods, using PlanetLab dataset in Cloudsim. Experimental results demonstrate that the proposed method outperforms existing prediction approaches in terms of prediction accuracy. Malik et al. (Malik et al., 2022) present a multi-resource utilization prediction technique based on Functional Link Neural Network (FLNN). A hybrid model, that combines genetic algorithm (GA) with particle swarm optimization (PSO) algorithm, is used to train the neural network and thus, improve its prediction accuracy. Mean Absolute Error (MAE) is calculated as fitness function for GA. Their experiments are carried out using a Google cluster workload and are focused mainly on CPU and memory utilization of VMs. Xie et al.(Xie et al., 2022) propose a hybrid model of ARIMA and triple exponential smoothing to accurately predict both linear and nonlinear relationships in container resource load sequence. The weighting values of the two single models in the hybrid model are chosen according to the sum of squares of their predicted errors for a period of time. They also introduce a real-time resource prediction system for Docker container that optimizes CPU and memory resource usage based on predicted values. Khan et al. (Khan, Tian, Ilager, & Buyya, 2022) propose an intelligent prediction model based on machine learning for workload prediction and energy state estimation for VMs in cloud data centers. The model explores different Machine Learning (ML) algorithms for workload prediction including Linear Regression (LR), Ridge Regression (RR), ARD Regression (ARDR), ElasticNet (EN) and deep learning algorithm called Gated Recurrent Unit (GRU). The obtained results show that the GRU achieved the most negligible root mean square error (RMSE) value compared to other ML algorithms. In addition to workload prediction, the authors propose four different clustering algorithms including semi-supervised affinity propagation based on transfer learning (TSSAP), CLA based on transfer learning (TCLA), k-means based on transfer learning (TKmeans), and P-teda based on transfer learning (TP-teda), for identifying similar groups of VMs with different energy-consuming states. Based on their experiments, the TSSAP outperformed other methods by achieving the highest accuracy in clustering.

In this article, we present a prediction approach called MSPR, for forecasting the utilization of server resources, encompassing CPU, memory, received bandwidth, and transmitted bandwidth. The proposed model leverages a combination of Support Vector Regression (SVR)

and Kalman Filter algorithms to accurately predict future resource utilization. Kalman Filter acts as a pre-processing step to improve the accuracy of SVR predictions. This predictive model can be employed to anticipate workload of diverse systems, including servers, virtual machines (VMs), and containers etc. In our work, we concentrate on utilizing this model to forecast incoming workload for servers, thereby enabling reliable estimation of their future states.

Workload consolidation is a crucial strategy for optimizing resource re-allocation and achieving energy efficiency in cloud environments (Chaurasia et al., 2021)(Helali & Omri, 2021b)(Zolfaghari & Rahmani, 2020). It involves the consolidation of workloads onto a reduced number of physical machines, thereby maximizing the utilization of available resources, and minimizing the energy consumed. Researchers have explored different approaches and perspectives to address this problem effectively. Some of these include VM or container placement algorithms, techniques for detecting overloading and underloading states, and strategies for selecting VMs or containers to migrate. For instance, Nath et al. (Nath et al., 2020) propose EASY, an energy-efficient approach for container consolidation in cloud data centers. EASY utilizes a Bayesian optimization-based algorithm for container placement, to minimize energy consumption while considering trade-offs with service response time. The authors compare the performance of the EASY algorithm with baseline methods, including Consecutive Allocation, Best Fit, and First Fit Decreasing mechanisms. Simulation results demonstrate the effectiveness of EASY approach in reducing energy consumption, although slightly increasing average response time.

In this paper, our primary focus is on the detection of host overloading and underloading conditions. We recognize the significance of accurately identifying these states as they have a direct impact on the performance and efficiency of the consolidation system. Overloaded hosts may experience resource scarcity and can lead to degraded performance and potential SLA violations. On the other hand, underloaded hosts indicate underutilization of resources, which results in resource wastage and unnecessary costs. State-of-the-art employs three primary approaches: Static Threshold, Dynamic Threshold, and Predictive Models, to identify

overloaded and underloaded hosts. The Static Threshold approaches set fixed upper and lower thresholds on resource utilization to determine whether a server is overloaded or underloaded. Conversely, Dynamic Threshold approaches dynamically adjust the thresholds based on the observed workload historical data. For example, the authors in (Beloglazov & Buyya, 2012) propose the utilization of two statistical estimators, namely Median Absolute Deviation (MAD) and Interquartile Range (IQR). By comparing the current utilization of the host to these dynamically computed thresholds, the methods provide an evaluation of the host's state over time. Predictive models play a crucial role in resource consolidation by predicting whether a server will experience overloading or underloading conditions in the near-future. This prediction aids in avoiding unnecessary migrations that can lead to significant overhead. In (Beloglazov & Buyya, 2012) two predictive models are proposed, the Local Regression (LR) technique which estimates future server resource utilization, and its enhanced version Robust Local Regression (LRR) which is designed to be more robust against outliers.

The workload trace of a server may falsely indicate an increase in resource usage, but the load rapidly decreases in the near future. To address this issue, the adoption of a multi-step prediction model is more accurate, as it can anticipate a sequence of future host utilization values. This method aims to avoid unreliable decisions based on temporary fluctuations in resource utilization. Hieu et al. (Hieu et al., 2020) propose a VM consolidation technique based on a multiple resource usage prediction model. The prediction technique employs multiple linear regression to forecast k future resource utilization of servers. A server is detected as overloaded in a resource d if its multiple predicted resource utilization U_{t+k}^{d} exceeds a hot threshold h. Whereas, it is defined as under-utilized in a resource d when its multiple predicted resource utilization U_{t+k}^{d} is less or equal to its current utilization U_t. Minarolli et al. (Minarolli, Mazrekaj, & Freisleben, 2017) address the challenge of detecting overloaded hosts in cloud computing by making long-term predictions of resource demands for VMs. The authors employ Gaussian processes as a machine learning approach for time series forecasting. The approach constructs a probability distribution model of the prediction error, to quantify the uncertainty associated with the long-term predictions. Based on this model, a decision-theoretic approach utilizing a utility function is introduced to address the impact of live

migration overheads in VMs. This approach selectively initiates live migration actions only when the anticipated penalty associated with SLA violations outweighs the utility value attributed to live migration overhead. Arshad et al. (Arshad, Aleem, Srivastava, & Lin, 2022) propose a scheduling mechanism called Energy Efficiency Heuristic using VM Consolidation (EEHVMC), which consolidates VMs on host machines. By setting two thresholds T_{high} and T_{low}, EEHVMC mechanism classifies hosts in a cloud data center into three main categories: Host Over-Loaded, Host Medium-Loaded, and Host Under-Loaded. The mechanism identifies host state by comparing its CPU and memory utilizations to defined thresholds. Once overloaded and underloaded hosts are identified, specific VMs are selected for migration to medium-loaded hosts using a method called Maximum ratio of CPU utilization to memory utilization (MRCU). If a VM is identified as being CPU-bound or memory-intensive, it is considered as a candidate for migration. Sayadnavard et al. (H. Sayadnavard et al., 2022) present a multi-objective approach for dynamic VM consolidation in cloud data centers. The main objectives are to reduce energy consumption, improve system reliability, and minimize resource wastage. The proposed consolidation approach includes: a model that combines Discrete Time Markov Chain (DTMC) and Continuous Time Markov Chain (CTMC) for physical machines (PMs) categorization, a heuristic VM selection algorithm considering the task completion time, and a VM placement strategy using ε-dominance-based multi-objective artificial bee colony (ε-MOABC) algorithm. The approach is compared to traditional consolidation approaches integrated into CloudSim simulation platform. The comparison demonstrates the superiority of the proposed approach in achieving the defined objectives.

In our previous work (Awad, Kara, & Leivadeas, 2022), we proposed a consolidation approach based on a multi-step-ahead workload prediction model that combines Support Vector Regression with Kalman filter. However, the limitations of our proposed algorithms were evident as they focused solely on CPU utilization forecast and on static thresholds to make underload and overload decisions. In our current work, we have made significant enhancements to address these limitations. Firstly, we have optimized the prediction technique to forecast future utilization trends of multiple resources, including CPU, memory, and bandwidth. Additionally, we have revamped the underload and overload techniques to consider

all these resource types when estimating the host state. Moreover, instead of using static thresholds, we now calculate adaptive Median Absolute Deviation (MAD) thresholds for overload detection, which vary for each resource type. These thresholds are determined based on historical utilization data for each specific resource. Another different aspect of our work is the redefinition of underloaded hosts. In our previous approach, hosts with the minimum CPU utilization were considered underloaded when there was insufficient data for prediction. With the integration of multi-resource considerations, we have redefined underloaded hosts to include those with actual resource utilizations falling below the underloaded thresholds for all resources. Furthermore, we have refined our objective metrics to include all considered resource types in the calculation of energy consumption and SLA violation. To ensure a formal comparative study, we have also improved the implementation of other approaches such as LR, THR, MAD, IQR, and Arima-based methods. Similar to our approach, these approaches now consider multi-resources in their overload and underload decisions. Lastly, for testing purposes, we have utilized two datasets, namely Materna and bitbrains, instead of the previously used planetlab dataset (Park & Pai, 2006).

4.4 Workload Prediction Model

This section explains our suggested **M**ulti-**R**esource and multi-**S**tep-ahead **P**rediction model, called **MSPR,** for forecasting host resource utilizations based on Support vector regression (SVR) and Kalman. SVR (Abdullah et al., 2020) is a well-known machine learning technique derived from Support Vector Machine (SVM) specifically to solve regression problems. It is suitable for the complex and dynamic cloud environment and is mainly used in our work to proactively predict future host resource utilization. Kalman Filter (Kalyvianaki et al., 2014) is also a famous algorithm originally built to estimate the time-varying states in dynamic systems, which makes it suitable for the dynamic load estimation of cloud applications. Our prediction model integrates Kalman Filter as a data pre-processing step which aims to filter data, eliminate noises, and enhance the SVR prediction accuracy. In the following, the working principles of the aforementioned techniques are explained.

4.4.1 Kalman Filter

Kalman filter aims to estimate the state x of a discrete-time controlled process using a set of measurements observed over time. The following linear stochastic difference equation shows the evolution of the state x from time k-1 to time k:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.1}$$

The above equation can be combined with a measurement z, as follows:

$$z_k = Hx_k + v_k \tag{4.2}$$

Where A is the state transition matrix from time k-1 to time k. B is a control matrix that relates the control vector u to x. H is a matrix that illustrates the relation between x_k and z_k. In our work, there is no control input (B=0), and the measurement z is the state directly (H=1). Assuming that the state does not change from time step to another, A is set to 1. w_{k-1} and v_k represent the process and measurement noises respectively. They are random variables assumed to be white and independent of each other, with $w_{k-1} \sim N(0, Q)$ and $v_k \sim N(0, R)$. Q and R represent the process noise covariance matrix and the measurement noise covariance matrix respectively. In our approach, we integrate Kalman Filter as a data pre-processing step, to benefit essentially from its filtering technique that may eliminate noises from resource usage data, whatever these noises are coming from the measurements technique or other factors, while still discovering the main load fluctuations.

To estimate a process, Kalman filter iteratively applies two computation steps: (a) the prediction step that projects the state estimation ahead of time, and (b) the correction step that adjusts the projected estimate based on an actual measurement value at that time. The equations used in each of the mentioned steps are as follows:

Prediction phase

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{4.3}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{4.4}$$

Correction phase

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (4.5)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (4.6)$$

$$P_k = (I - K_k H) P_k^- \quad (4.7)$$

Where $\hat{x}_k^-$ denotes the priori state estimate and $\hat{x}_k$ denotes the posteriori estimate at time k. Similarly, P_k^- is the priori estimate error covariance matrix, while P_k is the posteriori estimate error covariance matrix. K_k represents the Kalman Gain matrix. A high gain means that the filter mostly depends on the accurate measurements to estimate $\hat{x}_k$. Conversely, a low gain means that the state estimation mostly depends on the model predictions $\hat{x}_k^-$ calculated in the prediction phase.

4.4.2 Support Vector Regression

SVR is a statistical learning method that estimates a function f(x) by training a SVM model using observed data. In our case, the observed data represent the historical host resource utilizations. By performing time series forecasts, the workload data are first divided into input and output datasets (X and Y respectively). Each combination of input/output vectors (x_i, y_i) represents a training point. Eq. (4.8) defines both linear and non-linear regression prediction functions:

$$f(x) = w\emptyset(x) + b \quad (4.8)$$

Where $\emptyset$ is a mapping function that non-linearly maps x from "input space" to higher dimension feature space. To simplify the mapping, a Radial Basis Function (RBF) is employed for its easier computation and fewer parameters compared to other functions. $f(x)$ denotes the predicted value, w is a weight coefficient, and b is a bias. The main goal is to find the optimal weights and thresholds according to two essential criteria. The first is the flatness of the weights, which is defined in terms of minimum Euclidean norm (e.g., minimize $\frac{1}{2} \| w \|^2$). The second is the empirical risk R_{emp} minimization, which denotes the error generated by the prediction process of the value. R_{emp} is computed using the ε-insensitive loss function L^{ε}. Combining the mentioned two sub-objectives, the overall objective is to minimize the

regularized risk R_{reg} defined in Eq. (4.9) in order to find the flattest function that allows the error to remain within a threshold epsilon ε.

$$Minimize\ R_{reg} = \frac{1}{2} \| w \|^2 + R_{emp} = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{N} L^{\varepsilon}(y_i, f(x_i)) \tag{4.9}$$

Where

$$L^{\varepsilon}(y_i, f(x_i)) = \begin{cases} | y_i - f(x_i) - \varepsilon | & if\ | y_i - f(x_i) | \geq \varepsilon \\ 0 & otherwise \end{cases} \tag{4.10}$$

Where ε and C are user-defined constants. C determines the trade-off between the empirical and regularized risk. Finally, Slack variables, ξ_i and ξ_i^* should be added to estimate the error for underestimation and upper estimation of the actual value. In other terms, slack variables allow regression errors to exist up to the value of ξ_i and ξ_i^*, yet still satisfying the required conditions. Consequently, the equations are updated as follows:

$$Minimize\ R_{reg} = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{N} (\xi_i^* + \xi_i) \tag{4.11}$$

$$Subject\ to \begin{cases} f(x_i) - w\phi(x_i) - b \leq \varepsilon + \xi_i^* \\ w\phi(x_i) + b - f(x_i) \leq \varepsilon + \xi_i \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{4.12}$$

Algorithm 4.1 MSPR prediction algorithm

```
1:  Input: History_s^r, WS_r, n
2:  Output: U_{t+n}^r(s)  /* set of n predicted values */
3:  /* Preprocess data */
4:  Filtered_history = Kalman_filter (History_s^r)
5:  /* extract and divide training dataset into X and Y*/
6:  Training_data = extract (Filtered_history, WS_r)
7:  Xtrain, Ytrain = divide (Training_data)
8:  /* Train SVM model */
9:  svmTrain(Xtrain, Ytrain)
10: /* extract testing data */
11: Xtest = extract (Filtered_history, n)
12: U_{t+n}^r(s) = svmPredict (Xtest)
13: Return U_{t+n}^r(s)
```

4.4.3 MSPR Algorithm

In our work, the MSPR workload forecasting model is multi-resource in the sense that it is used to predict multiple types of resources including CPU and memory host utilizations, and the network bandwidth received and transmitted. For each resource type $r \in R$, a different window size WS and a number of prediction steps n can be set. MSPR model performs also multi-step-ahead predictions, meaning that it forecasts multiple future resource utilization values instead of just one value. Relying on one future resource utilization value to judge the host state, may lead to inaccurate estimations. For this fact, at each time t, our algorithm predicts n values of resource usage in order to estimate the future trend of the host resource consumption allowing us to perform the required resource management actions before encountering serious problems (e.g., SLA violation, QoS degradation, etc.). As shown in Algorithm 4.1, for each resource r, MSPR takes as input the host's historical data $History_s^r$, the pre-specified WS_r and n. The data are first filtered by Kalman filter and then divided into X and Y to train the SVM model as explained previously. Finally, the trained model is used to predict the future n values of the host's resource usage.

4.5 Workload consolidation approach

In this section, we present our workload consolidation approach based on the multi-resource and multi-step prediction model discussed in the previous section. We first describe all algorithms that constitute this approach, namely, the proposed Overload Detection algorithm based on MSPR prediction model (OD-MSPR), and Underload Detection algorithm based on MSPR prediction model (UD-MSPR), and the placement strategy. Then, we analyze the overall time complexity of this proposal.

4.5.1 Overload detection algorithm

In our approach, a server s is considered overloaded, if one of the following situations is met:

- It is overloaded in both current and future utilizations of at least one of its resources R. Precisely, if its current utilization $U_t^r(s)$ and the average of its n predicted utilizations $U_{t+n}^r(s)$ of any of its resources $r \in R$ exceed the upper threshold T^{up}.

  $for\ any\ r \in R$: $\quad U_t^r(s) > T_r^{up}\ and\ Avg\left(\widehat{U_{t+n}^r(s)}\right) > T_r^{up}$

- It is currently working normally but will be overloaded in the future time slots in at least one resource type $r \in R$.

  $for\ any\ r \in R$: $\quad Avg\left(\widehat{U_{t+n}^r(s)}\right) > T_r^{up}$

T_r^{up} is an upper adaptive threshold based on the Median Absolute Deviation technique. We assumed that a different upper threshold for each resource type r may be needed. The resources considered in our decision-making process are: CPU, memory, and bandwidth received by the server and transmitted. An overloading situation is detected when a server is overloaded in at least one of these resources (e.g., ***if*** $Avg\ (\ \widehat{U_{t+n}^{CPU}(s)}) > T_{CPU}^{up}$ *or* $Avg\ (\ \widehat{U_{t+n}^{mem}(s)}) > T_{mem}^{up}$ *or* $Avg\ (\ \widehat{U_{t+n}^{bwR}(s)}) > T_{bwR}^{up}$ *or* $Avg\ (\ \widehat{U_{t+n}^{bwT}(s)}) > T_{bwT}^{up}$).

As discussed previously, relying only on the current resource utilization of the host to decide its stats, may lead to unreliable decisions. An example of a CPU utilization trace of a cloud server is shown in Figure 4.1, borrowed from (Hieu et al., 2020). Assuming that the upper threshold for overload detection is set to 80%, depending on only the current server utilization will result in many false overloading detection decisions marked by circles due to sudden increases in its utilization. We can recognize that the load in all these time slots decreases rapidly in the next slots and there is no real overloading problem on the server. An effective approach should detect an overloading state for this server only in the period between 600 and 670 minutes (marked by a rectangle) because its utilization exceeds the threshold in both the current and future periods of time. The pseudocode of the proposed overload detection algorithm (OD-MSPR) is given in Algorithm 4.2. It takes an active server $s \in S_{active}$ as input, and then decides whether it is overloaded or not. In particular, for each resource $r \in R$, it verifies if there is sufficient historical data for prediction. If data is not sufficient, the decision is made by comparing the current resource utilization with the threshold T_r^{up} (Steps 5-8). Otherwise, it predicts the future n utilizations of this resource using MSPR prediction model and compares the average of these predicted values with the threshold (Steps 9-14). At the end, the algorithm returns true if an overloading problem is detected in one of the server's resources.

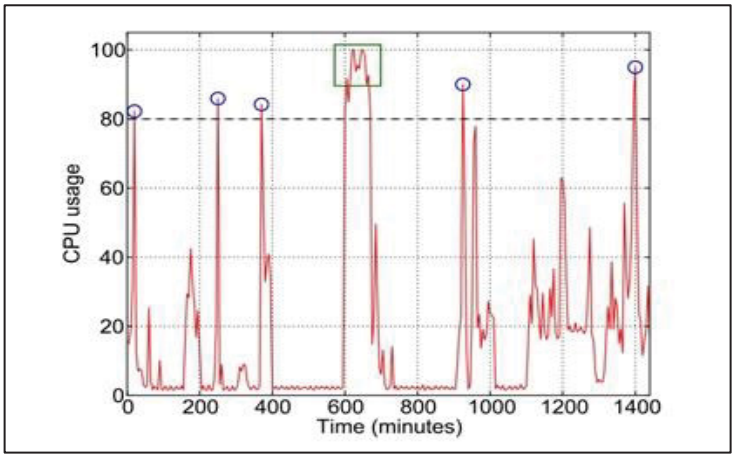

Figure 4.1 CPU utilization trace of a cloud server
Taken from Hieu et al. (2020, p. 190)

Algorithm 4.2 OD-MSPR

1: ***Input***: $s \in S_{active}$
2: ***Output***: *Boolean decision if s is overloaded or not*
3: ***For*** $\forall r \in R$ ***do***
4: *Get* $History_s^r$, T_r^{up}, WS_r, n
5: $T_r^{up} = getHistoryMAD(History_s^r)$
6: ***if*** *Length* ($History_s^r$) < WS_r ***then***
7: ***if*** $U_t^r(s) > T_r^{up}$ ***then***
8: $return\ true$
9: ***end***
10: ***else***
11: $\widehat{U_{t+n}^r(s)}$ = **MSPR** ($History_s^r$, WS_r, n)
12: ***if*** *Avg* ($\widehat{U_{t+n}^r(s)}$) > T_r^{up} ***then***
13: $return\ true$
14: ***end***
15: ***end***
16: ***End***
17: ***Return*** $false$

4.5.2 Underload detection algorithm

In this algorithm, a server s is defined as underloaded, if one of the following conditions is satisfied:

- It is underloaded in both current and future utilizations of all its resources R. In particular, if its current utilization $U_t^r(s)$ and the average of its multiple predicted utilization values $U_{t+n}^r(s)$ for each of its resources $r \in R$ are below the lower thresholds T_r^{Under}.

  $for\ all\ r \in R$: $U_t^r(s) <= T_r^{Under}\ and\ Avg\left(\widehat{U_{t+n}^r(s)}\right) <= T_r^{Under}$

- It is predicted to be underloaded in the future in all its resources R.

  $for\ all\ r \in R$: $Avg\left(\widehat{U_{t+n}^r(s)}\right) <= T_r^{Under}$

In our approach, R includes CPU, memory, and bandwidth received and transmitted. Thus, an underloading state is detected, if and only if the server is underloaded in all these resources (e.g., ***if*** *Avg*($\widehat{U_{t+n}^{CPU}(s)}$) <= T_{CPU}^{under} *and Avg* ($\widehat{U_{t+n}^{mem}(s)}$) <= T_{mem}^{under} *and Avg* ($\widehat{U_{t+n}^{bwR}(s)}$) <=

T_{bwR}^{under} *and Avg (* $\widehat{U_{t+n}^{bwT}(s)}$ *) <=* T_{bwT}^{under})). We assumed that a different lower threshold may be required for each resource. If an underloaded server is found, all VMs or containers hosted on this server should be migrated to other hosts if possible, and consequently, it will be switched to a low-power mode to save energy. The detailed pseudocode of the underload detection algorithm based on MSPR prediction model is illustrated in Algorithm 4.3. It iterates through the set of active servers S_{active} and searches if there is any underloaded host. Note that the overloaded servers detected by Algorithm 4.2 are excluded from S_{active}. To check a server state, it iterates through each of its resources $r \in R$ and verifies if the available historical data are enough for prediction. If no sufficient data are available to predict the resource, the algorithm compares the server's current utilization with the threshold. Otherwise, it compares the average of the future utilizations predicted by MSPR model, with the pre-specified threshold. The server is not considered underloaded if one of its resources exceeds its lower threshold.

Algorithm 4.3 UD-MSPR

19: ***Input****:* S_{active}
20: ***Output****: An underloaded server*
21: ***For*** $\forall s \in S_{active}$ ***do***
22: *IsUnderloaded = true*
17: ***For*** $\forall r \in R$ ***do***
23: *Get* $History_s^r$, T_r^{Under}, WS_r, n
18: ***if*** *Length (*$History_s^r$*)* < WS_r ***then***
19: ***if*** $U_t^r(s) > T_r^{Under}$ ***then***
24: *IsUnderloaded = false*
25: *Break*
26: ***end***
20: ***else***
27: $\widehat{U_{t+n}^r(s)}$ = **MSPR** *(*$History_s^r$, WS_r, n*)*
28: ***if*** *Avg (*$\widehat{U_{t+n}^r(s)}$*)* > T_r^{Under} ***then***
29: *IsUnderloaded = false*
30: *Break*
31: ***end***
32: ***End***
33: ***if (****IsUnderloaded) then*
34: ***Return*** *s*
35: ***end***
36: ***End***
37: ***Return*** *Null*

4.5.3 Migration and placement

After detecting overloaded and underloaded servers, the next step is to perform some migrations for the VMs or containers hosting the running applications on these servers. To perform a formal comparison between our approach and the techniques proposed in (Beloglazov & Buyya, 2012), we have re-used their VM selection approach and placement strategy. In particular, we have used the Minimum Migration Time (MMT) to select the VMs to migrate from overloaded hosts, and the Power Aware Best Fit Decreasing (PABFD) strategy to find destination servers for migrated VMs. However, these algorithms are modified to use our OD-MSPR algorithm. MMT-MSPR algorithm (Algorithm 4.4) iterates through the list of VMs V^s hosted on an overloaded server s and then selects for migration the VMs that have the least migration time. Migration time is measured by dividing the RAM utilized by a VM v by the available network bandwidth: $D_v^{migration} = \frac{RAM_v}{B_v}$. A set of VMs may be selected until the overloading issue is solved. Thus, after each VM selection, the algorithm verifies if the server will remain overloaded after deallocating the selected VM or not (Steps 14-17). This verification is done by calling our OD-MSPR algorithm explained in section 4.5.1.

Algorithm 4.4 MMT-MSPR algorithm

```
1:  Input: s ∈ S_over
2:  Output: List vmsToMigrate
3:  While (true) do
4:      Set min_time = MAX
5:      CandidateVM = NULL
6:      foreach v ∈ V^s do
7:          D_v^migration = RAM_v / B_v
8:          if D_v^migration < min_time then
9:              min_time = D_v^migration
10:             CandidateVM = v
11:         end
12:     end
13:     vmsToMigrate.add (CandidateVM)
14:     /* implicity call OD-MSPR(s) *
15:     if overloadedAfterDeallocation (s, v) = false then
16:         Break
17:     end
18:   end
19: Return vmsToMigrate
```

PABFD-MSPR strategy (Algorithm 4.5) iterates through the list of VMs to migrate and tries to find a destination server for each that meets certain criteria. First, the destination host should have sufficient capacity to meet the VM resource requirements in terms of CPU, memory, bandwidth, and disk (Step 7). Second, the candidate server should not become overloaded after hosting the VM (Steps 9-11). To verify the host state, the algorithm simulates the VM allocation and then uses OD-MSPR algorithm (Algorithm 4.2) to check the server state. Third, the selected server should have the least increase in its power consumption caused by this allocation because energy consumption is one of our main objectives in this work (Steps 12-17). At the end, the algorithm returns the migration map that includes the suitable destination hosts for the target VMs. Once the VMs to be migrated and their destinations are selected, a pre-copy live migration is applied to move them from their current hosts to the chosen ones.

Algorithm 4.5 PABFD-MSPR

```
1: Input: S_active, List vmsToMigrate
2: Output: Migration Map
3: For ∀v ∈ vmsToMigrate do
4:     minPower = Max
5:     destinationServer = NULL
6:     For ∀s ∈ S_active do
7:       if s.hasSufficientCapacity(v) then
8:           /* implicity call OD-MSPR(s) */
9:           if overloadedAfterAllocation(v, s) then
10:                  Continue
11:              end
12:              oldPower = s.getPower()
13:              newPower = estimatePowerAfterAllocation(v, s)
14:              powerDiff = newPower – oldPower
15:              if powerDiff < minPower then
16:                  minPower = powerDiff
17:                  destinationServer = s
18:              end
19:          end
20:      end
21:      if destinationServer is not NULL then
22:          migrationMap.add(v, destinationServer)
23:      end
24:    End
25:    Return migrationMap
```

Note that, in this work, we have focused mainly on resource prediction, overload detection, and underload detection parts. We have combined our proposed techniques (MSPR model, OD-MSPR, UD-MSPR) with simple VM selection and placement strategies to conduct our test experiments. However, our techniques can be combined with other advanced strategies, and be employed to consolidate workloads for other types of applications or services (e.g., containerized applications by using container migration strategies, virtual network functions consolidation by considering some migration constraints related to their service function chain requirements, etc.).

4.5.4 Overall approach

The overall predictive workload consolidation approach is presented in Algorithm 4.6. It is executed periodically to manage the cloud resources in two sequential procedures: (a) Overload Avoidance Phase (OAP) (Steps 2-10); and (b) Resource Wastage Avoidance Phase (RWAP) (Steps 11-27). OAP aims to release some resources from overloaded servers to avoid SLA violations. It starts by checking the hosts' states and detecting overloaded ones using Algorithm 4.2 (OD-MSPR). Then, it selects the virtual resources to migrate from these servers using Algorithm 4.4 (MMT) and chooses the destination hosts for the migrated VMs by executing Algorithm 4.5 (PABFD-MSPR). To start RWAP, the list of active servers S_{active} is first updated to exclude the overloaded servers list S_{over} and the destination hosts $S_{destinations}$ selected in OAP phase, because these servers should not be turned off (Step 11). RWAP aims to switch off the underloaded servers to optimize resource utilization in the data center and save energy. Through continuous iterations, the algorithm checks if there is any underloaded server in the data center using Algorithm 4.3 (UD-MSPR). If an underloaded server is detected, it tries to find destination hosts for all virtual resources running on this server using Algorithm 4.5 (PABFD-MSPR). If and only if all hosted VMs can be migrated to other destinations, the underloaded server can be turned off. Otherwise, the server remains active and all migrations planned from this server are canceled. In the following, the time complexity of the overall approach is detailed.

Algorithm 4.6 Workload consolidation approach

```
1:  Input: S_active, V
2:  /* start of OAP Phase */
3:  for ∀s ∈ S_active do
4:      if OD-MSPR(s) then
5:          S_over.add (s)
6:          vmsToMigrate.add (MMT-MSPR( s ))
7:      end
8:  end
9:  migrationMap = PABFD-MSPR (S_active, vmsToMigrate)
10: S'_active = S_active − (S_over ∪ S_destinations)
11: /* Start of RWAP Phase */
12: While (true) do
13:     U_server = UD-MSPR (S'_active)
14:     if U_server = NULL then
15:         Break
16:     end
17:     Exclude U_server from S'_active
18:     S_under.add(U_server)
19:     MigrationMap2 = PABFD-MSPR (S_active, V^s)
20:     if migrationMap2 is complete then
21:         migrationMap.addAll(migrationMap2)
22:         U_server can be turned off after migrations
23:     else
24:         Discard migrationMap2
25:         U_server will remain active
26:     end
27: End
```

4.5.5 Complexity Analysis

In this section, we analyze step by step the time complexity of the overall predictive workload consolidation approach and its main phases (OAP and RWAP phases) described in section 4.5.4 and Algorithm 4.6. The following notation is used to facilitate the complexity analysis: A is the number of active servers in the system; N denotes the total number of virtual resources (VMs or containers), N_{V^s} represents the number of virtual resources running on a server s; $N_{V^{mig}}$ is the number of virtual resources selected for migrations; and H_s^r is the historical data length of each resource $r \in R$ of a server s; D is the dimensions or the number of considered resources R.

4.5.5.1 Complexity – OAP Phase

Starting with line 3 of Algorithm 4.6, the time complexity of the for loop is equal to the number of active servers $O(A)$. Inside the loop, Algorithm 4.2 (OD-MSPR) is called. Its time complexity depends mainly on MSPR algorithm (Algorithm 4.1). The time complexity of Kalman Filter is analyzed in (Valade et al., 2017) as $O(4n^3)$ with n is the state vector size. In our approach, kalman is used to filter historical data before proceeding with SVR prediction, and so its complexity is $O(4\,H_s^{r\,3})$. In (Abdiansah & Wardoyo, 2015), the time complexity of SVM in LibSVM library which was used to complete our implementation is discussed. According to their analysis, the worst complexity for svmPredict and svmTrain is $O(n^3)$ where n is the amount of data used in training and in prediction respectively. In our prediction model, WS represents the prediction window size and n is the number of prediction steps. Thus, the complexity of MSPR is $O(4\,H_s^{r\,3} + WS^3 + n^3)$. Going back to Algorithm 4.2 (OD-MSPR), the algorithm iterates through the R resources of the server (D dimensions) and calls MSPR to predict each of them. Its complexity is then $O(D \cdot (4\,H_s^{r\,3} + WS^3 + n^3))$. In line 6, MMT algorithm (Algorithm 4.4) is used to choose the VMs to migrate from an overloaded server. This algorithm also calls (OD-MSPR) to verify if the server remains overloaded after the deallocation of each selected VM. Its complexity is $O(N_{V^s} \cdot D \cdot (4\,H_s^{r\,3} + WS^3 + n^3))$. Therefore, the total complexity of the for loop (lines 3-8) is $O(A \cdot N_{V^s} \cdot D^2 \cdot (4H_s^{r\,3} + WS^3 + n^3)^2)$.

After the loop, Algorithm 4.5 (PABFD-MSPR) at line 9 is called to select destination hosts for migrated VM. Its complexity is $O(N_{V^{mig}} \cdot A \cdot \mathrm{D} \cdot (4H_s^{r\,3} + WS^3 + n^3))$ because it also uses OD-MSPR algorithm to check the status of the potential destination server after allocating the target VM. Hence, the total complexity of OAP phase is $O(A \cdot N_{V^s} \cdot D^2 \cdot (4H_s^{r\,3} + WS^3 + n^3)^2) + O(N_{V^{mig}} \cdot A \cdot \mathrm{D} \cdot (4H_s^{r\,3} + WS^3 + n^3))$. However, H_s^r, WS, n, and D are typically small numbers and the complexity can be represented by $O(A \cdot N_{V^s}) + O(N_{V^{migrate}} \cdot A)$.

4.5.5.2 Complexity – RWAP Phase

In RWAP phase (lines 12-27), Algorithm 4.3 (UD-MSPR) is called to find underloaded hosts. Its complexity is based also on MSPR model and can be illustrated by $O(A \cdot D \cdot (4H_s^{r3} + WS^3 + n^3))$. Then, Algorithm 4.5 (PABFD-MSPR) is executed to find destination hosts for the VMs running on the detected underloaded host. In this case, $N_{V^{mig}} = N_{V^s}$ and the complexity of the latest algorithm is $O(N_{V^s} \cdot A \cdot \mathrm{D} \cdot (4H_s^{r3} + WS^3 + n^3))$. Therefore, the complexity of this phase is $O(A^2 \cdot D^2 \cdot (4{H_s}^3 + WS^3 + n^3)^2 \cdot N_{V^s})$, and can be simplified to $O(A^2 \cdot N_{V^s})$.

4.5.5.3 Overall Complexity

The overall complexity can be calculated by the summation of the complexities of OAP and RWAP phases. Consequently, it is equal to $O(A \cdot N_{V^s}) + O(N_{V^{migrate}} \cdot A) + O(A^2 \cdot N_{V^s})$. Again, to simplify it, the total number of active servers can be approximated by dividing the total number of VMs by the number of VMs that can be allocated to a server ($A \approx \frac{N}{N_{V^s}}$). The modified complexity will be $O(N) + O\left(N_{V^{migrate}} \cdot \frac{N}{N_{V^s}}\right) + O(\frac{N^2}{N_{V^s}})$. Finally, the worst-case complexity is $O(N^2)$.

4.5.6 Performance Metrics

In this study, our main objective is to decrease energy consumption and minimize the violations rate of service level agreements (SLAs). To assess the effectiveness of our algorithms, we utilize the following metrics.

1. SLA violation

SLA represents a contractual agreement between a cloud service provider and its customers, defining the desired quality of service (QoS). Within the SLA, Service Level Objectives

(SLOs) specify the QoS measurements and constraints. Meeting these requirements is crucial for evaluating the quality of the cloud service and avoiding penalties. We evaluate SLA violations based on two SLO parameters: SLA violation due to host overloading issue ($SLOH$) and SLA violation due to resource under-provisioning per VM ($SLOVM$).

$SLOH$ measures the average ratio of time during which a host is fully utilized. When a host's resource utilization (e.g., CPU, memory, bandwidth) reaches 100%, it may fail to provide VMs with the necessary resources, resulting in degraded performance. $SLOH$ can be calculated using the following formula:

$$SLOH = \frac{1}{A}\sum_{i=1}^{A}\frac{\max_{R}(T_{O_i}^{r})}{T_{a_i}} \qquad \forall r \in R \tag{4.13}$$

Where A is the number of hosts; $T_{O_i}^{r}$ is the total time during which the host i experiences 100% utilization of resource r; T_{a_i} is the total time in which host i is active.

$SLOVM$ measures the average violation caused by resource under-provisioning to VMs. It is calculated by comparing the allocated amount of each resource r to the requested amount. The formula is as follows:

$$SLOVM = \sum_{r=1}^{R}\frac{1}{N}\sum_{j=1}^{N}\frac{requested^{r} - allocated^{r}}{requested^{r}} \tag{4.14}$$

Where N represents the number of VMs.

2. Energy consumption

We evaluate the total energy consumed by the physical machines in a data center. The energy consumed by each server i is calculated by summing the power consumption of each resource type $r \in R$ including CPU, memory, and bandwidth (equation 4.16). The power consumed by each resource type is calculated using the formulas 4.17-4.22.

$$Total\ Power = \sum_{i=1}^{A} P_i \tag{4.15}$$

$$P_i = \sum_{r=1}^{R} P_i^r \qquad \forall\, r \in R \tag{4.16}$$

$$P_i^{CPU} = specPower(u^{CPU}) \tag{4.17}$$

$$P_i^{RAM} = u^{RAM} \,.\, P_{max}^{RAM} \tag{4.18}$$

$$P_{max}^{RAM} = \frac{RAM_i}{x} \tag{4.19}$$

$$P_i^{BW} = P_i^{static} + P_i^{dynamic} \tag{4.20}$$

$$P_i^{static} = P_{NIC}^{idle} + \sum_{l=1}^{L} P_l \tag{4.21}$$

$$P_i^{dynamic} = u^{BW} \,.\, P_{max}^{BW} \tag{4.22}$$

$$P_{max}^{BW} = \frac{BW_i}{y} \tag{4.23}$$

The processing power measurements (consumed by CPU) are derived from real data obtained from SPECpower benchmark results (« The SPECpower Benchmark », s.d.). Table 4.1 provides the power consumption of the servers HP G4 and G5 at various loads. The power consumed by the RAM is calculated by multiplying the RAM utilization of the server by the maximum potential power consumption of this resource (equation 4.18). Equation 4.19 is used to calculate this maximum power, where RAM_i represents the total memory of server i, and x is an input value that can be easily updated (Lin, Xu, He, & Li, 2017). To provide a specific

input value, we assume that each 3 GB of RAM consumes 1 watt. The power consumed by the bandwidth consists of static and dynamic power components. The static power is considered constant and is calculated by summing the idle power of the network card with the power increase in relation to the number of active links L (equation 4.21). In our testing, we assume that the utilized NIC is an intel Multiport (4*1G) with only one active link, and the idle power consumption is 9 watts (Sohan, Rice, Moore, & Mansley, 2010). Whereas, the dynamic power is associated to the bandwidth utilization of the server and is calculated using equation 4.22. The maximum potential power for this resource is calculated by dividing the total bandwidth of the server BW_i by an input value y. To specify this input value, we consider that the max active power of intel multiport (4*1G) NIC is 1 watt for 0.45 Gbps (Sohan et al., 2010).

3. Number of migrations

Minimizing the number of VM migrations is important to avoid negative impacts on application performance. Live migrations incur additional costs, including increased resource utilization on the source host, network bandwidth usage, service delay due to downtime during migration, and total migration time.

4. Execution Time

We also compare the algorithms based on their execution time. Specifically, we measure the average time required to complete an entire consolidation cycle, including the steps of overloaded host detection, underloaded host detection, VM selection for migration, and VM placement.

Table 4.1 Power consumption of hosts according to their CPU usage (in watts)

Server	Sleep	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
HP ProLiant G4	10	86	89.4	92.6	96	99.5	102	106	108	112	114	117
HP ProLiant G5	10	93.7	97	101	105	110	116	121	125	129	133	135

4.6 Experiments

4.6.1 Setup

4.6.1.1 Environment

We have conducted simulations using the CloudSim toolkit (Calheiros et al., 2011) to test our proposed algorithms. Our testing environment consists of 800 heterogeneous servers divided as follows: 400 HP ProLiant ML110 G4 machines with dual-core processors, each having 1860 MIPS, and 400 HP ProLiant ML110 G5 machines with dual-core, each having 2660 MIPS. Both server types are equipped with 4 GB of memory and 1 GB/s of bandwidth. The characteristics of VM instances are provided in Table 4.2. To implement our *MSPR* prediction model, LibSVM library (Chang & Lin, 2011) is used in Java. We have conducted tests using different threshold values (ranging from 20% to 30%) for underload detection, and different windows sizes (8, 12, 16, 20, 24, and 28) for prediction. The underload threshold T_r^{Under} is set to 30% and the prediction window size WS_r to 20 for all resources. However, it is possible to set different threshold and window size for each resource r. Additionally, we have set the number of prediction steps *n* to *3,* to predict three future utilizations of each resource type for the host. Nevertheless, our implementation is not limited to 3, and the value of n can be easily adjusted. All testing parameters are summarized in Table 4.3. These input values can be easily modified according to specific requirements.

Table 4.2 VM instances characteristics

VM Instance Type	CPU (MIPS)	RAM (GB)	Bandwidth (Mbits/s)
High-CPU medium instance	2500	0.85	100
Extra-large instance	2000	3.75	100
Small instance	1000	1.7	100
Micro instance	500	0.613	100

Table 4.3 Testing Parameters

Kalman filter	A	H	Q	R
	1	1	0.01	1
SVR	ε	kernel	γ	C
	0.1	RBF	0.0625	1
Consolidation and prediction	T_r^{Under}	WS_r	n	
	30%	20	3	
Arima	p	d	q	
	1	0	1	

4.6.1.2 Datasets

Our simulation utilizes two publicly available real-world datasets: Bitbrains (Shen et al., 2015) and Materna (Kohne et al., 2014)(Kohne et al., 2016). The fastStorage trace in the Bitbrains dataset comprises 1,250 VMs connected to high-speed storage area network (SAN) devices. The Rnd trace in the same dataset consists of 500 VMs that are connected to either fast SAN devices or slower Network Attached Storage (NAS) devices. The Rnd trace is further divided into three sub-traces, each corresponding to a specific month when the metrics were recorded. The Materna dataset includes three distinct traces, each representing one month of data collection. The first trace comprises 520 VMs, the second trace consists of 527 VMs, and the third trace encompasses 547 VMs. These traces are recorded over a three-month duration. To ensure an adequate number of VMs for testing purposes, we have combined certain traces, as indicated in Table 4.4. Before conducting the testing process, we have carried out a pre-processing phase on the datasets. This phase involves converting the datasets into a suitable format. Furthermore, by using the following linear transformation formula, we have normalized the resource utilization data.

$$x_k^n = \frac{x_k - x_{min}}{x_{max} - x_{min}} \tag{4.24}$$

where x_k^n represents the normalized resource utilization value calculated based on the original data x_k. x_{max} and x_{min} are the minimum and maximum values of x_k, respectively. Furthermore, we have performed data cleaning procedures to remove histories with insufficient data or null values. After completing the pre-processing phase, the obtained resource utilization dataset is used to conduct our testing.

Table 4.4 Datasets characteristics

Workloads	Datasets	Traces	Number of VMs	Number of servers
W1	Bitbrains	Trace Fast storage	1237	800
W2	Bitbrains	Rnd (3 traces)	1500	800
W3	Materna	Traces 1-3	1063	800
W4	Materna	Traces 1-2	1043	800
W5	Materna	Traces 3-2	1074	800

4.6.1.3 Benchmarks comparison

To demonstrate the efficiency of our approach, we have conducted a comparison with modified versions of consolidation techniques integrated into the Cloudsim toolkit (Beloglazov & Buyya, 2012). Specifically, we consider four consolidation strategies where overload detection depends on: Static Threshold (THR), InterQuartile Range (IQR), Median Absolute Deviation (MAD), and Local Regression (LR). These approaches have been adapted to be multi-resource and to consider all resources in their overload and underload decisions. Similar to our approach, they detect an overloading situation when a server is overloaded in at least one of its resources, including CPU, memory, and bandwidth. An underloading state is detected only if the server is underloaded in all of these resources. Two different versions of underload detection is considered for these benchmarks. In the first experiment, underloaded hosts are identified as those whose actual resource utilizations are lower than the underload thresholds for all resources. In the second experiment, our predictive underload detection algorithm, UD-MSPR, discussed in section 4.5.2., is incorporated into these benchmarks. This integration allows a stronger comparison between different overload detection techniques and a clearer interpretation of the obtained results. To perform a formal comparison, all algorithms use a common VM selection strategy, Minimum migration time (MMT), and employ the same VM

placement method explained in sub-section 4.5.3. All evaluations are made based on the performance metrics described in section 4.5.6.

In addition to the benchmark algorithms mentioned, we test our proposed consolidation approach against an Arima-based multi-resource consolidation technique. Instead of using the Kalman-SVR combination for resource utilization prediction, our consolidation technique is updated to employ Arima. The purpose of implementing this alternative technique is to evaluate our proposed approach against another predictive consolidation mechanism. Details of the experimental results are presented in the following sub-section.

4.6.2 Results and discussion

4.6.2.1 Experiment 1

Figures 4.2-4.6 illustrate a comparison of our MSPR-based consolidation approach with an alternative Arima-based consolidation approach and optimized multi-resource versions of Cloudsim benchmarks discussed in subsection 4.6.1.3. In this experiment, Cloudsim benchmarks identify underloaded hosts as those whose actual resource usage are below the underload thresholds for all resources.

From the results depicted in Figure 4.2, our MSPR-based consolidation approach outperforms all approaches in term of energy consumption reduction. It demonstrates a noteworthy reduction in total energy consumed in the datacenter, averaging 9.384%, 14.126%, 19.270%, and 23.223% lower compared to LR, MAD, IQR, and THR, respectively. Arima-based approach ranks second in minimizing energy consumption. Among the benchmark algorithms, LR performs optimally in terms of energy optimization. Notably, our UD-MSPR and OD-MSPR algorithms effectively identify underloaded and overloaded hosts, enabling the migration of VMs from these hosts to alternate machines and transitioning idle hosts into sleep mode to conserve energy.

Moreover, by estimating the trend of future resource utilizations, our approach allows for proactive measures to prevent overloading situations and potential SLA violations. Consequently, our approach significantly reduces SLA violations resulting from host overloading ($SLOH$) compared to the other Cloudsim benchmarks, as shown in Figure 4.3. MSPR-based approach yields the best outcomes in reducing $SLOH$, with the Arima-based approach ranking second. LR is the worst among the benchmarks in reducing $SLOH$ in the first two workloads, but THR is the worst in the others. In terms of the average SLA violation per VM caused by resource under-provisioning ($SLOVM$), Figure 4.4 demonstrates that both the MSPR and Arima-based approaches outperform the other methods across all tested datasets. The MSPR-based approach reduces the $SLOVM$ by an average of 95.871%, 77.953%, 81.075%, and 87.188% compared to LR, MAD, IQR, and THR, respectively. Moreover, it achieves a 15.278% average reduction in $SLOVM$ compared to the Arima-based approach.

Once overloaded and underloaded hosts are identified, effective VM migration plans can be applied to readjust resource allocations and alleviate the issue. Figure 4.5 presents the comparison results in terms of the number of migrations. Remarkably, our MSPR-based approach dramatically minimizes the number of migrations across all datasets compared to the Cloudsim benchmarks. For example, for dataset W1, the MSPR-based technique initiates 6378 migrations, while the other approaches perform 18695 (LR), 16940 (MAD), 18027 (IQR), 17944 (THR), and 7273 (Arima) migrations. Minimizing the number of migrations is desirable as it reduces system overhead, extra expenses, and potential violations. Furthermore, avoiding unnecessary migrations contributes to a more efficient reallocation process, resulting in reduced runtime encompassing VM selection and destination host determination. Figure 4.6 provides insights into the runtime performance, indicating that MSPR-based technique achieves lower execution time compared to other approaches. Arima-based technique occupies the second-best runtime. Overall, MSPR-based approach strikes a favorable balance between energy consumption and SLA violation, outperforming the compared algorithms in almost all studied metrics.

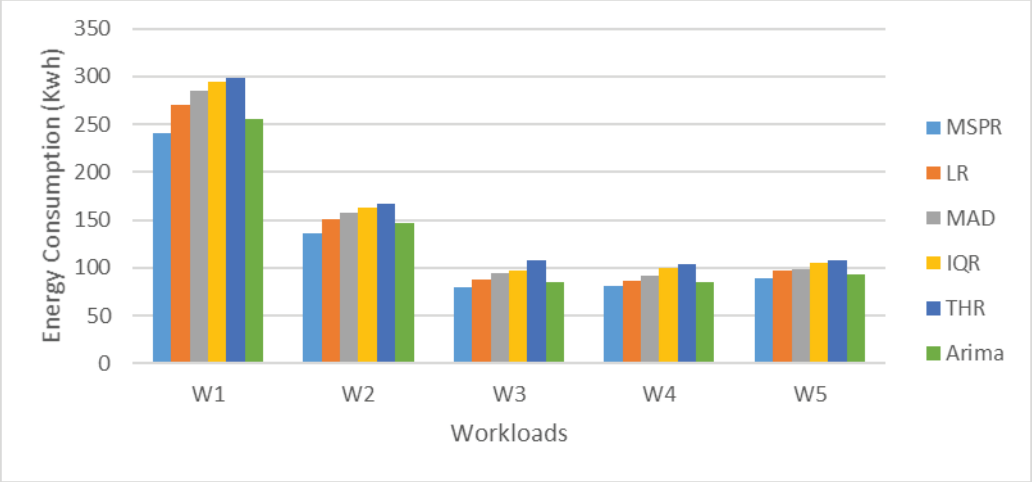

Figure 4.2 Comparison of energy consumption for 5 workloads- experiment 1

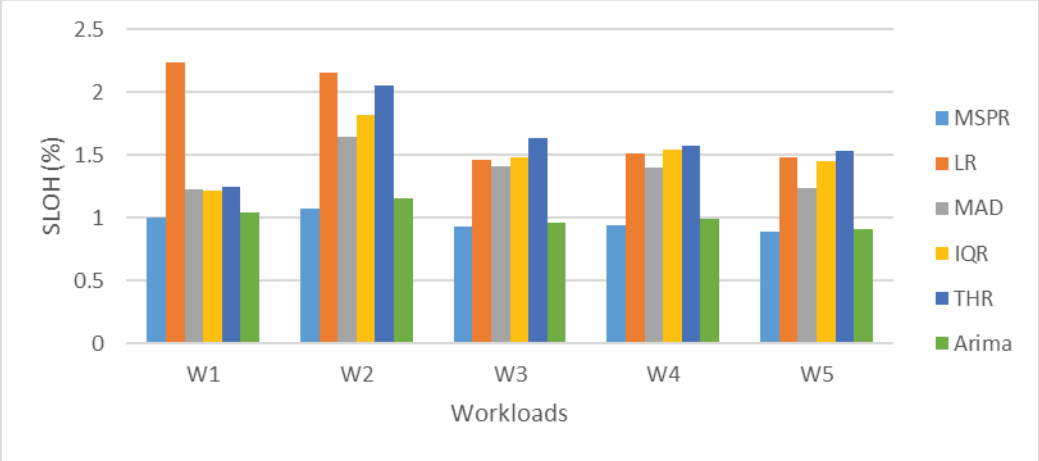

Figure 4.3 Comparison of SLOH metric for 5 workloads- experiment 1

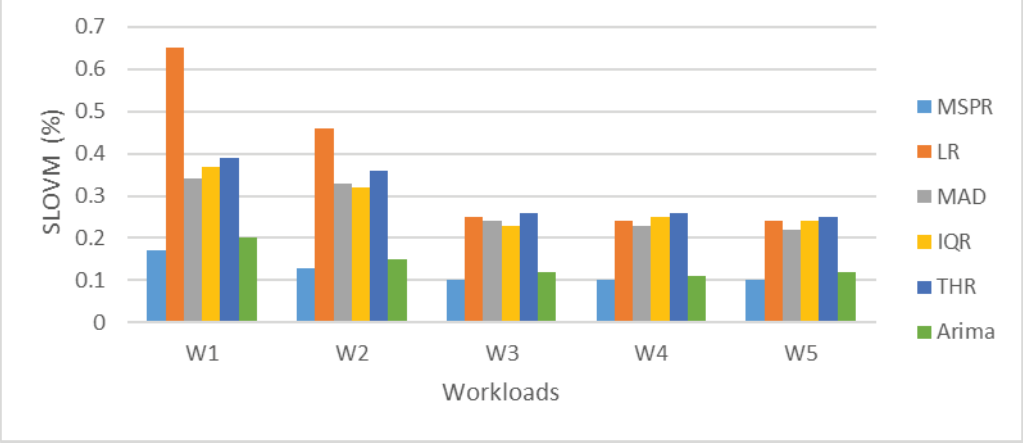

Figure 4.4 Comparison of the SLOVM metric for 5 workloads- experiment 1

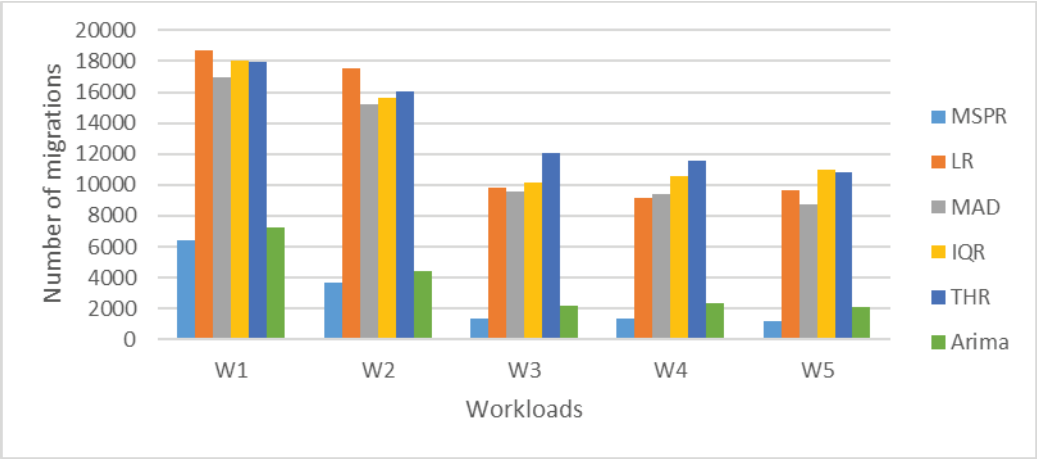

Figure 4.5 Comparison of number of migrations for 5 workloads - experiment 1

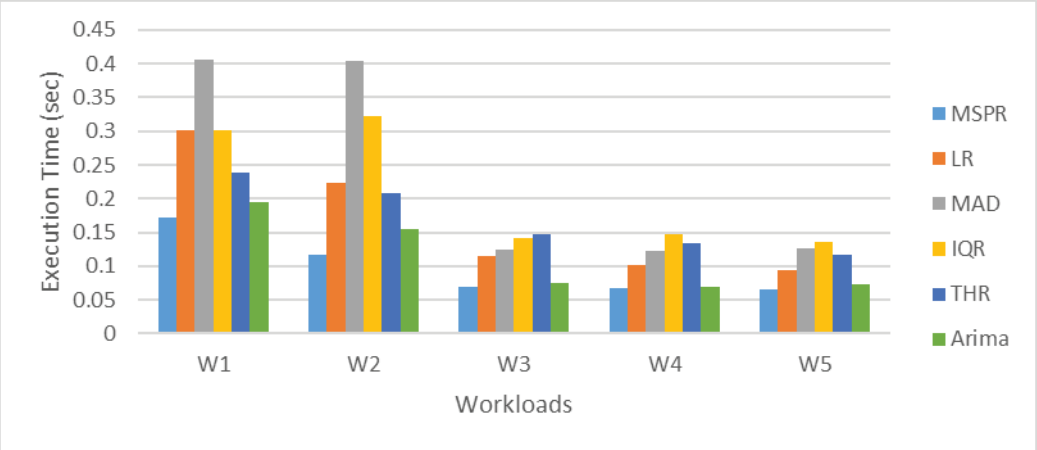

Figure 4.6 Comparison of execution time for 5 workloads - experiment 1

4.6.2.2 Experiment 2

Figures 4.7-4.11 provide a comparison of our consolidation approach with another modified versions of Cloudsim benchmark algorithms. In this experiment, these benchmark algorithms incorporate our proposed underload detection algorithm UD-MSPR explained in subsection 4.5.2. As a result, the differentiation lies only in the overload detection part, allowing us to focus on comparing the performance in handling overloading and verifying if the benchmark algorithms combined with our predictive underload detection can outperform our approach. It

is important to note that the testing results for the MSPR and Arima-based approaches remain the same as in experiment 1.

Figure 4.7 reveals that MSPR-based approach obtains the best results in terms of energy reduction. It also exhibits competitive performance, between Arima-based and LR-based techniques in the first two workloads (W1 and W2), and between Arima-based and MSPR-based approaches in the last three workloads (W3, W4 and W5). Threshold techniques (THR, IQR, and MAD) consume the highest amount of energy, with THR having the worst results.

Figure 4.8 highlights the superior performance of MSPR-based technique in minimizing $SLOH$ across most datasets. Figure 4.9 also indicates that MSPR achieves the lowest $SLOVM$ in all datasets. However, it is worth noting that the average difference in results between our approach and the other techniques is relatively smaller compared to experiment 1. This is primarily because the combination of our underload detection algorithm (UD-MSPR) with other approaches has reduced their $SLOH$ and $SLOVM$ violation rates.

In Figure 4.10, the number of VM migrations is compared among the different techniques. Our MSPR-based approach consistently performs the lowest number of migrations for resource reallocation. The difference in the average number of migrations is substantial between our approach and the other techniques: **76.486**% compared to LR, **84.113**% compared to MAD, **89.664**% compared to IQR, and **91.111**% compared to THR. It is worth noting that the benchmark algorithms also exhibit a reduction in the number of migrations compared to the results of experiment 1. Arima-based approach also performs significantly fewer migrations than these benchmarks.

Figure 4.11 reveals that LR and THR have competitive execution time and they outperform MSPR in terms of runtime. MSPR ranks the third. It is important to note that combining our UD-MSPR algorithm with the benchmark approaches results in reduced runtime compared to the results of experiment 1. This combination enables the benchmark algorithms to outperform also Arima-based approach in terms of execution time.

In summary, incorporating our underload detection algorithm into the benchmark approaches leads to improved performance results, significant reduction in SLA violation rates, the number of VM migrations, and runtime. However, our approach achieves the best overall results and outperforms these approaches, in terms of SLA violations, number of migrations, and power consumption.

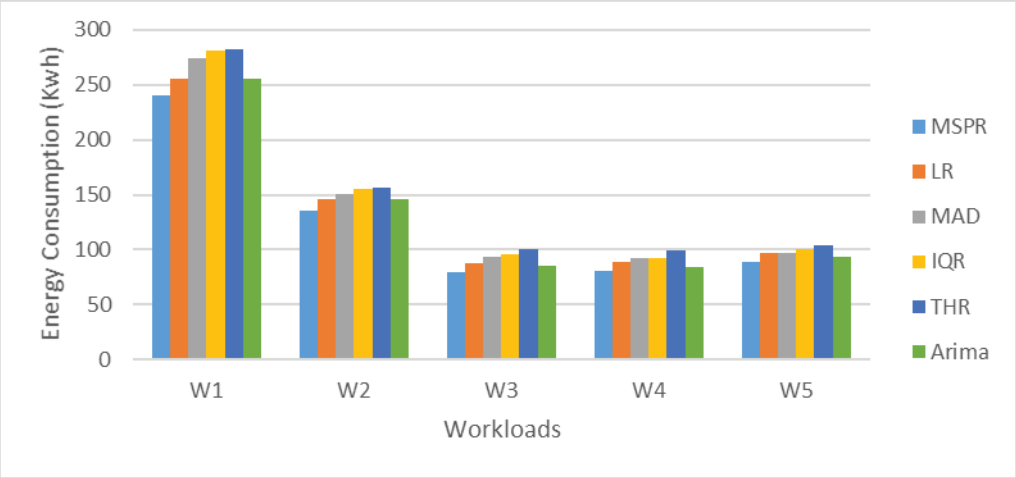

Figure 4.7 Comparison of energy consumption for 5 workloads- experiment 2

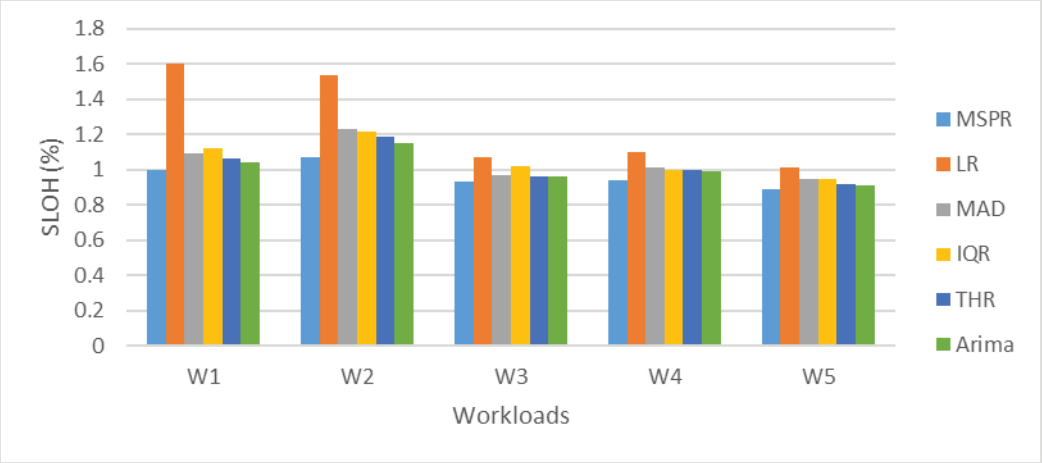

Figure 4.8 Comparison of SLOH metric for 5 workloads- experiment 2

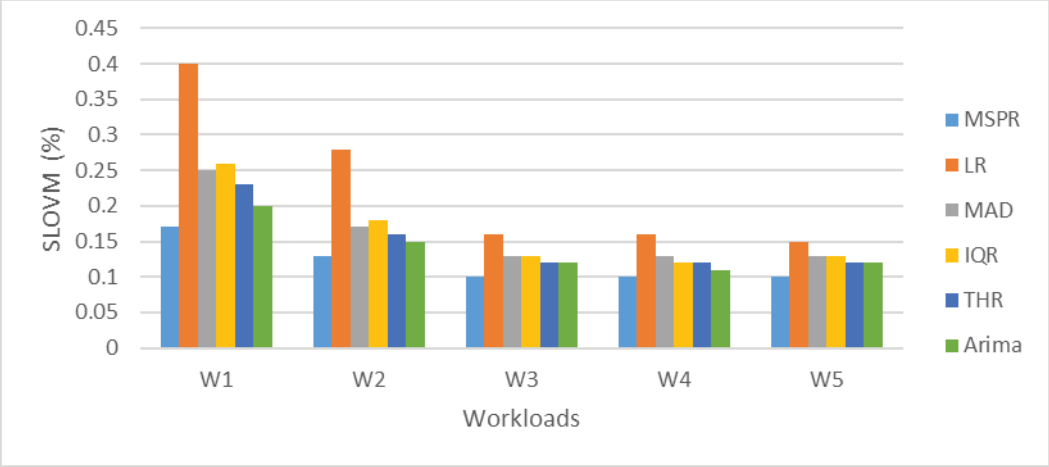

Figure 4.9 Comparison of SLOVM metric for 5 workloads- experiment 2

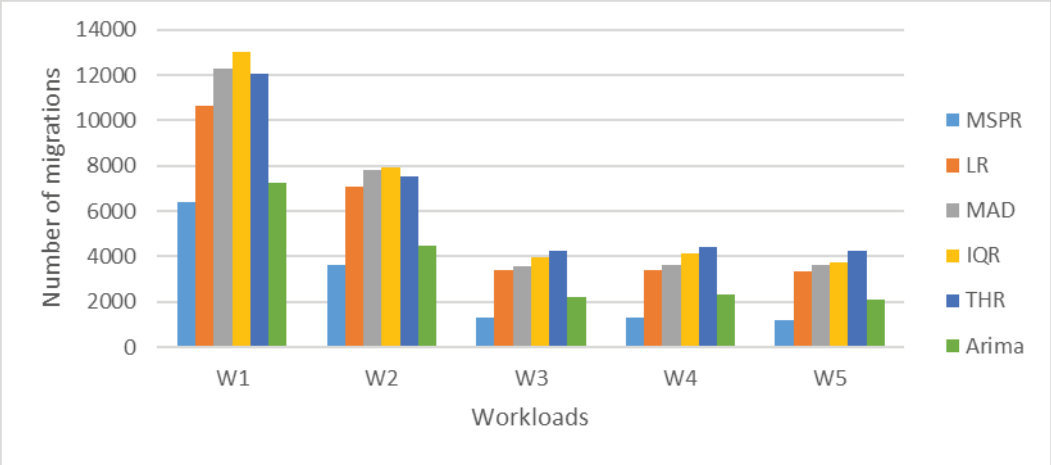

Figure 4.10 Comparison of number of migrations for 5 workloads- experiment 2

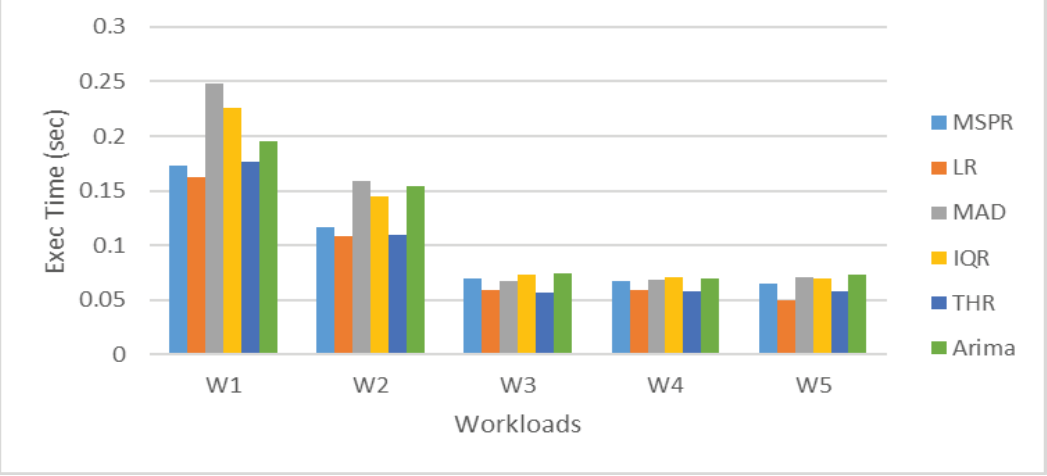

Figure 4.11 Comparison of execution time for 5 workloads- experiment 2

4.7 Conclusion

In this paper, we propose a predictive workload consolidation mechanism that aims to reduce energy consumption, minimize SLA violations, and optimize the resource re-allocation process. Our approach consists of several components. Firstly, we introduce a multi-step-ahead and multi-resource prediction model, called MSPR that combines the Kalman Filter with Support Vector Regression (SVR). This model allows us to forecast the future resource utilization of hosts, including CPU, memory, and bandwidth. By leveraging historical data and utilizing the strengths of both Kalman Filter and SVR, we can accurately predict resource demands. Secondly, we present novel techniques based on MSPR model, for detecting underload and overload states of hosts. These techniques consider the current and predicted resource utilization trend across all resource types (CPU, memory and bandwidth) for each host to proactively estimate its state. To make informed decisions about host states, we calculate an adaptive upper-threshold for overload detection using Median Absolute Deviation (MAD) based on historical data for each resource type. Additionally, we provide the flexibility to specify different underload thresholds and prediction window sizes for each resource type. To pursue testing experiments, we combined our proposed techniques with existing VM selection and VM placement strategies. It is worth noting that our techniques are not limited to VMs and can be combined with other selection and placement methods, such as those designed for containers. Although VM placement is re-used from Cloudsim platform, we updated it to incorporate our overload detection technique for the identification of potential overloading issues on candidate destination hosts after migrating a VM.

To evaluate our proposed techniques, we conducted simulations using real-world workload traces from Bitbrains and Materna. We compared our approach against modified and optimized versions of benchmark algorithms integrated into Cloudsim, including MAD-based, IQR-based, THR-based, and LR-based approaches. These benchmarks are updated to consider multi-resource aspects in their host state estimation. In addition to these approaches, we implemented another predictive consolidation technique by combining our overload and underload techniques with a multi-resource Arima prediction model, as an alternative to the

Kalman-SVR model. This allowed us to compare the performance of our proposal against another predictive consolidation mechanism. Our experimental results demonstrated the effectiveness of the Kalman-SVR-based consolidation approach in minimizing defined cost metrics and outperforming the other algorithms. For future work, we aim to combine our proposed algorithms with more advanced selection and placement strategies for VMs or containers. These strategies play a crucial role in resource reallocation decisions and can affect the overall system performance. Future work can also involve testing our proposals on real Cloud environment using monitoring tools instead of relying on existing resource utilization datasets.

Acknowledgment

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Grant no. RGPIN-2019-05250.

CONCLUSION

In recent years, virtualization technologies and cloud solutions have revolutionized the way computing resources are managed and utilized. Virtualization technologies like virtual machines (VMs), containers, and serverless architectures have enabled cloud providers to offer scalable, flexible, and cost-effective services to their customers. However, with the increasing demand for cloud services, resource management has become a critical issue in the cloud. Workload consolidation approaches, resource utilization prediction, resource scaling and migration are some of the mechanisms that have been proposed to manage resources in virtualized environments like the cloud. Workload consolidation approaches aim to maximize the utilization of resources by consolidating multiple workloads on a smaller number of physical servers. Resource utilization prediction techniques anticipate the future resource usage based on historical data. Resource scaling techniques dynamically adjust the amount of resources allocated to an application based on its current usage. Resource migration techniques allow applications to be moved between physical servers to optimize resource utilization. To create an efficient resource management system, these techniques need to be employed together in a coherent and interrelated manner.

The objective of this thesis is to propose generic resource management techniques that can be applied in virtualized environments, independent of the type of executed services or the virtualization technology used. These techniques aim to ensure efficient resource utilization, reduce energy consumption, and meet Service Level Agreement (SLA) requirements. To achieve this objective, the thesis presents several significant contributions. It proposes innovative techniques for dynamic resource adaptation in NFV-cloud environments, including resource scaling and migration. The research addresses the challenges of variability in workloads, diversity in applications, and conflicting optimization goals. It formulates the resource adaptation problem by integrating horizontal scaling, vertical scaling, and migration strategies, and develops an Integer Linear Programming (ILP) model to provide optimal solutions. Additionally, multi-objectives decision-making metaheuristic algorithms based on NSGAII, CRO, and PSO are proposed for real-time resource adaptation decisions. Moreover,

the thesis explores proactive resource reallocation by combining a multi-step-ahead workload prediction model. By integrating Kalman filter and support vector regression, host resource utilization including CPU, memory and bandwidth, are accurately anticipated. Building upon this work, an optimized predictive consolidation approach is introduced, considering proactive host state estimation strategies (overload and underload detection) and incorporating adaptive thresholds. It aims to overcome the limitations of existing approaches, ensures accurate assessment of the overall host state and reliable migration decisions, prevent energy waste, avoid performance degradation, and SLA violations. The effectiveness of the proposed techniques is validated through extensive experiments, using various datasets (Planetlab, Materna, and Bitbrains) and simulator (Cloudsim) demonstrating their potential to enhance resource management in virtualized environments.

These contributions are presented as three distinct journal papers. It is important to acknowledge that building a complete resource management system may require additional techniques and further research. One potential area for future exploration is the development of advanced virtual machine or container selection strategy for migration and placement technique. These strategies can significantly impact the effectiveness and efficiency of the resource management system. Another possible enhancement is integrating the prediction model with the resource adaptation meta-heuristic algorithms. This integration would enable proactive decision-making by allowing the algorithms to utilize the predicted resource needs in their resource adaptation processes. In addition, a coordination entity can be added to filter the decisions made by both the consolidation framework and the resource adaptation algorithms. This coordination entity would generate the final decisions and apply them accordingly. Moreover, the framework can benefit from incorporating monitoring tools instead of using existing datasets or generating random workloads. By gradually adding these elements, the current resource management capabilities of the system can be further refined. Furthermore, future research can involve investigating how to efficiently manage resources in hybrid cloud-edge environments (Zhao, 2023) or serverless architectures (Hoseinyfarahabady et al., 2021)(Cordingly et al., 2020). These environments introduce new challenges and constraints for the resource reallocation problem in a virtualized environment(Gill et al., 2022).

LIST OF BIBLIOGRAPHICAL REFERENCES

Abdelaal, M. A., Ebrahim, G. A., & Anis, W. R. (2021). Efficient placement of service function chains in cloud computing environments. *Electronics (Switzerland)*, *10*(3), 1-22. https://doi.org/10.3390/electronics10030323

Abdiansah, A., & Wardoyo, R. (2015). Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM. *International Journal of Computer Applications*, *128*(3), 28-34. https://doi.org/10.5120/ijca2015906480

Abdullah, L., Li, H., Al-Jamali, S., Al-Badwi, A., & Ruan, C. (2020). Predicting Multi-Attribute Host Resource Utilization Using Support Vector Regression Technique. *IEEE Access*, *8*, 66048-66067. https://doi.org/10.1109/ACCESS.2020.2984056

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2017). Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER. *IEEE International Conference on Cloud Computing, CLOUD*, *2017-June*, 472-479. https://doi.org/10.1109/CLOUD.2017.67

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges IEEE TRANSACTIONS ON SERVICES COMPUTING, MANUSCRIPT ID 1 Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing (TSC)*, *11*(2), 430-447. https://doi.org/10.1109/TSC.2017.2711009

Ali, A., Pinciroli, R., Yan, F., & Smirni, E. (2018). CEDULE: A scheduling framework for burstable performance in cloud computing. *Proceedings - 15th IEEE International Conference on Autonomic Computing, ICAC 2018*, 141-150. https://doi.org/10.1109/ICAC.2018.00024

Alzahrani, E. J., Tari, Z., Zeephongsekul, P., Lee, Y. C., Alsadie, D., & Zomaya, A. Y. (2016). SLA-Aware Resource Scaling for Energy Efficiency. *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 852-859. https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0123

Amiri, M., & Mohammad-Khanli, L. (2017). Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, *82*(December 2016), 93-113. https://doi.org/10.1016/j.jnca.2017.01.016

Apostolopoulos, P. A., Tsiropoulou, E. E., & Papavassiliou, S. (2019). Risk-aware social cloud computing based on serverless computing model. *2019 IEEE Global Communications Conference, GLOBECOM 2019 - Proceedings*. https://doi.org/10.1109/GLOBECOM38437.2019.9013182

Arshad, U., Aleem, M., Srivastava, G., & Lin, J. C. W. (2022). Utilizing power consumption and SLA violations using dynamic VM consolidation in cloud data centers. *Renewable and Sustainable Energy Reviews*, *167*(July), 112782. https://doi.org/10.1016/j.rser.2022.112782

Aslanpour, M. S., & Branch, S. (2016). SLA-Aware Resource Allocation for Application Service Providers in the Cloud. *Second International Conference on Web Research (ICWR) SLA-Aware*, 31-42.

Astudillo, L., Melin, P., & Castillo, O. (2015). Introduction to an optimization algorithm based on the chemical reactions. *Information Sciences*, *291*(C), 85-95. https://doi.org/10.1016/j.ins.2014.08.043

Awad, M., Kara, N., & Edstrom, C. (2022). SLO-aware dynamic self-adaptation of resources. *Future Generation Computer Systems*, *133*, 266-280. https://doi.org/10.1016/j.future.2022.03.018

Awad, M., Kara, N., & Leivadeas, A. (2022). Utilization prediction-based VM consolidation approach. *Journal of Parallel and Distributed Computing*, *170*, 24-38. https://doi.org/10.1016/j.jpdc.2022.08.001

B, A. N., Gounaris, A., & Sioutas, S. (2016). Cloud Elasticity: A Survey, *10230*, 151-167. https://doi.org/10.1007/978-3-319-57045-7

Banerjee, S., Roy, S., & Khatua, S. (2021). Efficient resource utilization using multi-step-ahead workload prediction technique in cloud. *Journal of Supercomputing*, *77*(9), 10636-10663. https://doi.org/10.1007/s11227-021-03701-y

Basmadjian, R., Niedermeier, F., & De Meer, H. (2012). Modelling and analysing the power consumption of idle servers. *2012 Sustainable Internet and ICT for Sustainability, SustainIT 2012*.

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, *28*(5), 755-768. https://doi.org/10.1016/j.future.2011.04.017

Beloglazov, A., & Buyya, R. (2012). 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. https://doi.org/10.1002/cpe

Benmakrelouf, S., Kara, N., Tout, H., Rabipour, R., & Edstrom, C. (2019). Resource needs prediction in virtualized systems: Generic proactive and self-adaptive solution. *Journal of Network and Computer Applications*, *148*(102443). https://doi.org/10.1016/j.jnca.2019.102443

Benmakrelouf, S., St-Onge, C., Kara, N., Tout, H., Edstrom, C., & Lemieux, Y. (2020). Abnormal behavior detection using resource level to service level metrics mapping in virtualized systems. *Future Generation Computer Systems*, *102*, 680-700. https://doi.org/10.1016/j.future.2019.07.051

Bharanidharan, G., & Jayalakshmi, S. (2021). Predictive virtual machine placement for energy efficient scalable resource provisioning in modern data centers. *Proceedings of the 2021 8th International Conference on Computing for Sustainable Global Development, INDIACom 2021*, 299-305. https://doi.org/10.1109/INDIACom51348.2021.00052

Bharany, S., Sharma, S., Khalaf, O. I., Abdulsahib, G. M., Al Humaimeedy, A. S., Aldhyani, T. H. H., … Alkahtani, H. (2022). A Systematic Survey on Energy-Efficient Techniques in Sustainable Cloud Computing. *Sustainability (Switzerland)*, *14*(10), 1-89. https://doi.org/10.3390/su14106256

Bhattacharjee, A., Chhokra, A. D., Kang, Z., Sun, H., Gokhale, A., & Karsai, G. (2019). BARISTA: Efficient and scalable serverless serving system for deep learning prediction services. *Proceedings - 2019 IEEE International Conference on Cloud Engineering, IC2E 2019*, 23-33. https://doi.org/10.1109/IC2E.2019.00-10

Bouabdallah, R., Lajmi, S., & Ghedira, K. (2016). Use of reactive and proactive elasticity to adjust resources provisioning in the cloud provider. *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 1155-1162. https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0162

Calheiros, R. N., Ranjan, R., Beloglazov, A., Buyya, R., & De Rose, C. A. F. (2011). 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. https://doi.org/10.1002/spe

Chaloemwat, W., & Kitisin, S. (2016). Horizontal auto-scaling and process migration mechanism for cloud services with skewness algorithm. *2016 13th International Joint Conference on Computer Science and Software Engineering, JCSSE 2016*, 0-5. https://doi.org/10.1109/JCSSE.2016.7748936

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A Library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 1-39. https://doi.org/10.1145/1961189.1961199

Chaurasia, N., Kumar, M., Chaudhry, R., & Verma, O. P. (2021). Comprehensive survey on energy-aware server consolidation techniques in cloud computing. *Journal of Supercomputing*, *77*(10), 11682-11737. https://doi.org/10.1007/s11227-021-03760-1

Chaurasia, N., Kumar, M., Vidyarthi, A., Pal, K., & Alkhayyat, A. (2023). An efficient and optimized Markov chain-based prediction for server consolidation in cloud environment. *Computers and Electrical Engineering*, *108*. https://doi.org/10.1016/j.compeleceng.2023.108707

Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S., & Kapil, D. (2017). A critical survey of live virtual machine migration techniques. *Journal of Cloud Computing*, *6*(1), 1-41. https://doi.org/10.1186/s13677-017-0092-1

Chouliaras, S., & Sotiriadis, S. (2022). Auto-scaling containerized cloud applications: A workload-driven approach. *Simulation Modelling Practice and Theory*, *121*. https://doi.org/10.1016/j.simpat.2022.102654

Cordingly, R., Shu, W., & Lloyd, W. J. (2020). Predicting Performance and Cost of Serverless Computing Functions with SAAF. *Proceedings - IEEE 18th International Conference on Dependable, Autonomic and Secure Computing, IEEE 18th International Conference on Pervasive Intelligence and Computing, IEEE 6th International Conference on Cloud and Big Data Computing and IEEE 5th Cybe*, 640-649. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00111

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197. https://doi.org/10.1109/4235.996017

Devi, K. L., & Valli, S. (2023). Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing*, *105*(2), 353-374. https://doi.org/10.1007/s00607-022-01129-7

Dias, A. H. T., Correia, L. H. A., & Malheiros, N. (2022). A Systematic Literature Review on Virtual Machine Consolidation. *ACM Computing Surveys*, *54*(8). https://doi.org/10.1145/3470972

Ding, W., Luo, F., Han, L., Gu, C., Lu, H., & Fuentes, J. (2020). Adaptive virtual machine consolidation framework based on performance-to-power ratio in cloud data centers. *Future Generation Computer Systems*, *111*, 254-270. https://doi.org/10.1016/j.future.2020.05.004

Dogani, J., Khunjush, F., & Seydali, M. (2023). Host load prediction in cloud computing with Discrete Wavelet Transformation (DWT) and Bidirectional Gated Recurrent Unit (BiGRU) network. *Computer Communications*, *198*, 157-174. https://doi.org/10.1016/j.comcom.2022.11.018

Duong-Ba, T. H., Nguyen, T., Bose, B., & Tran, T. T. (2018). A Dynamic Virtual Machine Placement and Migration Scheme For Data Centers. *IEEE Transactions on Services Computing*, *1374*(c), 1-14. https://doi.org/10.1109/TSC.2018.2817208

El Mensoum, I., Wahab, O. A., Kara, N., & Edstrom, C. (2020). MuSC: A multi-stage service chains embedding approach. *Journal of Network and Computer Applications*, *159*(April 2019), 102593. https://doi.org/10.1016/j.jnca.2020.102593

Emmerich, M. T. M., & Deutz, A. H. (2018). A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing*, *17*(3), 585-609. https://doi.org/10.1007/s11047-018-9685-y

Eramo, V., Miucci, E., Ammar, M., & Lavacca, F. G. (2017). An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking*, *25*(4), 2008-2025. https://doi.org/10.1109/TNET.2017.2668470

Farahnakian, F., Pahikkala, T., Liljeberg, P., Plosila, J., Hieu, N. T., & Tenhunen, H. (2019). Energy-aware VM consolidation in cloud data centers using utilization prediction model. *IEEE Transactions on Cloud Computing*, *7*(2), 524-536. https://doi.org/10.1109/TCC.2016.2617374

Gil Herrera, J., & Botero, J. F. (2016). Resource Allocation in NFV: A Comprehensive Survey. *IEEE Transactions on Network and Service Management*, *13*(3), 518-532. https://doi.org/10.1109/TNSM.2016.2598420

Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., … Uhlig, S. (2022). AI for next generation computing: Emerging trends and future directions. *Internet of Things (Netherlands)*, *19*(March), 100514. https://doi.org/10.1016/j.iot.2022.100514

H. Sayadnavard, M., Toroghi Haghighat, A., & Rahmani, A. M. (2022). A multi-objective approach for energy-efficient and reliable dynamic VM consolidation in cloud data centers. *Engineering Science and Technology, an International Journal*, *26*, 100995. https://doi.org/10.1016/j.jestch.2021.04.014

Hariharan, B., Siva, R., Kaliraj, S., & Prakash, P. N. S. (2023). ABSO: an energy-efficient multi-objective VM consolidation using adaptive beetle swarm optimization on cloud environment. *Journal of Ambient Intelligence and Humanized Computing*, *14*(3), 2185-2197. https://doi.org/10.1007/s12652-021-03429-w

He, S., Hu, C., Shi, B., Wo, T., & Li, B. (2016). Optimizing virtual machine live migration without shared storage in hybrid clouds. *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 921-928. https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0132

He, T. Z., Toosi, A. N., & Buyya, R. (2021). SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds. *Journal of Systems and Software*, *176*, 110943. https://doi.org/10.1016/j.jss.2021.110943

Helali, L., & Omri, M. N. (2021a). A survey of data center consolidation in cloud computing systems. *Computer Science Review*, *39*, 100366. https://doi.org/10.1016/j.cosrev.2021.100366

Helali, L., & Omri, M. N. (2021b). A survey of data center consolidation in cloud computing systems. *Computer Science Review*, *39*, 100366. https://doi.org/10.1016/j.cosrev.2021.100366

Hieu, N. T., Francesco, M. Di, & Yla-Jaaski, A. (2020). Virtual Machine Consolidation with Multiple Usage Prediction for Energy-Efficient Cloud Data Centers. *IEEE Transactions on Services Computing*, *13*(1), 186-199. https://doi.org/10.1109/TSC.2017.2648791

Hirashima, Y. (2016). Parameter Optimization for Hybrid Auto-scaling Mechanism, 111-116.

Hirashima, Y., Yamasaki, K., & Nagura, M. (2016). Proactive-reactive auto-scaling mechanism for unpredictable load change. *Proceedings - 2016 5th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2016*, 861-866. https://doi.org/10.1109/IIAI-AAI.2016.180

Hoseinyfarahabady, M. R., Taheri, J., Zomaya, A. Y., & Tari, Z. (2021). Data-Intensive Workload Consolidation in Serverless (Lambda/FaaS) Platforms. *2021 IEEE 20th International Symposium on Network Computing and Applications, NCA 2021*. https://doi.org/10.1109/NCA53618.2021.9685244

Houidi, O., Soualah, O., Louati, W., Mechtri, M., Zeghlache, D., & Kamoun, F. (2017). An Efficient Algorithm for Virtual Network Function Scaling. *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, *2018-Janua*, 1-7. https://doi.org/10.1109/GLOCOM.2017.8254727

Hsieh, S. Y., Liu, C. S., Buyya, R., & Zomaya, A. Y. (2020). Utilization-prediction-aware virtual machine consolidation approach for energy-efficient cloud data centers. *Journal of Parallel and Distributed Computing*, *139*, 99-109. https://doi.org/10.1016/j.jpdc.2019.12.014

Hu, Y., Bo, D., & Fuyang, P. (2016). Autoscaling prediction models for cloud resource provisioning. *2016 2nd IEEE International Conference on Computer and Communications, ICCC 2016 - Proceedings*, 1364-1369. https://doi.org/10.1109/CompComm.2016.7924927

Huang, F., Li, H., Yuan, Z., & Li, X. (2017). An Application Deployment Approach Based on Hybrid Cloud. *Proceedings - 3rd IEEE International Conference on Big Data Security on Cloud, BigDataSecurity 2017, 3rd IEEE International Conference on High Performance and Smart Computing, HPSC 2017 and 2nd IEEE International Conference on Intelligent Data and Securit*, 74-79. https://doi.org/10.1109/BigDataSecurity.2017.54

Huang, G., Wang, S., Zhang, M., Li, Y., Qian, Z., Chen, Y., & Zhang, S. (2016). Auto scaling virtual machines for web applications with queueing theory. *2016 3rd International Conference on Systems and Informatics, ICSAI 2016*, (Icsai), 433-438. https://doi.org/10.1109/ICSAI.2016.7810994

Islam, M. R., Saifullah, C. M. K., & Mahmud, M. R. (2019). Chemical reaction optimization: survey on variants. *Evolutionary Intelligence*, *12*(3), 395-420. https://doi.org/10.1007/s12065-019-00246-1

Janjanam, T. S., Siram, K. S., & Kollu, P. K. (2023). Cloud Resources Forecasting based on Server Workload using ML Techniques. Dans *IDCIoT 2023 - International Conference on Intelligent Data Communication Technologies and Internet of Things, Proceedings* (pp. 427-433). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/IDCIoT56793.2023.10053532

Jeong, B., Baek, S., Park, S., Jeon, J., & Jeong, Y. S. (2023). Stable and efficient resource management using deep neural network on cloud computing. *Neurocomputing*, *521*, 99-112. https://doi.org/10.1016/j.neucom.2022.11.089

Jia, Y., Wu, C., Li, Z., Le, F., & Liu, A. (2018). Online Scaling of NFV Service Chains Across Geo-Distributed Datacenters. *IEEE/ACM Transactions on Networking*, *26*(2), 699-710. https://doi.org/10.1109/TNET.2018.2800400

Junjie Liu Fen Zhou, Ping Lu, Zuqing Zhu, W. L. (2017). On Dynamic Service Function Chain Deployment and Readjustment. *Ieee Transactions on Network and Service Management*, *14*(3), 543-553.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, Transactions of the ASME*, *82*(1), 35-45. https://doi.org/10.1115/1.3662552

Kalyvianaki, E., Charalambous, T., & Hand, S. (2014). Adaptive resource provisioning for virtualized servers using kalman filters. *ACM Transactions on Autonomous and Adaptive Systems*, *9*(2). https://doi.org/10.1145/2626290

Kan, C. (2016). Docloud: an elastic cloud platform for web applications based on Docker. *2016 18th International Conference on Advanced Communication Technology (ICACT)*, 478-483.

Khan, T., Tian, W., Ilager, S., & Buyya, R. (2022). Workload forecasting and energy state estimation in cloud data centres: ML-centric approach. *Future Generation Computer Systems*, *128*, 320-332. https://doi.org/10.1016/j.future.2021.10.019

Khan, T., Tian, W., Zhou, G., Ilager, S., Gong, M., & Buyya, R. (2022). Machine learning (ML)-centric resource management in cloud computing: A review and future directions. *Journal of Network and Computer Applications*, *204*(March), 103405.

Khebbache, S., Hadji, M., & Zeghlache, D. (2018). A multi-objective non-dominated sorting genetic algorithm for VNF chains placement. *CCNC 2018 - 2018 15th IEEE Annual Consumer Communications and Networking Conference*, *2018-Janua*, 1-4. https://doi.org/10.1109/CCNC.2018.8319250

Khoshkholghi, M. A., Derahman, M. N., Abdullah, A., Subramaniam, S., & Othman, M. (2017). Energy-Efficient Algorithms for Dynamic Virtual Machine Consolidation in Cloud Data Centers. *IEEE Access*, *5*, 10709-10722. https://doi.org/10.1109/ACCESS.2017.2711043

Kohne, A., Pasternak, D., Nagel, L., & Spinczyk, O. (2016). Evaluation of SLA-based decision strategies for VM scheduling in cloud data centers. *3rd Workshop on CrossCloud Infrastructures and Platforms, CrossCloud 2016 - Colocated with EuroSys 2016*, *1*(212). https://doi.org/10.1145/2904111.2904113

Kohne, A., Spohr, M., Nagel, L., & Spinczyk, O. (2014). FederatedCloudSim: A SLA-aware federated cloud simulation framework. *Proceedings of the 2nd International Workshop on Cross-Cloud Systems, CrossCloud Brokers 2014 - Held in conjunction with the 15th ACM/IFIP/USENIX International Middleware Conference, Middleware 2014*. https://doi.org/10.1145/2676662.2676674

Laaziz, L., Kara, N., Rabipour, R., Edstrom, C., & Lemieux, Y. (2019). FASTSCALE: A fast and scalable evolutionary algorithm for the joint placement and chaining of virtualized services. *Journal of Network and Computer Applications*, *148*(July). https://doi.org/10.1016/j.jnca.2019.102429

Lam, A. Y. S., & Li, V. O. K. (2012). Chemical Reaction Optimization: A tutorial. *Memetic Computing*, *4*(1), 3-17. https://doi.org/10.1007/s12293-012-0075-1

Leivadeas, A., Papagianni, C., & Papavassiliou, S. (2015). Going Green with the Networked Cloud: Methodologies and Assessment. Dans *Quantitative Assessments of Distributed Systems: Methodologies and Techniques* (pp. 351-374). (S.l.) : (s.n.).

Level, S. (2016). An Efficient Resource Utilization Technique for Consolidation of Virtual Machines in Cloud Computing Environments. *33 rd NATIONAL RADIO SCIENCE CONFERENCE Nrsc*, 316-324.

Li, L., Dong, J., Zuo, D., & Liu, J. (2018). SLA-aware and energy-efficient VM consolidation in cloud data centers using host states naive Bayesian prediction model. Dans *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)* (pp. 80-87). IEEE. https://doi.org/10.1109/BDCloud.2018.00025

Li, L., Dong, J., Zuo, D., & Wu, J. (2019). SLA-Aware and Energy-Efficient VM Consolidation in Cloud Data Centers Using Robust Linear Regression Prediction Model. *IEEE Access*, *7*, 9490-9500. https://doi.org/10.1109/ACCESS.2019.2891567

Li, Y., & Xia, Y. (2017). Auto-scaling web applications in hybrid cloud based on docker. *Proceedings of 2016 5th International Conference on Computer Science and Network Technology, ICCSNT 2016*, 75-79. https://doi.org/10.1109/ICCSNT.2016.8070122

Li, Zhihua, Yu, X., Yu, L., Guo, S., & Chang, V. (2020). Energy-efficient and quality-aware VM consolidation method. *Future Generation Computer Systems*, *102*, 789-809. https://doi.org/10.1016/j.future.2019.08.004

Li, Zhiyong, Li, Y., Yuan, T., Chen, S., & Jiang, S. (2019). Chemical reaction optimization for virtual machine placement in cloud computing. *Applied Intelligence*, *49*(1), 220-232. https://doi.org/10.1007/s10489-018-1264-5

Lin, W., Xu, S., He, L., & Li, J. (2017). Multi-resource scheduling and power simulation for cloud computing. *Information Sciences*, *397-398*, 168-186. https://doi.org/10.1016/j.ins.2017.02.054

Liu, J., Lu, W., Zhou, F., Lu, P., & Zhu, Z. (2017). On Dynamic service function chain deployment and readjustment. *IEEE Transactions on Network and Service Management*, *14*(3), 543-553. https://doi.org/10.1109/TNSM.2017.2711610

Luo, S., Xu, H., Ye, K., Xu, G., Zhang, L., Yang, G., & Xu, C. (2022). The Power of Prediction: Microservice Auto Scaling via Workload Learning. *SoCC 2022 - Proceedings of the 13th Symposium on Cloud Computing*, 355-369. https://doi.org/10.1145/3542929.3563477

Luo, Z., & Wu, C. (2020). An online algorithm for VNF service chain scaling in datacenters. *IEEE/ACM Transactions on Networking*, *28*(3), 1061-1073. https://doi.org/10.1109/TNET.2020.2979263

Mahdhi, T., & Mezni, H. (2018). A prediction-Based VM consolidation approach in IaaS Cloud Data Centers. *Journal of Systems and Software*, *146*, 263-285. https://doi.org/10.1016/j.jss.2018.09.083

Mai, L., Ding, Y., Zhang, X., Fan, L., Yu, S., & Xu, Z. (2021). Energy efficiency with service availability guarantee for Network Function Virtualization. *Future Generation Computer Systems*, *119*, 140-153. https://doi.org/10.1016/j.future.2021.02.002

Malik, S., Tahir, M., Sardaraz, M., & Alourani, A. (2022). A Resource Utilization Prediction Model for Cloud Data Centers Using Evolutionary Algorithms and Machine Learning Techniques. *Applied Sciences*, *12*(4), 2160. https://doi.org/10.3390/app12042160

Masdari, M., & Khoshnevis, A. (2020). A survey and classification of the workload forecasting methods in cloud computing. *Cluster Computing*, *23*(4), 2399-2424. https://doi.org/10.1007/s10586-019-03010-3

McGrath, G., & Brenner, P. R. (2017). Serverless Computing: Design, Implementation, and Performance. *Proceedings - IEEE 37th International Conference on Distributed Computing Systems Workshops, ICDCSW 2017*, 405-410. https://doi.org/10.1109/ICDCSW.2017.36

Medhat, A. M., Taleb, T., Elmangoush, A., Carella, G. A., Covaci, S., & Magedanz, T. (2016). Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges, 2-9.

Melhem, S. B., Agarwal, A., Goel, N., & Zaman, M. (2017). Selection process approaches in live migration: A comparative study. *2017 8th International Conference on Information and Communication Systems, ICICS 2017*, 23-28. https://doi.org/10.1109/IACS.2017.7921940

Melin, P., Astudillo, L., Castillo, O., Valdez, F., & Garcia, M. (2013). Optimal design of type-2 and type-1 fuzzy tracking controllers for autonomous mobile robots under perturbed torques using a new chemical optimization paradigm. *Expert Systems with Applications*, *40*(8), 3185-3195. https://doi.org/10.1016/j.eswa.2012.12.032

Meng, Y., Rao, R., Zhang, X., & Hong, P. (2016). CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis. *PIC 2016 - Proceedings of the 2016 IEEE International Conference on Progress in Informatics and Computing*, 468-472. https://doi.org/10.1109/PIC.2016.7949546

Minarolli, D., Mazrekaj, A., & Freisleben, B. (2017). Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing. *Journal of Cloud Computing*, *6*(1). https://doi.org/10.1186/s13677-017-0074-3

Moghaddam, S. M., Piraghaj, S. F., O'Sullivan, M., Walker, C., & Unsworth, C. P. (2018). Energy-efficient and SLA-aware virtual machine selection algorithm for dynamic resource allocation in cloud data centers. *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2018*, 103-113. https://doi.org/10.1109/UCC.2018.00019

Moghaddassian, M., Bannazadeh, H., & Leon-Garcia, A. (2017). Adaptive auto-scaling for virtual resources in software-defined infrastructure. *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, 548-551. https://doi.org/10.23919/INM.2017.7987326

Mostafavi, S., Hakami, V., & Sanaei, M. (2021). *Quality of service provisioning in network function virtualization: a survey*. *Computing* (Vol. 103). (S.l.) : Springer Vienna. https://doi.org/10.1007/s00607-021-00925-x

Nadgowda, S., Suneja, S., Bila, N., & Isci, C. (2017). Voyager: Complete Container State Migration. *Proceedings - International Conference on Distributed Computing Systems*, (Section III), 2137-2142. https://doi.org/10.1109/ICDCS.2017.91

Nadgowda, S., Suneja, S., & Kanso, A. (2017). Comparing scaling methods for linux containers. *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*, 266-272. https://doi.org/10.1109/IC2E.2017.42

Nadjaran Toosi, A., Son, J., Chi, Q., & Buyya, R. (2019). ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds. *Journal of Systems and Software*, *152*, 108-119. https://doi.org/10.1016/j.jss.2019.02.052

Nath, S. B., Addya, S. K., Chakraborty, S., & Ghosh, S. K. (2020). Green Containerized Service Consolidation in Cloud. *IEEE International Conference on Communications*, *2020-June*. https://doi.org/10.1109/ICC40277.2020.9149173

Nezamabadi-Pour, H., Rostami-Shahrbabaki, M., & Maghfoori-Farsangi, M. M. (2008). Binary Particle Swarm Optimization: challenges and New Solutions. *The Journal of Computer Society of Iran (CSI) On Computer Science and Engineering (JCSE)*, *6*(May 2014), 21-32. Repéré à https://www.researchgate.net/publication/258456389

Nguyen, T. T., Li, Z., Zhang, S., & Truong, T. K. (2014). A hybrid algorithm based on particle swarm and chemical reaction optimization. *Expert Systems with Applications*, *41*(5), 2134-2143. https://doi.org/10.1016/j.eswa.2013.09.012

Noshy, M., Ibrahim, A., & Ali, H. A. (2018). Optimization of live virtual machine migration in cloud computing: A survey and future directions. *Journal of Network and Computer Applications*, *110*(March), 1-10. https://doi.org/10.1016/j.jnca.2018.03.002

Noureddine, A. (2022). PowerJoular and JoularJX : Multi-Platform Software Power Monitoring Tools. Dans *18th International Conference on Intelligent Environments*. Biarritz, France.

Olivas, F., Valdez, F., Castillo, O., & Melin, P. (2016). Dynamic parameter adaptation in particle swarm optimization using interval type-2 fuzzy logic. *Soft Computing*, *20*(3), 1057-1070. https://doi.org/10.1007/s00500-014-1567-3

Olivas, F., Valdez, F., Melin, P., Sombra, A., & Castillo, O. (2019). Interval type-2 fuzzy logic for dynamic parameter adaptation in a modified gravitational search algorithm. *Information Sciences*, *476*, 159-175. https://doi.org/10.1016/j.ins.2018.10.025

Panwar, S. S., Rauthan, M. M. S., & Barthwal, V. (2022). A systematic review on effective energy utilization management strategies in cloud data centers. *Journal of Cloud Computing*, *11*(1). https://doi.org/10.1186/s13677-022-00368-5

Park, K., & Pai, V. S. (2006). CoMon: A Mostly-Scalable Monitoring System for PlanetLab. *ACM SIGOPS Operating Systems Review*, *40*(1), 65-74. https://doi.org/10.1145/1113361.1113374

Qiu, F., Zhang, B., & Guo, J. (2016). A deep learning approach for VM workload prediction in the cloud. *2016 IEEE/ACIS 17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2016*, 319-324. https://doi.org/10.1109/SNPD.2016.7515919

Radhika, E. G., & Sadasivam, G. S. (2021). A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment. *Materials Today: Proceedings*, *45*, 2793-2800. https://doi.org/10.1016/j.matpr.2020.11.789

Rahman, S., Ahmed, T., Huynh, M., Tornatore, M., & Mukherjee, B. (2018). Auto-Scaling VNFs Using Machine Learning to Improve QoS and Reduce Cost. *IEEE International Conference on Communications ICC*, *2018-May*, 1-6. https://doi.org/10.1109/ICC.2018.8422788

Rajan, R. A. P. (2018). Serverless Architecture - A Revolution in Cloud Computing. *2018 10th International Conference on Advanced Computing, ICoAC 2018*, 88-93. https://doi.org/10.1109/ICoAC44903.2018.8939081

Rankothge, W., Le, F., Russo, A., & Lobo, J. (2017). Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms. *IEEE Transactions on Network and Service Management*, *14*(2), 343-356. https://doi.org/10.1109/TNSM.2017.2686979

Rankothge, W., Ramalhinho, H., & Lobo, J. (2019). On the scaling of virtualized network functions. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, 125-133.

Rodriguez, E., Alkmim, G. P., Da Fonseca, N. L. S., & Batista, D. M. (2017). Energy-Aware Mapping and Live Migration of Virtual Networks. *IEEE Systems Journal*, *11*(2), 637-648. https://doi.org/10.1109/JSYST.2015.2467159

Saha, A., & Jindal, S. (2018). EMARS: Efficient Management and Allocation of Resources in Serverless. *IEEE International Conference on Cloud Computing, CLOUD*, *2018-July*, 827-830. https://doi.org/10.1109/CLOUD.2018.00113

Santhosh, S., & Binu, A. (2016). Auto scaling for various patterns of workflow within deadline time and energy aware VM allocation in cloud environment. *Proceedings of the 2016 International Conference on Data Science and Engineering, ICDSE 2016*, 0-4. https://doi.org/10.1109/ICDSE.2016.7823941

Schardong, F., Nunes, I., & Schaeffer-Filho, A. (2021). NFV Resource Allocation: a Systematic Review and Taxonomy of VNF Forwarding Graph Embedding. *Computer Networks*, *185*(July 2020), 107726. https://doi.org/10.1016/j.comnet.2020.107726

Shao, Y., Yang, Q., Gu, Y., Pan, Y., Zhou, Y., & Zhou, Z. (2020). A Dynamic Virtual Machine Resource Consolidation Strategy Based on a Gray Model and Improved Discrete Particle Swarm Optimization. *IEEE Access*, *8*, 228639-228654. https://doi.org/10.1109/ACCESS.2020.3046318

Shariffdeen, R. S., Munasinghe, D. T. S. P., Bhathiya, H. S., Bandara, U. K. J. U., & Dilum Bandara, H. M. N. (2016). Workload and resource aware proactive auto-scaler for PaaS cloud. *IEEE International Conference on Cloud Computing, CLOUD*, 11-18. https://doi.org/10.1109/CLOUD.2016.10

Shen, S., Van Beek, V., & Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters. *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*, 465-474. https://doi.org/10.1109/CCGrid.2015.60

Siddique, N., & Adeli, H. (2017). Nature-Inspired Chemical Reaction Optimisation Algorithms. *Cognitive Computation*, *9*(4), 411-422. https://doi.org/10.1007/s12559-017-9485-1

Silva Filho, M. C., Monteiro, C. C., Inácio, P. R. M., & Freire, M. M. (2018). Approaches for optimizing virtual machine placement and migration in cloud environments: A survey. *Journal of Parallel and Distributed Computing*, *111*, 222-250. https://doi.org/10.1016/j.jpdc.2017.08.010

Singh, P., Gupta, P., Jyoti, K., & Nayyar, A. (2019). Research on auto-scaling of web applications in cloud: Survey, trends and future directions. *Scalable Computing*, *20*(2), 399-432. https://doi.org/10.12694/scpe.v20i2.1537

Sohan, R., Rice, A., Moore, A. W., & Mansley, K. (2010). Characterizing 10 Gbps Network Interface Energy Consumption Abstract—This paper quantifies the energy consumption in six 10 Gbps and four 1 Gbps interconnects at a fine-grained level, introducing two metrics for calculating the energy efficiency of a netw. *IEEE Local Computer Network Conference*, 268-271. Repéré à https://www.cl.cam.ac.uk/~acr31/pubs/sohan-10gbpower.pdf

Songara, N., & Jain, M. K. (2023). MRA-VC: multiple resources aware virtual machine consolidation using particle swarm optimization. *International Journal of Information Technology (Singapore)*, *15*(2), 697-710. https://doi.org/10.1007/s41870-022-01102-9

Sotiriadis, S., Bessis, N., Amza, C., & Buyya, R. (2016). Vertical and horizontal elasticity for dynamic virtual machine reconfiguration. *IEEE Transactions on Services Computing*, *1374*(c), 1-1. https://doi.org/10.1109/TSC.2016.2634024

St-Onge, C., Benmakrelouf, S., Kara, N., Tout, H., Edstrom, C., & Rabipour, R. (2021). Generic SDE and GA-based workload modeling for cloud systems. *Journal of Cloud Computing*, *10*(1). https://doi.org/10.1186/s13677-020-00223-5

Taherizadeh, S., & Stankovski, V. (2018). Dynamic Multi-level Auto-scaling Rules for Containerized Applications. *The Computer Journal*, (June). https://doi.org/10.1093/comjnl/bxy043

Tavakoli-Someh, S., & Rezvani, M. H. (2019). *Multi-objective virtual network function placement using NSGA-II meta-heuristic approach*. *Journal of Supercomputing* (Vol. 75). (S.l.) : Springer US. https://doi.org/10.1007/s11227-019-02849-y

The SPECpower Benchmark. (s.d.). *http://www.spec.org/power_ssj2008/*.

Valade, A., Acco, P., Grabolosa, P., & Fourniols, J. Y. (2017). A study about kalman filters applied to embedded sensors. *Sensors (Switzerland)*, *17*(12), 1-18. https://doi.org/10.3390/s17122810

Wahab, O. A., Kara, N., Edstrom, C., & Lemieux, Y. (2019). MAPLE: A Machine Learning Approach for Efficient Placement and Adjustment of Virtual Network Functions. *Journal of Network and Computer Applications*, *142*(October 2018), 37-50. https://doi.org/10.1016/j.jnca.2019.06.003

Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*(2), 387-408. https://doi.org/10.1007/s00500-016-2474-6

Wang, X., Wu, C., Le, F., Liu, A., Li, Z., & Lau, F. (2016). Online VNF scaling in datacenters. *IEEE International Conference on Cloud Computing, CLOUD*, (1), 140-147. https://doi.org/10.1109/CLOUD.2016.26

Witanto, J. N., Lim, H., & Atiquzzaman, M. (2018). Adaptive selection of dynamic VM consolidation algorithm using neural network for cloud resource management. *Future Generation Computer Systems*, *87*, 35-42. https://doi.org/10.1016/j.future.2018.04.075

Xiao, H., Hu, Z., & Li, K. (2019). Multi-objective vm consolidation based on thresholds and ant colony system in cloud computing. *IEEE Access*, *7*, 53441-53453. https://doi.org/10.1109/ACCESS.2019.2912722

Xie, Y., Jin, M., Zou, Z., Xu, G., Feng, D., Liu, W., & Long, D. (2022). Real-Time Prediction of Docker Container Resource Load Based on a Hybrid Model of ARIMA and Triple Exponential Smoothing. *IEEE Transactions on Cloud Computing*, *10*(2), 1386-1401. https://doi.org/10.1109/TCC.2020.2989631

Yadav, R., Zhang, W., Li, K., Liu, C., & Laghari, A. A. (2021). Managing overloaded hosts for energy-efficiency in cloud data centers. *Cluster Computing*, *24*(3), 2001-2015. https://doi.org/10.1007/s10586-020-03182-3

Yang, S., Li, F., Trajanovski, S., Yahyapour, R., & Fu, X. (2021). Recent Advances of Resource Allocation in Network Function Virtualization. *IEEE Transactions on Parallel and Distributed Systems*, *32*(2), 295-314. https://doi.org/10.1109/TPDS.2020.3017001

Ye, T., Guangtao, X., Shiyou, Q., & Minglu, L. (2017). An Auto-Scaling Framework for Containerized Elastic Applications. *Proceedings - 2017 3rd International Conference on Big Data Computing and Communications, BigCom 2017*, 422-430. https://doi.org/10.1109/BIGCOM.2017.40

Yi, B., Wang, X., & Huang, M. (2017). Design and evaluation of schemes for provisioning service function chain with function scalability. *Journal of Network and Computer Applications*, *93*(June), 197-214. https://doi.org/10.1016/j.jnca.2017.05.013

Yi, B., Wang, X., Li, K., Das, S. k., & Huang, M. (2018). A comprehensive survey of Network Function Virtualization. *Computer Networks*, *133*, 212-262. https://doi.org/10.1016/j.comnet.2018.01.021

Zhang-Jian, D.-J., Lee, C.-N., & Hwang, R.-H. (2013). An energy-saving algorithm for cloud resource management using a Kalman filte. *International Journal of Communication Systems*, *27*, 4078-4091. https://doi.org/10.1002/dac

Zhang, F., Fu, X., & Yahyapour, R. (2017). CBase: A new paradigm for fast virtual machine migration across data centers. *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, 284-293. https://doi.org/10.1109/CCGRID.2017.26

Zhang, F., Liu, G., Fu, X., & Yahyapour, R. (2018). A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues, *20*(2), 1206-1243. https://doi.org/10.1109/COMST.2018.2794881

Zhang, Q., Chen, H., & Yin, Z. (2017). PRMRAP: A Proactive Virtual Resource Management Framework in Cloud. *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, 120-127. https://doi.org/10.1109/IEEE.EDGE.2017.24

Zhao, S. (2023). Energy efficient resource allocation method for 5G access network based on reinforcement learning algorithm. *Sustainable Energy Technologies and Assessments*, *56*(April 2022), 103020. https://doi.org/10.1016/j.seta.2023.103020

Zhou, Q., Xu, M., Singh Gill, S., Gao, C., Tian, W., Xu, C., & Buyya, R. (2020). Energy Efficient Algorithms based on VM Consolidation for Cloud Computing: Comparisons and Evaluations. *Proceedings - 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020*, 489-498. https://doi.org/10.1109/CCGrid49817.2020.00-44

Zolfaghari, R., & Rahmani, A. M. (2020). *Virtual Machine Consolidation in Cloud Computing Systems: Challenges and Future Trends*. *Wireless Personal Communications* (Vol. 115). (S.l.) : Springer US. https://doi.org/10.1007/s11277-020-07682-8